Blocks Programming Reference Manual
By Bruce Schafer
February 7, 2019

Table of Contents

Introduction

This reference manual provides brief descriptions of each feature of FTC Blocks Programming. Because there are so many features, attempting to read this manual can be overwhelming. It helps to remember that reference manuals are not meant to be read like a tutorial or a training manual. Rather, it should be used more like a dictionary where you can lookup features as needed.

Before you attempt to use this reference document we recommend you read the Bocks Programming Training Manual. It can be found linked to
http://www.firstinspires.org/resource-library/ftc/technology-information-and-resources

You should also consider reviewing the FTC YouTube videos available on the following play list:
https://www.youtube.com/playlist?list=PLEuGrYl8iBm4A4yrRcatGcK7q0od0LYov

Much of this reference document is based on the “roll-over” documentation that is available when you move the cursor over blocks while you use a web browser to access the features of the Blocks Programming.

Values and objects have types

Because Blocks programing is built “on top of” Java, the values or objects produced by blocks have types. The type may be simple like numeric value (a number) or complex like a Vuforia tracking results object. Other blocks require values or objects of a particular type when you plug a block into their right side. If you attempt to plug a block that produces one type into a block that requires another, the Blocks editor will not allow the block to attach. If, however, you use a set-variable block to store the value or object in a variable and then plug that variable into another block, the Blocks editor will not know what type of value or object that the variable contains and will allow the variable to be attached. If the variable contains the wrong type of value or object when your op mode executes, no value will be provided to the block and an error will be added to the log (see below).

Printing an op mode

There are three ways to print an op mode program using Windows. In all three cases you start by displaying the op mode using your web browser.

(1) Click on "Download Image of Blocks" button toward the top. Save the PNG image file on your computer. If you have a Windows computer, select Paint in the Windows Accessories sub-menu. From Paint open the PNG file. Use File-Print-Page Setup and then use the Scaling feature to select enough pages to assure the blocks in your op mode will be large enough to read. For instance, if your op mode is fairly long, you may need to be spread it across five pages by specifying "Fit to: 1 by 5 pages." Then use File-Print to print the image. If your printer is on your Local Area Network (LAN), you'll need to disconnect the WiFi connection between your Windows computer and your phone and connect your computer to your LAN before you request the Print.

(2) If you have a printer directly connected to your computer, then print it using the browser's print feature (usually in the File menu). If your printer is on your Local Area Network (LAN), you'll need to disconnect the WiFi connection between your Windows computer and your phone and connect your computer to your LAN. Then print the program from your browser. If your browser doesn't print the whole program, try clicking the zoom-out (minus) symbol in the lower right part of the browser screen to get all or most of the program on the screen and then print again.

(3) Use the PrtSc key on your keyboard to capture a screen shot. Then start the Windows Paint program. (You'll find it under Windows Accessories in the main Windows menu.) In Paint use the Paste button from the Home menu or just press control-V to place the screen shot on the Paint canvas. Then use Paint's File Print to print the screen shot. (Per above you may need to switch your computer from being connected to your RC phone to being connected to your LAN to do this.) If you didn't capture the whole program in the screen shot, go back to the web browser, scroll down and repeat the process. (You probably won't need re-connect to your RC phone to scroll the op mode already displayed by your browser.)

Reviewing the Robot Controller log

When the Robot Controller app is running it will add lines to a log. To access the log, touch the three dots in the upper right corner of the Robot Controller app and then select Settings then select View Logs.

This log is a good place to look for hints if your op mode appears to be misbehaving. In many cases an error message will be displayed on your Driver Station phone when a problem is detected in your program. More details will usually be available in the log. For example, if your op mode includes a block that uses a variable that contains the wrong type of object you should be able to find a line in the log describing what happened. In this case the message might be something like "RobotCore: MatrixFtoText matrix is not a MatrixF."

Format of this manual

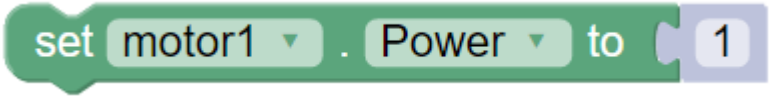
The rest this document describes each block in two ways:

(a) The format of the block. To save space the actual blocks are not given in this manual. One or more lines that mirror the text of that block are used instead. When the text of block changes with the name of an actuator or sensor that variable portion is denoted by <… name> where "…" is replaced by the type of actuator or sensor, e.g. <gyro name>. When the block requires that one or more other blocks be plugged into it, the type of those other blocks are indicated by text enclosed in angle brackets, e.g. <numeric value>. When a block returns a value or an object (as indicated by a tab on its left side) the description of the block in this document starts with an indication of the type of the returned value/object in angle brackets and then a left arrow, e.g. <vector object> ←.

(b) A description of the effect of the block enclosed.

Here are some examples of how the formats of the blocks are represented:

(a) If we have a motor named "motor1" we could have a block like this:



This type of block would be represented in this document as
set <motor name>.Power to <numeric value>

(b) If we have a color sensor named "color1" we could have a block like this which returns the amount of red detected by the sensor:

This type of block would be represented in this document as

<numeric value> ← <color sensor name>.Red

LinearOpMode

These blocks allow you to control and monitor the execution of an op mode.

call <op mode name>.waitForStart

Description: Wait until Start is selected at Driver Station

call <op mode name>.idle

Description: Put this thread to sleep for a bit allowing other threads to run

call <op mode name>.sleep milliseconds

Description: Delay for specified number of milliseconds

<logic value> ← call <op mode name>.opModeIsActive

Description: Return true if op mode is active

<logic value> ← call <op mode name>.isStarted

Description: Return true if Start has been selected

<logic value> ← call <op mode name>.isStopRequested

Description: Returns true if Stop has been selected

<logic value> ← call <op mode name>.getRuntime

Description: Get elapsed time in seconds since op mode was initialized

Gamepad

These blocks in this group allow you add programming elements that connect one or two gamepads to the operation of an op mode. The names of the gamepads are set by editing the config file using the Robot Controller app.

<logic value> ← <gamepad name>.A

Description: Returns true if A button is pressed, false otherwise.

<logic value> ← <gamepad name>.AtRest

Description: Returns true if joys sticks and triggers in neutral position, false otherwise.

<logic value> ← <gamepad name>.B

Description: Returns true if B button is pressed, false otherwise.

<logic value> ← <gamepad name>.Back

Description: Returns true if Back button is pressed, false otherwise.

<logic value> ← <gamepad name>.DpadDown

Description: Returns true if D-pad Down is pressed, false otherwise.

<logic value> ← <gamepad name>.DpadLeft

Description: Returns true if D-pad Left is pressed, false otherwise.

<logic value> ← <gamepad name>.DpadRight

Description: Returns true if D-pad Right is pressed, false otherwise.

<logic value> ← <gamepad name>.DpadUp

Description: Returns true if D-pad Up is pressed, false otherwise

<logic value> ← <gamepad name>.Guide

Description: Returns true if Guide Button is pressed, false otherwise. The Guide is often in the middle of the controller.

<logic value> ← <gamepad name>.LeftBumper

Description: Returns true if the left bumper is pressed, false otherwise.

<logic value> ← <gamepad name>.LeftStickButton

Description: Returns true if the left stick button is pressed, false otherwise.

<numeric value> ← <gamepad name>.LeftStickX

Description: Returns the left-right deflection of the left stick. Negative values represent left deflections and positive right deflections. Range is -1.0 to +1.0.

<numeric value> ← <gamepad name>.LeftStickY

Description: Returns the up-down deflection of the left stick. Negative values represent up deflections and positive values down deflections. Range is -1.0 to +1.0.

<logic value> ← <gamepad name>.LeftTrigger

Description: Returns true if the left trigger is pressed, false otherwise.

<logic value> ← <gamepad name>.RightBumper

Description: Returns true if the right bumper is pressed, false otherwise.

<logic value> ← <gamepad name>.RightStickButton

Description: Returns true if the right stick button is pressed, false otherwise.

<numeric value> ← <gamepad name>.RightStickX

Description: Returns the left-right deflection of the right stick. Negative values represent left deflections and positive right deflections. Range is -1.0 to +1.0.

<numeric value> ← <gamepad name>.RightStickY

Description: Returns the up-down deflection of the right stick. Negative values represent up deflections and positive values down deflections. Range is -1.0 to +1.0.

<logic value> ← <gamepad name>.RightTrigger

Description: Returns true if the right trigger is pressed, false otherwise.

<logic value> ← <gamepad name>.Start

Description: Returns true if the Start trigger is pressed, false otherwise.

<logic value> ← <gamepad name>.X

Description: Returns true if X button is pressed, false otherwise.

<logic value> ← <gamepad name>.Y

Description: Returns true if Y button is pressed, false otherwise.

Actuators

Blocks Programming supports two types of actuators – motors and servos. The blocks associated for a particular motor or sensor appear in toolbox area only if that motor or sensor is described in the configuration file that is currently activated on the Robot Controller phone.

DcMotor

When an op mode initializes, DC motors will have the following attributes until a block executes that changes the attribute.

- Direction: FORWARD
- Mode: RUN_WITHOUT_ENCODER
- Power: 0
- Zero Power Behavior: BRAKE

<position> ← <motor name>.CurrentPosition
Description: Return the position value of the named motor.

Set <motor name>.Direction to <direction>
Description: Set the direction of the named motor to the specified direction. Valid values are Direction.REVERSE and Direction.FORWARD.

<direction> ← <motor name>.Direction
Description: Return the direction of the named motor.

Set <motor name>.MaxSpeed to <numeric value>
Description: Set the maximum speed of the named motor. Speed is given in encoder ticks per second. As of Version 3, this block is no longer supported. Instead, the software keeps track of the maximum speed based on the motor brand and model specified in the configuration file.

<numeric value> ← <motor name>.Maxspeed
Description: Return the current maximum speed of the named motor. As of Version 3 this block is no longer supported and returns zero.

Set <motor name>.Mode to <run mode>

Description: Set the mode of the named motor to the indicated <run mode>. Valid values of <run mode> are

- RunMode.RUN_WITHOUT_ENCODER: Motor is controlled by power supplied to it and actual speed may vary.
- RunMode.RUN_USING_ENCODER: The motor controller will adjust the power provided to the motor using PID control to achieve a particular speed relative to the maximum speed of the motor.
- RunMode.RUN_TO_POSITION: Motor will run until a target position is reached, which is specified as a tick count.
- RunMode.STOP_AND_RESET_ENCODER: Motor is stopped and the encoder value is reset to 0.

The RUN_USING_ENCODER and RUN_TO_POSITION modes cause the motor controller to use a proportional–integral–derivative (PID) algorithm as a feedback mechanism to control the speed of the motor under varying loads by varying the power provided to the motor.

<run mode> ← <motor name>.Mode
Description: Return the current mode of the named motor.

set <motor name>.Power to <numeric value>
Description:

- When the RunMode is RUN_WITHOUT_ENCODER, Set the power of the named motor name to the given numeric value. Valid values of <numeric value> are -1.0 to 1.0, which represent -100% to 100% of full power. Negative values cause the motor to run in the direction opposite to its normal direction.
- When the RunMode is RUN_USING_ENCODER or RUN_TO_POSITION the value provided for power is actually used by the motor controller as the target speed in its PID algorithm to adjust power levels provided to the motor. Valid values of <numeric value> are -1.0 to 1.0, which represent -100% to 100% of maximum speed. The maximum speed is based on the brand and model of the motor specified in your configuration file. Negative numeric values represent speeds in the opposite direction to the normal direction.

<numeric value> ← <motor name>.Power
Description: Return the current power level of the named motor. If the current RunMode is RUN_USING_ENCODER or RUN_TO_POSITION the current target speed will be returned.

<logic value> ← <motor name>.PowerFloat
Description: Return true if the motor power is currently in the float state; false otherwise.

set <motor name>.TargetPosition to <numeric value>
Description: Set the target position of the named motor to the specific numeric value.

<numeric value> ← <motor name>.TargetPosition
Description: Return the current target position of the named motor.

set <motor name>.ZeroPowerBehavior to <zero power behavior>
Description: Set the zero power behavior of the named motor to indicated value. Valid values of <zero power behavior> are

ZeroPowerBehavior.BRAKE: Stop the motor's axle.

ZeroPowerBehavior.FLOAT: Let the motor's axle rotate but don't generate torque

<zero power behavior> ← <motor name>.ZeroPowerBehavior
Description: Return the current zero power behavior value of the named motor.

<logic value> ← call <motor name>.isBusy
Description: Return true if the named motor is busy. False otherwise.

Dual

Because motors are often operated in pairs – particular to drive wheels or tracks, Blocks Programming includes a set of blocks that allow two motors to be controlled by a single block. These blocks only appear in the toolbox area if two or more motors are described in the configuration file on the Robot Controller Android device.

Set Power
<motor name> to <numeric value>
<motor name> to <numeric value>
Description:

- When the RunMode is RUN_WITHOUT_ENCODER, Set the power of the named motor name to the given numeric value. Valid values of <numeric value> are -1.0 to 1.0, which represent -100% to 100% of full power. Negative values cause the motor to run in the direction opposite to its normal direction.
- When the RunMode is RUN_USING_ENCODER or RUN_TO_POSITION the value provided for power is actually used by the motor controller as the target speed in its PID algorithm to adjust power levels provided to the motor. Valid values of <numeric value> are -1.0 to 1.0, which represent -100% to 100% of maximum speed. The maximum speed is based on the brand and model of the motor specified in your configuration file. Negative numeric values represent speeds in the opposite direction to the normal direction.

Set MaxSpeed
<motor name> to <numeric value>
<motor name> to <numeric value>
Description: Set the maximum speeds of the two named motors to the specified speed values. Speed values are given in encoder ticks per second. As of version 3, this block is no longer supported. See MaxSpeed in the section above.

Set Mode
<motor name> to <run mode>
<motor name> to <run mode>

Description: Set the run mode of the named motors. Valid values of <run mode> are

- RunMode.RUN_WITHOUT_ENCODER: Motor is controlled by power supplied to it and actual speed may vary.
- RunMode.RUN_USING_ENCODER: The motor controller will adjust the power provided to the motor using PID control to achieve a particular speed relative to the maximum speed of the motor.
- RunMode.RUN_TO_POSITION: Motor will run until a target position is reached, which is specified as a tick count.
- RunMode.STOP_AND_RESET_ENCODER: Motor is stopped and the encoder value is reset to 0.

set TargetPosition
<motor name> to <numeric value>
<motor name> to <numeric value>

Description: Set the target positions of the two named motors to the specified values.

set ZeroPowerBehavior
<motor name> to <zero power behavior>
<motor name> to <zero power behavior>
Description: Set the zero power behavior of the named motors. Valid values of <zero power behavior> are

ZeroPowerBehavior.BRAKE: Stop the motor's axle.

ZeroPowerBehavior.FLOAT: Let the motor's axle rotate but don't generate torque

Servo

When an op mode initializes, servos will have the following attributes until a block executes that changes the attribute.

- Direction: FORWARD
- Scale range
  - Minimum: 0 degrees
  - Maximum: 180 degrees

set <servo name>.Direction to <direction value>
Description: Set direction of the named servo to the indicated direction. Valid values of <direction value> are direction.FORWARD and direction.FORWARD.

<direction value> ← <servo name>.Direction
Description: Return current direction of named servo.

set <servo name>.Position to <numeric value>
Description: Set position of named servo to specified value.

<numeric value> ← <servo name>.Position
Description: Return the current position value of the named value.

Call <servo name>.scaleRange
min <numeric value>
max <numeric value>
Description: Set scale range of specified servo using the given min and max value.

ServoController

<PWM Status> ← <Servo Controller #>.pwmStatus
Description: Return the output status of the servo controller indicated by the #.

Call < Servo Controller #>.pwmDisable
Description: Disable servo outputs by the servo controller indicated by the #

Call < Servo Controller #>.pwmEnable
Description: Enable servo outputs by the servo controller indicated by the #

Sensors

Blocks Programming supports a variety of FTC-compatible sensors. The blocks associated with a particular sensor appear in the Sensor menu only if that sensor is described in the configuration file

that is currently activated on the Robot Controller phone.

ColorSensor (Modern Robotics)

The sensor is accessed via what is known as an I2C address. These are 8-bit values in the range 16 to 254 decimal or 10 to FE hexadecimal and must be even. A Modern Robotics color sensor has a default address of 3C hexadecimal and that address is assumed by an op mode written with Blocks Programming unless you specify otherwise. You can assign a Modern Robotics I2C sensor a different hexadecimal address using Modern Robotics Core Device Discovery utility found at http://www.modernroboticsinc.com/coredevicediscovery.

If you change the address of a color sensor you should inform the Blocks Programming environment of the new address using "Set <color sensor name>.i2cAddress7Bit" or "Set <color sensor name>.i2cAddres8Bit to <numeric value>". If you use blocks that refer to "7Bit" the high 7 of the 8 bits are used and the numbers range from 8 to 7F hexadecimal or 8 to 127 decimal. For example, the 8-bit address of 22 hexadecimal or 34 decimal would be half these values as a 7-bit address: 11 hexadecimal or 17 decimal.

The following blocks are used to communicate with a color sensor. There are additional blocks that allow you to manipulate color information that are found in the Color folder under Utilities.

<numeric value> ← <color sensor name>.Alpha
Description: Return the current alpha value from named sensor.

<numeric value> ← <color sensor name>.Argb
Description: Return the current alpha-RGB value from named sensor.

<numeric value> ← <color sensor name>.Blue
Description: Return the current blue component from the name sensor.

<numeric value> ← <color sensor name>.Green
Description: Return the current green component from the named sensor.

Set <color sensor name>.i2cAddress7Bit to <numeric value>
Description: Set the 7 bit I2C address of the named color sensor to the specified decimal value. Not all sensors support this feature. Example: If you use Robotics Core Device Discovery to set the sensor's I2C address to 22 hexadecimal, you would convert that value to 34 decimal and then divide by 2 and provide 17 as the numeric value to the Set i2cAddress7bit block.

<numeric value> ← <color sensor name>.i2cAddress7Bit
Description: Return the 7-bit I2C address of the named color sensor.

Set <color sensor name>.i2cAddres8Bit to <numeric value>
Description: Set the 8 bit I2C address of the named color sensor to the specified decimal value. This number must be even. Not all sensors support this feature. Example: If you use Robotics Core Device Discovery to set the sensor's I2C address to 22 hexadecimal, you would convert that value to 34 decimal and provide 34 as the numeric value to the Set i2cAddress8bit block.

<numeric value> ← <color sensor name>.i2cAddres8Bit
Description: Return the 7-bit I2C address of the named color sensor.

<numeric value> ← <color sensor name>.Red
Description: Return the current red component from the named sensor.

call <color sensor name>.enableLed

enable <logic value>
Description: Enable the LED of the light sensor if the logic value. Disable otherwise. Not all sensors support this feature.

GyroSensor (Modern Robotics)

The sensor is accessed via what is known as an I2C address. These are 8-bit values in the range 16 to 254 decimal or 10 to FE hexadecimal and must be even. A Modern Robotics gyro sensor has a default address of 20 hexadecimal and that address is assumed by an op mode written with Blocks Programming unless you specify otherwise. You can assign a Modern Robotics I2C sensor a different hexadecimal address using Modern Robotics Core Device Discovery utility found at http://www.modernroboticsinc.com/coredevicediscovery.

If you change the address of a gyro sensor you should inform the Blocks Programming environment of the new address using "Set <gyro sensor name>.i2cAddress7Bit" or "Set <gyro sensor name>.i2cAddres8Bit to <numeric value>". If you use blocks that refer to "7Bit" the high 7 of the 8 bits are used and the numbers range from 8 to 7F hexadecimal or 8 to 127 decimal. For example, the 8-bit address of 22 hexadecimal or 34 decimal would be half these values as a 7-bit address: 11 hexadecimal or 17 decimal.

<numeric value> ← <gyro name>.Heading
Description: Return the current heading of the named gyro.

Set <gyro name>.HeadingMode to <heading mode value>
Description: Set heading mode of named gyro. Valid values are HeadingMode.HEADING_CARTESIAN and HeadingMode.HEADING_CARINAL

<heading mode> ← <gyro name>.HeadingMode
Description: Return current heading mode for named gyro.

set <gyro name>.I2cAddress7Bit to <numeric value>
Description: Set the 7-bit I2C address of the named gyro to the specified decimal value. Not all gyro sensors support this feature.

<numeric value> ← <gyro name>. I2cAddress7Bit
Description: Return the 7-bit I2C address of the named gyro. Not all gyro sensors support this feature.

set <gyro name>.I2cAddress8Bit to <numeric value>
Description: Set the 8-bit I2C address of the named gyro to the specified decimal value. Not all gyro sensors support this feature.

<numeric value> ← <gyro name>. I2cAddress8Bit
Description: Return the 8-bit I2C address of the named gyro. Not all gyro sensors support this feature.

<numeric value> ← <gyro name>.RawX
Description: Return the current raw X value of the named gyro. Not all gyro sensors support this feature.

<numeric value> ← <gyro name>.RawY
Description: Return the current raw Y value of the named gyro. Not all gyro sensors support this feature.

<numeric value> ← <gyro name>.RawZ

Description: Return the current raw Z value of the named gyro. Not all gyro sensors support this feature.

<numeric value> ← <gyro name>.RotationFraction
Description: Return the rotation fraction of the named gyro. Valid values are from 0.0 and 1.0. Not all gyro sensors support this feature.

call <gyro name>.calibrate
Description: Cause the named gyro to calibrate itself. Not all gyro sensors support this feature. For the modern robotics gyro sensor this will cause the z-axis heading to be reset.

call <gyro name>.isCalibrating
Description: Return true if the named gyro is currently calibrating, false otherwise. Not all gyro sensors support this feature.

call <gyro name>.resetZAxisIntegrator
Description: Reset the integrated Z axis to zero. Not all gyro sensors support this feature.

IMU-BNO055 (REV Expansion Hub)

Introduction: The REV Expansion Hub includes an Inertial Measurement Unit (IMU) based on a Bosch BNO055 intelligent 9-axis absolute orientation sensor. This sensor measures acceleration and direction using compass, accelerometer and gyroscope features. The sensor usually needs to calibrated after it is initialized. Some modes are self-calibrating, only requiring that the robot not be moved for one or two seconds. Other modes require that the Hub be moved for calibration to occur. This can be done by rotating the Expansion Hub (or more likely the robot in which it is installed) 90 degrees around the x, y and z axis. You can thing of this as pitching forward 90 degrees, rolling right or left 90 degrees, and yawing (turning) left or right 90 degrees.

The IMU can be programmed to use integration of the acceleration it detects to calculate velocity and distance travelled using an integration process. It reports distance traveled as a change in position since integration was started. Note: Integration is a reference to summing values over time.

The blocks described below return a variety of object types. Blocks programming includes blocks for each type as submenus of the Utilities menu. For instance, to extract the x, y and z components of a position object, use blocks found in the Utilities-Position menu.

<acceleration object> ← imu.Acceleration

Description: Returns most recent acceleration information.

<orientation object> ← imu.AngularOrientation

Description: Returns absolute orientation of REV Expansion Hub as an orientation object, which is a set of three angles .

<text object> ← imu.AngularOrientationAxes

Description: Returns list of axes on which the sensor measures angular orientation.

<angular velocity object> ← imu.AngularVelocity

Description: Returns rate of change of angular orientation along up to three axes.

<text object> ← imu.AngularVelocityAxes

Description: Returns list of axes on which the angular velocity is available. Some sensors return velocity around X, Y, and Z axes while others only measure velocity around one or two axes.

<logical value> ← imu.CalibrationStatus

Description: Returns a text string giving the calibration status of the sensor. The string is in the format, “IMU Calibration Status : sx gx ax mx” where s stands for system, g for gyro, a for accelerometer and m for magnetometer. The x values or 0, 1, 2 or 3 where 0 means uncalibrated, 3 means fully calibrated and 1 and 2 mean partially calibrated. For example, “IMU Calibration Status : s0 g3 a0 m1” the system is not calibrated, the gyro is fully calibrated, the accelerometer is not calibrated and the magnetometer is partially calibrated.

<acceleration object> ← imu.Gravity

Description: Returns orientation of the force of gravity relative to the REV Expansion Hub.

set imu.I2cAddress7Bit to <numeric value>

Description: Set the 7-bit I2C address of the IMU.

<numeric value> ← imu.I2cAddress7Bit

Description: Returns 7-bit address of the IMU.

Set imu.I2cAddress8Bit to <numeric value>

Description: Set the 8-bit I2C address of the IMU.

<numeric value> ← imu.I2cAddress8Bit

Description: Returns 8-bit address of the IMU.

<acceleration object> ← imu.LinearAcceleration

Description: Returns linear acceleration.

<magnetic flux object> ← imu.MagneticFieldStrength

Description: Returns magnetic field strength.

<acceleration object> ← imu.OverallAcceleration

Description: Returns overall acceleration, which is the sum of acceleration do to change in velocity of the REV Expansion Hub and the acceleration due to gravity.

<position object> ← imu.Position

Description: Returns current position based on direction and distance traveled since integration was started by one of the imu.startAccelerationIntegration blocks. The IMU integrates the acceleration it detects once to determine velocity (direction and speed) and then a second time to determine distance traveled.

<quaternion object> ← imu.QuarternionOrientation

Description: Returns absolute orientation of the REV Expansion Hub as quarternion object. The x, y and z components of the quarternion object are the unitless orientation angles around the x, y and z axes. For example, z values from -1 to 1 represent turn angles of 180 degrees clockwise to 180 counterclockwise.

<text object> ← imu.SystemError

Description: If SystemStatus is “SYSTEM_ERROR”, returns more detail about the error.

<text object> ← imu.SystemStatus

Description: Returns the IMU system status. Possible values include “SYSTEM_ERROR”.

<temperature object> ← imu.Temperature

Description: Returns the current temperature of the REV Expansion Hub.

<velocity object> ← imu.Velocity

Description: Returns the current velocity based on integrating the acceleration values over time. For this feature to work, integration must have been previously started using one of the imu.startAccelerationIntegration blocks.

<orientation object> ← call imu.getAngularOrientation
axesReference [AxesReference.EXTRINSIC or AxesReference.INTRINSIC]
axesOrder AxesOrder.[XYX or XYZ or ZZX or XZY or YXY or YXZ or YZX or YZY or YZY or ZXY or ZXZ or ZYX or ZYZ]
angleUnit AngleUnit [DEGREES or RADIANS]

Description: Returns absolute orientation of the sensor as an orientation object containing three angles. The angles will be in the order specified by AxesOrder. If one of the angles is not supported, it is reported as 0.

<angular velocity object> ← call imu.getAngularVelocity
angleUnit AngleUnit [DEGREES or RADIANS]

Description: Returns the angular rotation rate around up to three axes (x, y and z). If a particular axes is not supported it is reported as 0.

call imu.initialize
parameters <parametersVariable>

Description: Initializes the IMU based on an IMU parameters object (see next section).

<logic value> ← call imu.isAccelerometerCalibrated

Description: Returns true if the Accelerometer has completed calibration; false otherwise. See Introduction section above for a discussion of calibration.

<logic value> ← call imu.isGyroCalibrated

Description: Returns true if the gyroscope has completed calibration; false otherwise. See Introduction section above for a discussion of calibration.

<logic value> ← call imu.isMagnetometerCalibrated

Description: Returns true if the Magnetopmeter has completed calibration; false otherwise. See Introduction section above for a discussion of calibration.

<logic value> ← call imu.isSystemCalibrated

Description: Returns true of all features associated with current mode have completed calibration. See Parameters section below for discussion of mode.

call imu.saveCalibrationData filename <text object>

Description: Save the current calibration information in the file specified by the given text object. Such files end with the "json" extension.

call imu.startAccelerationIntegration
msPollInterval <numeric value>

Description: Start integration of acceleration sensor values to calculate velocity and distance. Use the specified numeric value as the number of milliseconds between each integration step.

call imu.startAccelerationIntegration
initialPosition <position object>
initialVelocity <velocity object>
msPollInterval <numeric value>

Description: Start integration of acceleration sensor values to calculate velocity and distance. Use the given position object as the initial x, y and z components of the position. Use the given velocity object to specify the initial velocity. Use the specified numeric value as the number of milliseconds between each integration step. The first integration will calculate velocity by using change in velocity (i.e. acceleration) to update the previous velocity starting with the initial velocity. The second integration will update the previous position using the distance and direction of travel.

call imu.stopAcceleartionIntegration

Description: Stop the integration process. That is, stop updating the velocity and position of the REV Expansion hub.

IMU-BNO055.Parameters (REV Expansion Hub)

<IMU parameters object> ← new IMU-BNO055.Parameters

Description: Create a new IMU parameters object.

call IMU-BNO055.Parameters.setAccelUnit
parameters <parameters object>
accelUnit AccelUnit.[METERS_PERSEC_PERSEC
or MILLI_EARTH_GRAVITY]

Description: Update IMU parameters object by setting the acceleration units to either meters

per second per second or thousandth of the acceleration due to gravity (about 9.8 meters per second per second)

call IMU-BNO055.Parameters.setAccelerationIntegrationAlgorithm
parameters <parameters object>
accelerationIntegrationAlgorithm
accelerationIntegrationAlgorithm [NAÏVE or JUST LOGGING]

Description: Set the acceleration algorithm in the IMU parameters object.

call IMU-BNO055.Parameters.setI2cAdress7Bit
parameters <parameters object>
i2cAddress7Bit <numeric value>

Description: Set the 7-bit address of the IMU on the I2C bus.

call IMU-BNO055.Parameters.setI2cAdress8Bit
parameters <parameters object>
i2cAddress8Bit <numeric value>

Description: Set the 8-bit address of the IMU on the I2C bus.

call IMU-BNO055.Parameters.setLoggingEnabled
parameters <parameters object>
loggingEnabled [true OR false]

Description: Set logging-enabled state in the IMU parameters object based on whether loggingEnabled is true or false. If the state is set to true the IMU sensor activities will be added to the log. [Note: This feature does not appear to be working.]

call IMU-BNO055.Parameters.setLoggingTag
parameters <parameters object>
loggingTag <text object>

Description: Set the logging tag for the IMU to the specific text string. This tag will appear at the beginning of each log IMU log entry.

call IMU-BNO055.Parameters.setSensorMode
parameters <parameters object>
sensorMode SensorMode.[ACCONLY or MAGONLY
or GYROONLY or ACCMAG
or ACCGYRO or MAGGYRO
or AMG or IMU or COMPASS
or M4G or NDOF_FMC_OFF or NDOF]

Description: Set the sensor mode in the given IMU parameters object. Here is what the Bosch data sheet says about these modes:

ACCONLY: If the application requires only raw accelerometer data, this mode can be chosen. In this mode the other sensors (magnetometer, gyro) are suspended to lower the power consumption. In this mode, the BNO055 behaves like a stand-alone acceleration sensor.

MAGONLY: In MAGONLY mode, the BNO055 behaves like a stand-alone magnetometer, with acceleration sensor and gyroscope being suspended.

GYROONLY: In GYROONLY mode, the BNO055 behaves like a stand-alone gyroscope, with acceleration sensor and magnetometer being suspended.

ACCMAG: Both accelerometer and magnetometer are switched on, the user can read the data from these two sensors.

ACCGYRO: Both accelerometer and gyroscope are switched on; the user can read the data from these two sensors.

MAGGYRO: Both magnetometer and gyroscope are switched on, the user can read the data from these two sensors.

AMG (ACC-MAG-GYRO): All three sensors accelerometer, magnetometer and gyroscope are switched on.

IMU (Inertial Measurement Unit): In the IMU mode the relative orientation of the BNO055 in space is calculated from the accelerometer and gyroscope data. The calculation is fast (i.e. high output data rate).

COMPASS: The COMPASS mode is intended to measure the magnetic earth field and calculate the geographic direction. The earth magnetic field is a vector with the horizontal components x, y and the vertical z component. It depends on the position on the globe and natural iron occurrence. For heading calculation (direction of compass pointer) only the horizontal components x and y are used. Therefore, the vector components of the earth magnetic field must be transformed in the horizontal plane, which requires the knowledge of the direction of the gravity vector. To summarize, the heading can only be calculated when considering gravity and magnetic field at the same time. However, the measurement accuracy depends on the stability of the surrounding magnetic field. Furthermore, since the earth magnetic field is usually much smaller than the magnetic fields that occur around and inside electronic devices, the compass mode requires calibration

M4G (Magnet for Gyroscope): The M4G mode is similar to the IMU mode, but instead of using the gyroscope signal to detect rotation, the changing orientation of the magnetometer in the magnetic field is used. Since the magnetometer has much lower power consumption than the gyroscope, this mode is less power consuming in comparison to the IMU mode. There are no drift effects in this mode which are inherent to the gyroscope. However, as for compass mode, the measurement accuracy depends on the stability of the surrounding magnetic field. For this mode no magnetometer calibration is required and also not available.

NDOF_FMC_OFF: This fusion mode is same as NDOF mode, but with the Fast Magnetometer Calibration turned ‘OFF’.

NDOF: This is a fusion mode with 9 degrees of freedom where the fused absolute orientation data is calculated from accelerometer, gyroscope and the magnetometer. The advantages of combining all three sensors are a fast calculation, resulting in high output data rate, and high robustness from magnetic field distortions. In this mode the Fast Magnetometer calibration is turned ON and thereby resulting in quick calibration of the

magnetometer and higher output data accuracy. The current consumption is slightly higher in comparison to the NDOF_FMC_OFF fusion mode.

call IMU-BNO055.Parameters.setTempUnit
parameters <parameters object>
tempUnit TempUnit.[CELSIUS or FARENHEIT]

Description: Set the units to report the IMU's temperature.

<acceleration unit> ← IMU-BNO055.Parameters.AccelUnit
parameters <parameters object>

Description: Returns the acceleration units in the given IMU parameters object.

<acceleration integration algorithm> ←
IMU-BNO055.Parameters.AccelerationIntegrationAlgorithm
parameters <parameters object>

Description: Returns the acceleration integration algorithm in the given IMU parameters object.

<angle unit> ← IMU-BNO055.Parameters.AngleUnit
parameters <parameters object>

Description: Returns the angle unit in the given IMU parameters object.

<text object> ← IMU-BNO055.Parameters.CalibrationFile
parameters <parameters object>

Description: Returns the name of calibration data file associated with the IMU parameters object.

<numeric value> ← IMU-BNO055.Parameters.i2cAddress7Bit
parameters <parameters object>

Description: Returns the I2C 7-bit address associated with the IMU parameters object.

<numeric value> ← IMU-BNO055.Parameters.i2cAddress8Bit
parameters <parameters object>

Description: Returns the I2C -bit address associated with the IMU parameters object.

<logic value> ← IMU-BNO055.Parameters.LoggingEnabled
parameters <parameters object>

Description: Returns if logging is enabled in the IMU parameters object; false otherwise.

<text object> ← IMU-BNO055.Parameters.LoggingTag
parameters <parameters object>

Description: Returns logging tag in the IMU parameters object.

<sensor mode> ← IMU-BNO055.Parameters.SensorMode
parameters <parameters object>

Description: Returns sensor mode in the IMU parameters object.
parameters <parameters object>

<temperature unit> ← IMU-BNO055.Parameters.TempUnit
parameters <parameters object>

Description: Returns temperature unit in the IMU parameters object.

Mrl2cCompassSensor

The sensor is accessed via what is known as an I2C address. These are 8-bit values in the range 16 to 254 decimal or 10 to FE hexadecimal and must be even. A Modern Robotics compass sensor has a default address of 24 hexadecimal and that address is assumed by an op mode written with Blocks Programming unless you specify otherwise. You can assign a Modern Robotics I2C sensor a different hexadecimal address using Modern Robotics Core Device Discovery utility found at http://www.modernroboticsinc.com/coredevicediscovery.

If you change the address of a compass sensor you should inform the Blocks Programming environment of the new address using "Set <compass name>.i2cAddress7Bit" or "Set <compass name>.i2cAddres8Bit to <numeric value>". If you use blocks that refer to "7Bit" the high 7 of the 8 bits are used and the numbers range from 8 to 7F hexadecimal or 8 to 127 decimal. For example, the 8-bit address of 22 hexadecimal or 34 decimal would be half these values as a 7-bit address: 11 hexadecimal or 17 decimal.

<numeric value> ← <compass name>.Direction
Description: Return the current direction in degrees of the named compass.

set <compass name>.I2cAddress7Bit to <numeric value>
Description: Set the 7-bit I2C address of the named compass to the specified decimal value. Not all compass sensors support this feature.

<numeric value> ← <compass name>.I2cAddress7Bit
Description: Return the 7-bit I2C address of the named compass. Not all compass sensors support this feature.

set <compass name>.I2cAddress8Bit to <numeric value>
Description: Set the 8-bit I2C address of the named compass to the specified decimal value. Not all compass sensors support this feature.

<numeric value> ← <compass name>.I2cAddress8Bit
Description: Return the 8-bit I2C address of the named compass.

<numeric value> ← <compass name>.XAccel
Description: Return the current x acceleration in Gs of the named compass.

<numeric value> ← <compass name>.XMagneticFlux
Description: Return the current magnetic flux in Teslas in the x direction of the named compass.

<numeric value> ← <compass name>.YAcce
Description: Return the current y acceleration in Gs of the named compass.

<numeric value> ← <compass name>.YMagneticFlux
Description: Return the current magnetic flux in Teslas in the y direction of the named compass.

<numeric value> ← <compass name>.ZAccel

Description: Return the current z acceleration in Gs of the named compass.

<numeric value> ← <compass name>.ZMagneticFlux
Description: Return the current magnetic flux in Teslas in the z direction of the named compass.

<logic value> ← call <compass name>.calibrationFailed
Description: Return true if the last calibration of the named compass failed, true otherwise.

<logic value> ← call <compass name>.isCalibrating
Description: Return true if the named compass is currently calibrating, false otherwise.

Call <compass name>.setMode compassMode <compass mode value

Description: Set the mode of the named compass. Valid values of <compass mode value> are CompassMode.MEASUREMENT_MODE and CALIBRATION_MODE.

MrI2cRangeSensor

The sensor is accessed via what is known as an I2C address. These are 8-bit values in the range 16 to 254 decimal or 10 to FE hexadecimal and must be even. A Modern Robotics range sensor has a default address of 28 hexadecimal and that address is assumed by an op mode written with Blocks Programming unless you specify otherwise. You can assign a Modern Robotics I2C sensor a different hexadecimal address using Modern Robotics Core Device Discovery utility found at http://www.modernroboticsinc.com/coredevicediscovery.

If you change the address of a color sensor you should inform the Blocks Programming environment of the new address using "Set <range sensor name>.i2cAddress7Bit" or "Set <range sensor name>.i2cAddres8Bit to <numeric value>". If you use blocks that refer to "7Bit" the high 7 of the 8 bits are used and the numbers range from 8 to 7F hexadecimal or 8 to 127 decimal. For example, the 8-bit address of 22 hexadecimal or 34 decimal would be half these values as a 7-bit address: 11 hexadecimal or 17 decimal.

<numeric value> ← <range sensor name>.CmOptical
Description: Return the distance in centimeters from the optical sensor.

<numeric value> ← <range sensor name>.CmUltrasonic
Description: Return the distance in centimeters from the ultrasonic sensor.

Set <range sensor name>.I2cAddress7Bit to <numeric value>
Description: Set the 7-bit I2C address of the named range sensor.

<numeric value> ← <range sensor name>.I2cAddress7Bit
Description: Return the 7-bit I2C address of the named range sensor.

Set <range sensor name>.I2cAddress8Bit to <numeric value>
Description: Set the 8-bit I2C address of the named range sensor.

<numeric value> ← <range sensor name>.I2cAddress8Bit
Description: Return the 8-bit I2C address of the named range sensor.

<logic value> ← <range sensor name>.LightDetected
Description: Return the amount of light detected from the named range sensor. The value will be from 0.0 to 1.0.

<logic value> ← <range sensor name>.RawLightDetected
Description: Return a value of the named range sensor, which is proportional to the amount of light detected.

<numeric value> ← <range sensor name>.RawLightDetectedMax
Description: Return the maximum raw light value that can be returned by the named range sensor.

<numeric value> ← <range sensor name>.RawOptical
Description: Return the raw distance value from the named optical range sensor

<numeric value> ← <range sensor name>.RawUltrasonic
Description: Return the raw distance value from the ultrasonic range sensor.

<numeric value> ← call <range sensor name>.getDistance unit <distance unit>
Description: Return the current distance from the named range sensor using the requested distance unit. Valid values of <distance unit> are DistanceUnit.METER, DistanceUnit.CM, DistanceUnit.MM, DistanceUnit.INCH.

OpticalDistanceSensor

This sensor is connected as an analog device and is not connected via an I2C bus so it does not have an I2C address.

<numeric value> ← <optical distance sensor name>.LightDetected
Description: Return the amount of light detected by the named optical distance sensor using values in the range 0.0 to 1.0.

<numeric value> ← <optical distance sensor name>.RawLightDetected
Description: Return a signal value from the named optical distance sensor that is proportional the amount of light detected.

<numeric value> ← <optical distance sensor name>.RawLightDetectedMax
Description: Return the maximum raw-light-detected value that can be returned by the named optical distance sensor.

Call <optical distance sensor name>.enableLed enable <logic value>
Description: If the provided <logic value> is true, enable the LED of the named optical distance sensor. Otherwise disable the LED. Not all optical distance sensors support this feature.

REV Color/Range Sensor

The REV Color/Range Sensor is also known as a Color-Distance or a Color-Proximity Sensor. It contains a white LED and a color sensor. The LED serves as a flashlight and the color sensor measures the amount and color of light reflected back from the nearest surface. From this information it also estimates the distance to the surface.

<numeric value> ← <sensor name>.LightDetected
Description: Return the amount of light detected by the named optical distance sensor using values in the range 0.0 to 1.0. In most cases the light detected is light generated by the sensors LED and reflected off a nearby surface. The range of values returned is highly sensitive to the distance between the sensor and the surface from which light is being reflected

<numeric value> ← <sensor name>.RawLightDetected
Description: Return a signal value from the named optical distance sensor that is proportional the amount of light detected.

<numeric value> ← <sensor name>.RawLightDetectedMax
Description: Return the maximum raw-light-detected value that can be returned by the named

optical distance sensor.

<numeric value> ← <sensor name>.Alpha
Description: Return the current alpha value from named sensor.

<numeric value> ← <sensor name>.Argb
Description: Return the current alpha-RGB value from named sensor.

<numeric value> ← < sensor name>.Blue
Description: Return the current blue component from the name sensor.

<numeric value> ← <sensor name>.Green
Description: Return the current green component from the named sensor.

Set <sensor name>.i2cAddress7Bit to <numeric value>
Description: Set the 7 bit I2C address of the named color sensor to the specified decimal value. Not all sensors support this feature.

<numeric value> ← < sensor name>.i2cAddress7Bit
Description: Return the 7-bit I2C address of the named color sensor.

Set < sensor name>.i2cAddres8Bit to <numeric value>
Description: Set the 8 bit I2C address of the named color sensor to the specified decimal value. This number must be even. Not all sensors support this feature.

<numeric value> ← <sensor name>.i2cAddres8Bit
Description: Return the 7-bit I2C address of the named color sensor.

<numeric value> ← <sensor name>.Red
Description: Return the current red component from the named sensor.

<numeric value> ← <sensor name>.getDistance unit DistanceUnit [CM or INCH or METER or MM]

Description: Return the distance measured by the sensor in the indicated units.

<color value> ← <sensor name>.getNormalizedColors

Description: Return the color detected by the sensor.

TouchSensor

This sensor is connected as a digital device and not connected via an I2C bus so it does not have an I2C address.

<logic value> ← <touch sensor name>.IsPressed
Description: Return true if the named touch sensor has been pressed, false otherwise.

VoltageSensor

<numeric value> ← <Motor Controller #>.Voltage

Description: Return voltage available to motor controller indicated by the #

Other Devices

AnalogInput

<numeric value> ← <analog input name>.MaxVoltage
Description: Return the maximum voltage value that can be returned by the named analog input device.

<numeric value> ← <analog input>.Voltage
Description: Return the current voltage value of the named analog input device.

AnalogOutput

call <analog output name>.setAnalogOutputFrequency frequency <numeric value>
Description: Set the frequency of the named analog output device or channel to the specified value.

call <analog output name>.setAnalogOutputMode mode <numeric value>
Description: Set the output mode of the named analog output or channel device to the specified value.

call <analog output name>.setAnalogOutputVoltage voltage <numeric value>
Description: Set the named analog output device or channel to the specified voltage.

LED

call <LED name>enableLed enable <logic value>

Description: Enable the named LED light if logic value is true. Disable it otherwise.>

Android

Android devices often have sensors built into them. This section includes descriptions of Blocks Programming features that support the Android accelerometer and gyroscope (referred to as Orientation). In addition to these sensors, Android devices allow audio files to be played, supported in Blocks Programming as SoundPools. Blocks Programming also supports Text-to-Speech, which is available on some Android devices.

Accelerometer

The blocks in the Accelerometer folder allow you communicate with the accelerator feature on Android phones if the feature is available. Additional blocks that can be used to manipulate acceleration data can be found in the Acceleration folder in the Utilities folder.

<logic value> ← call AndroidAccelerometer.isAvailable Description: returns true if the accelerator feature is available on the Robot Controller phone, false otherwise

call AndroidAccelerometer.startListening

Description: Enable the Accelerator so your op mode can receive values from it.

call AndroidAccelerometer.stopListening

Description: Disable the accelerometer as your op mode no longer needs to receive values from it.

<acceleration object> ← AndroidAccelerometer.Acceleration

Description: Return new acceleration object based on current state of the acceleration sensor.

<numeric value> ← AndroidAccelerometer.X

Description: Returns the x attribute of the acceleration sensor

<numeric value> ← AndroidAccelerometer.Y
Description: Returns the y attribute of the acceleration sensor

<numeric value> ← AndroidAccelerometer.Z
Description: Returns the z attribute of the acceleration sensor

<numeric value> ← AndroidAccelerometer.DistanceUnit
[CM or INCH or METER or MM]

Description: Returns distance unit currently in effect for the accelerometer

set AndroidAccelerometer.DistanceUnit to <unit type>
Description: Sets the distance unit for the accelerometer.
Valid values of <unit type> are CM, INCH, METER or MM

Orientation

The blocks in this folder can be used to communicate with the orientation or gyroscopic features of an Android device if such features are available. Additional blocks for manipulating orientation data

can be found in the Orientation folder in the Utilities folder.

<logic value> ← call AndroidOrientation.isAvailable
Description: Returns true if the Android device makes gyroscopic orientation available. False otherwise.

call AndroidOrientation.startListening
Description: Enables the availability of orientation values

call AndroidOrientation.stopListening
Description: Disables the availability of orientation values

<numeric value> ← AndroidOrientation.Azimuth
Description: Returns the current azimuth angle

Note: You should set the angle units using the "set…AngleUnit" function below.

<numeric value> ← AndroidOrientation.Pitch
Description: Returns the current pitch angle

<numeric value> ← AndroidOrientation.Roll
Description: Returns the current roll angle

<numeric value> ← AndroidOrientation.Angle
Description: Returns synthetic tilt angle calculated as atan2(pitch,roll

<numeric value> ← AndroidOrientation.Magnitude
Description: Returns magnitude of the tilt. Values are between 0 and 1, where 0 represents horizontal (face up or down) and 1 represents vertical in portrait or landscape orientation

<angle unit> <== AndroidOrientation.AngleUnit
Description: Returns the units currently in effect for angles returned by the gyroscopic orientation feature. Valid values of <angle unit> are DEGREES and RADIANS

set AndroidOrientation.AngleUnit to AngleUnit [DEGREES or RADIANS]
Description: sets the angle unit to be used by the gyroscopic orientation feature to either DEGREES or RADIANS

SoundPool

Introduction: This feature allows you to have the Robot Controller phone play sounds files that are in WAV or MP3 format. It appears to limit the playback to about six seconds.

call AndroidSoundPool.initialize

Description: Initializes the Sound Pool feature

<logic value> ← call AndroidSoundPool.preloadSound <file name>

Description: Loads a WAV or MP3 sound file that can be played later. The block returns true if successful, false otherwise. It will look for the specified file in the

"/FIRST/blocks/sounds" directory on your Robot Controller phone. You may need to create the "sounds" directory. The AndroidSoundPool.initialize block must be called before this preloadSound block is called.

call AndroidSoundPool.play soundName <file name>

Description: Plays the sound file file loaded earlier.

call AndroidSoundPool.stop

Description: Disables the Sound Pool feature.

set AndroidSoundPool.Volume to <numeric value>

Description: Adjust the sound volume. Valid values are from 0 to 1 with 0.5 be average sound

<numeric value> ← AndroidSoundPool.Volume

Description: Returns the current sound volume.

set AndroidSoundPool.Rate to <numeric value>

Description: Sets the sound rate. Valid values are from 0.5 to 2.0 with 1.0 being the normal rate.

<numeric value> ← AndroidSoundPool.Rate

Description: Returns the sound rate.

set AndroidSoundPool.Loop to <numeric value>

Description: Set the number of times a sound should be repeat. -1 means repeat indefinitely.

<numeric value> ← AndroidSoundPool.Loop

Description: Returns the current loop value.

TextToSpeech

call AndroidTextToSpeech.initialize

Description: Initializes the text-to-speech feature. At least two seconds must pass after this initialize function is called before the other TextToSpeech features are used.

<logic value> ← AndroidTextToSpeech.Status

Description: Returns true if text to speech is available, false otherwise.

<language code> ← AndroidTextToSpeech.LanguageCode

Description: Returns a code indicating the current language.

Note: <language code> is ISO 639 alpha-2 or alpha-3 or alpha subtag up to characters long. See Wikipedia. "en" for English.

<country code> ← AndroidTextToSpeech.CountryCode

Description: Returns a code indicating country.

Note: <country code> is ISO 3136 alpha-2 alpha code or UN M.49 numeric-3 area code. See Wikipedia. "US" for United States.

set AndroidTextToSpeech.Pitch to <numeric value>
Description: Set the pitch of the speech. 1.0 is normal, less than 1 is lower; more than 1 is higher.

set AndroidTextToSpeech.SpeechRate to <numeric value>
Description: Sets the rate of the speech. 1.0 is normal. 0.5 is half speed. 2.0 is double speed.

<logic value> ← call AndroidTextSpeech.isLanguageAvailable
languageCode <language code>
Description: Returns true if indicated language is available, false otherwise.

<logic value> ← call AndroidTextToSpeech.isLanguageAvailable
languageCode <language code>
Description: Returns true if indicated langague is available, false otherwise.

call AndroidToSpeech.setLanguage language <language code>

Description: Sets the language.

call call AndroidToSpeech.setLanguage language <language code>
countryCode <country code>
Description: Sets the language and country.

call AndroidTextToSpeech.speak <text value>
Description: Generates speech from the text value.

Utilities

Acceleration

<text value> ← Acceleration.DistanceUnit acceleration <acceleration object>

Description: Return the distance unit attribute of the acceleration object. Valid values are CM, INCH, METER and MM

<numeric value> ← Acceleration.[XAccel or YAccel or ZAccel]

acceleration <acceleration Variable>

Description: Return the x, y or z attribute of the acceleration object.

<time value> ← Acceleration.AcquistionTime acceleration <acceleration object>

Description: Return the acquisition time attribute of the acceleration object.

<acceleration object> ← call Acceleration.toDistanceUnit

acceleration <acceleration Variable>

distanceUnit DistanceUnit.[CM or INCH or METER or MM]

Description: Return an acceleration object based on another acceleration object after converting it x, y and z values to the distant unit indicated.

<text value> ← call Acceleration.toText acceleration <acceleration object>

Description: Return text string that is a formatted version of the acceleration object.

<acceleration object> ← new Acceleration

Description: Return a new acceleration object.

<acceleration object> ← new Acceleration
distanceUnit DistanceUnit [CM or INCH or METER or MM]
xAccel <numeric value> yAccel <numeric value> zAccel <numeric value>
acquisitionTime call System.nanoTime

Description: Returns a new acceleration object using the specified distance unit, x, y and z values in those units and an acquisition time as the current time.

<acceleration object> ← call Acceleration.fromGravity
gx <numeric value> gy <numeric value> gz <numeric value>

acquisitionTime call System.nanoTime

Description: Returns a new acceleration object with x, y and z values given as factors of the gravitation acceleration constant, g, and the system current time as the acquisition time.

AngularVelocity

<text value> ← AngularVelocity.AngleUnit
angularVelocity <angular velocity object>

Description: Returns the angle unit (DEGREES or RADIANS) of the indicated angular velocity object.

<angular velocity object> ← AngularVelocity.
[XRotation OR YRotationRate OR ZRotationRate]
angularVelocity <angular velocity object>

Description: Returns the x, y or z rotation rate from the angular velocity object.

<time value> ← AngularVelocity.AcquisitionTime
angularVelocity <angular velocity object>

Description: Returns the acquisition time of the angular velocity object.

<numeric value> ← call AngularVelocity.toAngleUnit
angularVelocity <angular velocity object>

angleUnit AngleUNIT [DEGREES or RADIANS]

Description: Returns an angular velocity object based on another object after converting its x, y and z rotation rates to the specified units.

<angular velocity object> ← new AngularVelocity

Description: Return a new angular velocity object.

<angular velocity object> ← new AngularVelocity
angleUnit AngleUnit [DEGREES OR RADIANS]
XRotationRate <numeric value>
YRotationRate <numeric value>
ZRotationRate <numeric value>
acquisitionTime call System.nanoTime

Description: Return a new angular velocity object using the indicates units and x, y and z rotation rates and an acquisition time of the current system time.

Axis

Introduction:

<numeric value> ← Axis.X

Description: Returns the Axis value X.

<numeric value> ← Axis.Y

Description: Returns the Axis value X.

<numeric value> ← Axis.Z

Description: Returns the Axis value X.

Color

Introduction: Computerized color is modeled as a combination of red, green and blue. Colors also have the attributes alpha, hue, saturation and value.

- Alpha is a measure of transparency, with 0 representing fully transparent and 255 representing opaque.
- Hue a number that describes the color in terms of an angle on a color wheel with red at 0 degrees, green at 120, and blue at 240. See "hue" in Wikipedia. Testing for a range of hue values can be a useful way of using a color sensor.
- Value is brightness with 0 being black and 1 being white.
- Saturation is how strong the color is relative to black and white with 1 representing strong color and small saturation values representing washed out colors. For instance, a bold red would have a saturation of 1. A bright pink might have a value of 1 and saturation of 0.5.

<numeric value> (color intensity) ← Color.[Red OR Green OR blue] color <color value>

Description: Return the color intensity of red, green or blue attribute of the specified color object.

<numeric value> ← Color.Alpha <color object>

Description: Return the alpha attribute of the given color object.

<numeric value> ← Color.Hue <color object>

Description: Return the hue attribute of the given color object.

<numeric value> ← Color.Saturation <color object>

Description: Return the saturation of the given color object.

<numeric value> ← Color.Value <color object>

Description: Return the value attribute of the given color object

<color object> ← call Color.rgbToColor

red <numeric value> green <numeric value> blue <numeric value>

Description: Return a new color object using the given values for red, green and blue

<color object> ← call Color.argbToColor alpha <numeric value>

red <numeric value> green <numeric value> blue <numeric value>

Description: Return a new color object using the given values of alpha, red, green and blue

<color object> ← call Color.hsvToColor hue <numeric value>

saturation <numeric value> value <numeric value>

Description: Return a new color object using the given values of hue, saturation and value

<color object> ← call Color.ahsvToColor alpha <numeric value>

hue <numeric value> saturation <numeric value>

value <numeric value>

Description: Return a new color object using the given values of alpha, hue, saturation and value

<color object> ← call Color.textToColor text <text value>

Description: Return a new color object using the given text value assuming it has the format of #hhhhhh

Note: Each "h" represents a hexadecimal digIt from 0 to 9 and A to F, where A is the hexadecimal equivalent of 10 and F is the hexadecimal equivalent of decimal 15. Each pair of "h"s is the hexadecimal representation an 8-bit binary value. These values run from 00 to FF, which in decimal would be 0 to 255. The first pair of digits in the six-diget hexadecimal value is the red attribute, the second pair the green attribute and the third the blue attribute. For example, "#030F09" represents a color with a value of 3 for red, 15 for green and 9 for blue. See "web colors" in Wikipedia.

<color object> ← call Color.textToColor text <text value>

Description: Return a new color object using the given text value assuming it has the format of #hhhhhhh

Note: The first two hex digits represent alpha or transparency and the remaining digits are as described above.

<color object> ← call Color.textToColor text <text value>

Description: Return a new color object using the given text value assuming it a word describing a color, Valid values are "red", "green", "blue", "yellow", "purple" and "cyan"

DbgLog

call DbgLog.msg message <text value>

Description: Add the text value to the debug log as a message.

call DbgLog.error message <text value>

Description: Add the text value to the debug log as an error.

MagneticFlux

<numeric value> ← MagneticFlux.[X or Y or Z]
magneticFlux <magnetipc flux object>

Description: Return the x, y or z attribute of the magnetic flux object.

<time value> ← MagneticFlux.AcquisitionTime magneticFlux <magnetic flux object>

Description: Return the acquisition time of the magnetic flux object.

<text value> ← call MagneticFlux.toText magneticFlux <magnetic flux variable>

Description: Return the magnetic flux attributes of a magnetic flux object as formatted text string. The format is "Flux: (x, y, z")mT where x, y and z are given in milliTeslas

<magnetic flux object> ← new MagneticFlux

Description: Return a new magnetic flux object.

<magnetic flux object> ← new MagneticFlux
x <numeric value> y <numeric value> z <numeric value>
acquisitionTime call System.nanoTime

Description: Return a new magnetic flux object using the provided x, y and z values in Teslas and the current system time as the acquisition time.

Matrix

Introduction: OpenGL Matrices are 4x4 matrices used for mathematically tracking objects in 3 dimensional space. The first three columns contain rotation information and the fourth contains the position or direction according to whether the last item in the column is a 1 or a 0.

<matrix object> ← call OpenGLMatrix.identityMatrix

Description: Create a new matrix object containing the "identity value" which is 0s in each element except for 1s in the set of cells that form the diagonal from the upper left.

<matrix object> ← call OpenGLMatrix.multiplied
matrix1 <1st matrix object>
matrix2 <2nd matrix object>

Description: Create a matrix object which is the matrix product of the 1st and 2nd matrix objects.

<matrix object> ← call OpenGLMatrix.multiply
matrix1 <matrix variable>
matrix2 <matrix object>

Description: Update the matrix variable with the matrix product of the matrix variable and the matrix object.

<matrix object> ← call OpenGLMatrix.rotation
angleUnit AngleUnit [DEGREES or RADIANS]
angle <numeric value>
dx <numeric value> dy <numeric value> dz <numeric value>

Description: Create a new matrix object which when used in a multiplication will create the effect of rotating the spatial information in the other matrix by the indicated angle specified in the indicated angle units around the vector specified by the values of dx, dy and dz.

<matrix object> ← call OpenGLMatrix.rotation
axesReference AxesReference [EXTRINSIC or INTRINSIC]
axesOrder AxesOrder [XYX or XYZ or ZZX or XZY or YXY or YXZ or YZX or YZY or YZY or ZXY or ZXZ or ZYX or ZYZ]
angleUnit AngleUnit [DEGREES or RADIANS]

firstAngle <numeric value>
secondAngle <numeric value>

thirdAngle <numeric angle>

Description: Create a new matrix object which when used in multiplication will create the effect of rotating the spatial information in the other matrix through three successive angles specified in the indicated angle units around the axes specified in the AxesOrder. Whether the rotations are extrinsic or intrinsic is indicated by the selection after AxesReference.

<matrix object> ← call OpenGLMatrix.rotated
matrix <matrix variable>
angleUnit AngleUnit [DEGREES or RADIANS]
angle <numeric value>
dx <numeric value> dy <numeric value> dz <numeric value>

Description: Rotate the matrix variable by the number of degrees after "angle" around the vector specified by dx, dy, dz.

<matrix object> ← call OpenGLMatrix.rotated matrix <matrix variable>
AxesReference [EXTRINSIC or INTRINSIC]
axesOrder AxesOrder [XYX or XYZ or ZZX or XZY or YXY or YXZ or YZX or YZY or YZY or ZXY or ZXZ or ZYX or ZYZ]
angleUnit AngleUnit [DEGREES or RADIANS]
firstAngle <numeric value>
secondAngle <numeric value>

thirdAngle <numeric angle>

Description: Rotate the matrix variable through three successive rotations around the three axes specified in the AxesOrder. Whether the rotations are extrinsic or intrinsic is indicated by the selection after AxesReference.

<matrix object> ← call OpenGLMatrix.scale
matrix <matrix variable>
scaleX <numeric value>

scaleY <numeric value>

scaleZ <numeric value>

Description: Scale the matrix variable by the amounts specified by scaleX, scaleY and scaleZ.

<matrix object> ← call OpenGLMatrix.scale
matrix <matrix variable> scale <numeric value>

Description: Scale the matrix variable by the numeric value.

<matrix object> ← call OpenGLMatrix.scale matrix <matrix variable>
scaleX <numeric value>

scaleY <numeric value>

scaleZ <numeric value>

Description: Create a new matrix object by scaling the matrix object in the matrix variable by the scaleX, scaleY and scaleZ values.

<matrix object> ← call OpenGLMatrix.scaled matrix <matrix variable>
scale <numeric value>

Description: Create a new matrix object by scaling the matrix object in all three dimensions by numeric value.

<matrix object> ← call OpenGLMatrix.translation
dx <numeric value> dy <numeric value> dz <numeric value>

Description: Create a new matrix object that when used in a multiplication will cause a translation by the dx, dy and dz values. This block can also be used to create a location matrix where dx=x, dy=y, dz=z.

call OpenGLMatrix.translate matrix <matrix variable>
dx <numeric value> dy <numeric value> dz <numeric value>

Description: Translate the matrix variable by the dx, dy and dz values.

<matrix object> ← call OpenGLMatrix.translated matrix <matrix variable>
dx <numeric value> dy <numeric value> dz <numeric value>

Description: Create a new matrix object by translating the matrix object in the matrix variable using the dx, dy and dz values.

<matrix object> ← new OpenGLMatrix

Description: Create a new matrix object.

<matrix object> ← new OpenGLMatrix matrix <matrix variable>

Description: Create a new matrix object that is a copy of the matrix variable.

<numeric value> ← call MatrixF.NumCols matrix <matrix variable>

Description: Return the number of columns in the matrix variable.

<numeric value> ← call MatrixF.NumRows matrix <matrix variable>

Description: Return the number of rows in the matrix variable.

<matrix object> ← call MatrixF.diagonalMatrix
dim <numeric value> scale <numeric value>

Description: Create a square matrix of the indicated dimension with zeroes except the cells in the cells along the diagonal contain the scale value. When used in a multiplication it will have the effect of scaling the other matrix by the amount of the scale value.

<matrix object> ← call MatrixF.diagonalMatrix vector <vector object>

Description: Create a square matrix which contains zeros except along the diagonal which has the values given in the vector object.

<matrix object> ← call MatrixF.identityMatrix dim <numeric value>

Description: Create a square matrix of the specified dimension which contains zeros

except along the diagonal where the values are one.

<numeric value> ← call MatrixF.get matrix <matrix variable>
row <numeric value> col <numeric value>

Description: Return the value of the cell in the matrix variable specified by the row and column values.

call MatrixF.put matrix <matrix variable>
row <numeric value> col <numeric value>
value <numeric value>

Description: Replace the cell in matrix specified by the row and column values with the given value.

<vector object> ← call MatrixF.getRow matrix <matrix variable>
row <numeric value>

Description: Return a matrix object with values taken from the specified row of the matrix variable.

<vector object> ← call MatrixF.getColumn matrix <matrix variable>
col <numeric value>

Description: Return a matrix object with values taken from the specified column of the matrix variable.

<matrix object> ← call MatrixF.added
matrix1 <matrix object> matrix2 <matrix object>

Description: Create a new matrix object which is the sum of matrix1 and matrix2. The two matrices should have the same dimensions.

<matrix object> ← call MatrixF.added matrix <matrix variable> vector <vector object>

Description: Create a new matrix object which is the sum of the matrix variable and the vector object.

call MatrixF.add matrix1 <matrix variable> matrix2 <matrix variable>

Description: Add matrix2 to the matrix1 variable.

call MatrixF.add matrix <matrix variable> vector <vector object>

Description: Add the vector to the matrix variable.

<matrix object> ← call MatrixF.multiplied
matrix1 <matrix variable> matrix2 <matrix variable>

Description: Create a new matrix object with is the product of matrix1 and matrix2.

<matrix object> ← call MatrixF.multiplied
matrix <matrix variable> scale <numeric value>

Description: Create a new matrix object with is the product of matrix and the scale value.

<matrix object> ← call MatrixF.multiplied matrix <matrix variable> vector <vector

object>

Description: Create a new matrix object which is the produce of the matrix variable and the vector.

call MatrixF.multiply matrix1 <matrix variable> matrix2 <matrix object>

Description: Update the matrix variable by multiplying by matrix2.

call MatrixF.multiply matrix <matrix variable> scale <numeric value>

Description: Update the matrix variable by multiplying by the scale value.

call MatrixF.multiply matrix <matrix variable> vector <vector object>

Description: Update the matrix variable by multiplying by the vector object.

call MatrixF.slice matrix <matrix variable> row <numeric value> col <numeric value>
numRows <numeric value> numCols <numeric value>

Description: Create a new matrix object which is the sub-matrix of the provided matrix starting at the specified row and column and has the specified number of rows and columns.

<matrix object> ← call MatrixF.subtracted
matrix1 <matrix object> matrix2 <matrix variable>

Description: Create a new matrix object which is the result of subtracting matrix2 from matrix1.

<matrix object> ← call MatrixF.subtracted
matrix <matrix object> vector <vector object>

Description: Created a new matrix object which is the result of subtracting the vector object from the matrix

call MatrixF.subtract matrix1 <matrix variable> matrix2 <matrix object>

Description: Update the matrix1 variable by subtracting matrix2.

call MatrixF.subtract matrix <matrix variable> vector <vector object>

Description: Update the matrix variable by subtracting the vector object.

<matrix object> ← call MatrixF.inverted matrix <matrix object>

Description: Create a new matrix object by inverting the provided matrix object.

<matrix object> ← call MatrixF.transform
matrix <matrix object> vector <vector object>

Description: Create a new matrix object by transforming the provided matrix using the vector object.

<matrix object> ← call MatrixF.transposed matrix <matrix variable>

Description: Create a new matrix which is the transposition of the provided matrix object.

<text value> ← call MatrixF.toText matrix <matrix object>

Description: Return a text value which contains the formatted contents of the provided matrix object. The overall matrix is enclosed in {…}. The rows are formatted one after another, each row is enclosed in {…} and the values and the rows are separated by commas. So, if ricj represents the value in row i column j, the text value produced by this block is

{{r0c0,r0c1,r0c2,r0c3},{r1c0,r1c1,r1c2,r1c3},{r2c0,r2c1,r2c2,r2c3},{r3c0,r3c1,r3c2,r3c3}

Example: {{1.000,0.000,0.000,10.000},
{0.000,1.000,0.000,20.000},
{0.000,0.000,1.000,30.000},
{0.000,0.000,0.000,1.000}}

<text value> ← call Matrix.formatAsTransform matrix <matrix variable>

Description: Return a text value which contains the formatted contents of the provided matrix object treating it as a transform matrix. Four 4x4 matrics, the format is {EXTRINSIC abc r1 r2 r3}{dx dy dz} where EXTRINSIC indicates the transformation is extrinsic; abc indicates the axis order of the transformation and r1, r2, r3 are the three rotation angles in degrees; and dx, dy, dz are the three translation values. For example if abc is XYZ, r1 specifies the rotation around the X axis, r2 around the Y axis and r3 around the z axis. If the axes reference is intrinsic that will be indicated as INTRINSIC in place of EXTRINSIC.

<text value> ← call Matrix.formatAsTransform matrix <matrix variable>
AxesReference [EXTRINSIC or INTRINSIC]
axesOrder AxesOrder [XYX or XYZ or ZZX or XZY or YXY or YXZ or YZX or YZY or YZY or ZXY or ZXZ or ZYX or ZYZ]
angleUnit AngleUnit [DEGREES or RADIANS]

Description: Return a text value which contains the formatted contents of the provided matrix object treating it as a transform matrix. The format is {EXTRINSIC abc r1 r2 r3} {dx dy dz} in place of abc the axes order is provided. The three r numbers are the three rotation angles and dx, dy, dz are the translation values. If the axes reference is INTRINSIC in place of EXTRINSIC.

<vector object> ← call MatrixF.toVector matrix <matrix variable>

Description: Return a vector object which based on the only row or column of the provided matrix. Assumes the provided matrix has either only one row or one column.

<vector object> ← call MatrixF.getTranslation matrix <matrix variable>

Description: Returns the transformation component of the given matrix as a vector. Assumes the matrix contains a non-perspective transformation.

Orientation

The blocks in this folder allow you to manipulate data relating to orientation in 3-dimensional space. These blocks allow you to create orientation objects, manipulate these objects, and obtain information from these objects. For background on orientation calculations see Euler angles in Wikipedia.

<axes reference value> ← Orientation.AxesReference orientation <orientation object>

Description: Return the axes reference of the given orientation object.

<axes order value> ← Orientation.AxesOrder orientation <orientation object>
Description: Return the axes order of the given orientation object.

<angle unit value> ← Orientation.AngleUnit orientation <orientation object>

Description: Return the angle unit value of the given orientation object.

<numeric value> ← Orientation.FirstAngle orientation <orientation object>

Description: Return the first angle of the given orientation object.

<numeric value> ← Orientation.SecondAngle orientation <orientation object>

Description: Return the second angle of the given orientation object.

<numeric value> ← Orientation.ThirdAngle orientation <orientation object>
Description: Return the third angle of the given orientation object.

<time value> ← Orientation.AcquisitionTime orientation <orientation object>

Description: Return the acquisition time value of the given orientation object>

<orientation object> ← call Orientation.toAngleUnit
orientation <orientation object>
angleUnit AngleUnit [DEGREES or RADIANS]

Description: Return new orientation object based on the given orientation object and the angle unit to degrees or radians.

<orientation object> ← Orientation.toAxesReference
orientation <orientation object>
axesReference AxesReference [EXTRINSIC or INTRINSIC]

Description: Return new orientation object based on the given orientation object and the axes reference of either extrinsic or intrinsic.

<orientation object> ← Orientation.toAxesOrder orientation <orientation object>
axesOrder AxesOrder [XYX or XYZ or ZZX or XZY or YXY or YXZ or
YZX or YZY or YZY or ZXY or ZXZ or ZYX or ZYZ]
Description: Return new orientation object based on the given orientation object and the specified axes order.

<text value> ← call Orientation.toText orientation <orientation object>

Description: Return text representation of given orientation object.

<matrix object> ← call Orientation.getRotationMatrix orientation <orientation object>

Description: Return the rotation matrix associated with the given orientation object.

<matrix object> ← call Orientation.getRotationMatrix
axesReference AxesReference [EXTRINSIC or INTRINSIC]
axesOrder AxesOrder [XYX or XYZ or ZZX or XZY or YXY or YXZ
or YZX or YZY or YZY or ZXY or ZXZ or ZYX or ZYZ]

angleUnit AngleUnit [DEGREES or RADIANS]
firstAngle <numeric value>
secondAngle <numeric value>
thirdAngle <numeric angle>

Description: Return the rotation matrix associated with a given set of three rotation angles.

<orientation object> ← new Orientation

Description: Return new orientation object.

<orientation value> ← new Orientation AxesReference [EXTRINSIC or INTRINSIC]
axesOrder AxesOrder [XYX or XYZ or ZZX or XZY or YXY or YXZ or YZX or YZY or YZY or ZXY or ZXZ or ZYX or ZYZ]
angleUnit AngleUnit [DEGREES or RADIANS]
firstAngle <numeric value>
secondAngle <numeric value>
thirdAngle <numeric angle>
acquisitionTime call System.nanoTime

Description: Return new orientation object based on the information provided.

<orientation value> ← call Orientation.getOrientation matrix <matrix variable>
AxesReference [EXTRINSIC or INTRINSIC]
axesOrder AxesOrder [XYX or XYZ or ZZX or XZY or YXY or YXZ or YZX or YZY or YZY or ZXY or ZXZ or ZYX or ZYZ]
angleUnit AngleUnit [DEGREES or RADIANS]

Description: Return new orientation object that would produce the specified rotation based on a rotation matrix and the given axes reference and axes order.

Position

<distance unit value> ← Position.DistanceUnit position <position object>
Description: Return distance unit in use by given position object.

<numeric value> ← Position.X position <position object>
Description: Return x component of the position represented by the position object.

<numeric value> ← Position.Y position <position object>
Description: Return y component of the position represented by the position object.

<numeric value> ← Position.Z position <position object>
Description: Return z component of the position represented by the position object.

<time value> ← Position.AcquisitionTime position <position object>
Description: Return acquisition time associated with the position object.

<position object> ← call Position.toDistanceUnit position <position object>
distanceUnit DistanceUnit [CM or INCH or METER or MM]
Description: Return a new position object based on a given position object and a new distance unit.

<text value> call Position.toText position <position object>

Description: Return a text representation of a given position object.

<position object> ← new Position
Description: Return a new position object>

<position value> ← new Position distanceUnit
DistanceUnit [CM or INCH or METER or MM]
x <numeric value> y <numeric value> z <numeric value>
acquisitionTime call System.nanoTime
Description: Return a new position object using the given distance using and x, y and z values using the current system time as the acquisition time.

Quaternion

Introduction: Quaternion are four-dimensional objects which can be used to calculate rotations in three-dimensional space. For additional background see Quaternion in Wikipedia.

<numeric value> ← Quaternion.W quaternion <quaternion object>
Description: Return the w component of the given quaternion object.

<numeric value> ← Quaternion.X quaternion <quaternion object>
Description: Return the x component of the given quaternion object.

<numeric value> ← Quaternion.Y quaternion <quaternion object>
Description: Return the y component of the given quaternion object.

<numeric value> ← Quaternion.Z quaternion <quaternion object>
Description: Return the z component of the given quaternion object.

<time value> ← Quaternion.AcquisitionTime quaternion <quaternion object>
Description: Return the acquisition time associated with the given quaternion object.

<quaternion object> ← call Quaternion.normalized quaternion <quaternion object>
Description: Return a new quaternion object that is normalized from the given quarternion object.

<quaternion object> ← call Quaternion.congugate quaternion <quaternion object>
Description: Return a new quaternion object based is the conjugate of the given quaternion object.

<quaternion object> ← new Quaternion
Description: Return new quaternion object.

<quaternion object> ← new Quaternion
w <numeric value>
x <numeric value>
y <numeric value>
z <numeric value>
acquisitionTime call System.nanotime
Description: Return new quarternion object based on the w, x, y and z values provided using the current system time as the acquisition time.

Telemetry

call telemetry.addData key <text value> number <numeric value>

Description: Send key and numeric value to Driver Station for display.

call telemetry.addData key <text value> text <text value>

Description: Send key and text value to Driver Station for display.

call Telemetry.update

Description: Send accumulated telemetry data and cause Driver Station to display it.

Temperature

<temperature unit> ← Temperature.TempUnit temperature <temperature object>
Description: Return the temperature unit associated with the given temperature object.

<numeric value> ← Temperature.Temperature temperature <temperature object>
Description: Return the temperature associated with the given temperature object.

<time value> ← Temperature.AcquistionTime temperature <temperature object>
Description: Return the time associated with the given temperature object.

<temperature object> ← new.Temperature
Description: Return new temperature object.

<temperature value> ← new.Temperature tempUnit
TempUnit [CELSIUS or FARENHEIT or KELVIN]
temperature <numeric value>
acquisitionTime call System.nanoTime
Description: Return new temperature object based on the given values.

<temperature object> ← call Temperature.toTempUnit
temperature <temperature object>
tempUnit TempUnit [CELSIUS or FARENHEIT or KELVIN]
Description: Return new temperature object based on given temperature object and new temperature unit.

Tensor Flow Object Detection

Introduction: TensorFlow Object Detection gives the ability use either of the cameras on the Robot Controller phone to identify and track game elements like the Rover Ruckus gold and silver minerals. This feature formally known as Google TensorFlow Lite Object Detection, which is a lightweight version of Google's TensorFlow machine learning technology that is designed to run on mobile devices such as an Android smartphone. A trained TensorFlow model was developed to recognize the Gold and Silver Mineral game pieces from the 2018-2019 FIRST Tech Challenge game. This TensorFlow model has been integrated into the FIRST Tech Challenge Control System software and can be used to identify and track these game pieces during a match. When an object is detected the system determines a “bounding box” around the object and the coordinates and dimensions of the box become available. The TensorFlow Lite technology requires Android version 6.0 also known as Marshmallow or a more recent version. ZTE phones use an older Android version which does not support TensorFlow, while more recent phones like the Motorola Moto G4 use a more recent

version of Android and support TensorFlow Lite.

Optimized for Rover Ruckus

Introduction: These blocks are specific to the Rover Ruckus season. The TensorFlowObjectDetection system can only be initialized if the Vuforia system has been initialized because the TensorFlow system depends on some of the Vuforia system.

call TensorFlowObjectDetection.initialize
minimumConfidence <numeric value>
useObjectTracker <logic value>
enabled CameraMonitoring <logic value>

Description: Initializes the Rover Ruckus version of the Tensor Flow Objection Detection system. Objects will be reported as detected only if the probability that it corrected detected is greater than the minimum confidence level specified by the numeric value. For instance, a minimumConfidence level of 0.4 means that an object will only be considered detected if the probability that it has been correctly detected is 40% or greater. The system interprets images based on the phone's orientation (either Portrait or Landscape) at the time that the TensorFlow object detector was created and initialized. In our example, if you execute the TensorFlowObjectDetection.initialize block while the phone is in Portrait mode, then the images will be processed in Portrait mode. The "Left" and "Right" values of an object's bounding box correspond to horizontal coordinate values, while the "Top" and "Bottom" values of an object's bounding box correspond to vertical coordinate values. If the phone is in Landscape mode when the object detector is initialized, then the images will be interpreted in Landscape mode. Android devices can be locked into Portrait Mode so that the screen image will not rotate even if the phone is held in a Landscape orientation. If your phone is locked in Portrait Mode, then the TensorFlow object detector will interpret all images as Portrait images. If you would like to use the phone in Landscape mode, then you need to make sure your phone is set to "Auto-rotate" mode. In Auto-rotate mode, if the phone is held in a Landscape orientation, then the screen will auto rotate to display the contents in Landscape form.

call TensorFlowObjectDetection.activate
Description: Activates the Tensor Flow system using processor and memory resources.

call TensorFlowObjectDetection.deactivate
Descriptions: Deactivates the Tensor Flow system releasing system resources.

<recognition object> ← call TensorFlowObjectDetection.getRecognition
Description: Returns a list of current object detections.

Recognition

Description: These blocks provide the rest of the Tensor Flow features. Some of these blocks return distances and dimensions in pixels. The Tensor Flow image is typically not the full resolution of the camera. For instance, the image might be 800 pixels wide by 480 pixels high in landscape mode. The dimensions of the Tensor Flow image are available via two of the blocks described below.

<label value> ← Recognition.Label recognition <recognition variable>
Description: Returns a label value describing the object. For Rover Ruckus the value will be either Gold Mineral or Silver Mineral.

Label.Gold Mineral
Description: Gold Mineral value. This value can be used to compare to the value returned by Recognition.Label block.

Label.Silver Mineral
Description: Sliver Mineral value. This value can be used to compare to the value returned by Recognition.Label block.

<numeric value> ← Recognition.Confidence recognition <recognition variable>
Description: Returns the confidence level associated with the particular recognition. The confidence level is a probability from 0% to 100% (0.0 to 1.0) that describes the system's confidence that the object is actually what is described by the label value.

<numeric value> ← Recognition.Left recognition <recognition variable>
Description: Returns the distance from the left edge of the video image to the left edge of the recognized object in pixels.

<numeric value> ← Recognition.Right recognition <recognition variable>
Description: Returns the distance in pixels from the left edge of the video image to the right edge of the recognized object.

<numeric value> ← Recognition.Top recognition <recognition variable>
Description: Returns the distance in pixels from the top edge of the video image to the top edge of the recognized object.

<numeric value> ← Recognition.Bottom recognition <recognition variable>
Description: Returns the distance in pixels from the top edge of the video image to the bottom edge of the recognized object.

<numeric value> ← Recognition.Width recognition <recognition variable>
Description: Returns the width in pixels of the detected object.

<numeric value> ← Recognition.Height recognition <recognition variable>
Description: Returns the height in pixels of the detected object.

<numeric value> ← Recognition.ImageWidth recognition <recognition variable>
Description: Returns the width in pixels of the Tensor Flow image.

<numeric value> ← Recognition.ImageHeight recognition <recognition variable>
Description: Returns the height in pixels of the Tensor Flow image.

<text value> ← call Recognition.toText recognition <recognition variable>
Description: Returns a text value that includes all of the above information.

<numeric value> ← call Recognition.estimateAngleToObject
recognition <recognition variable>
angleUnit AngleUnit [DEGREES or RADIANS]
Description: Returns an estimate of the horizontal angle to the detected object.

Time

ElapsedTime

<elapsed time object> ← new ElapsedTime Description: Create a new elapsed time object.

<elapsed time object> ← new ElapsedTime startTime call System.nanoTime

Description: Create a new elapsed time object with seconds resolution and initialize it to the current system time or a value that you plug in in its place.

<elapsed time object> ← new ElapsedTime resolution
Resolution [SECONDS or MILLISECONDS]

Description: Create a new elapsed time object with the current system time with a resolution of SECONDS or MILLISECONDS.

<numeric value> ← ElapsedTime.StartTime timer <elapsedTime variable>

Description: Return the start time of the specified elapsed time object using the current resolution of the elapsed time object.

<numeric value> ← ElapsedTime.Time timer <elapsedTime variable>

Description: Return the elapsed time since the last reset of the specified elapsed time object in the resolution of the elapsed time object.

<numeric value> ← ElapsedTime.seconds timer <elapsedTime variable>

Description: Return the elapsed time in seconds since the last reset of the elapsed time object.

<numeric value> ← ElapsedTime.Milliseconds timer <elapsedTime variable>

Description: Return the elapsed time in milliseconds since the last rest of the elapsed time object.

<numeric value> ← ElapsedTime.Resolution timer <elapsedTime variable>

Description: Return the current resolution of the elapsed time object – either SECONDS or MILLISECONDS.

call ElapsedTime.reset timer <elapsedTimeVariable>

Description: Reset the specified elapsed time object.

call ElapsedTime.log timer <elapsedTimeVariable> label <text value>

Description: Log a message indicating how long the elapsed time object has been running using the indicated <text value>,

<text value> ← call ElapsedTime.toText timer <elapsedTimeVariable>

Description: Return a text string giving the elapsed time of the elapsed time object.

Vector

<numeric value> ← VectorF.Length vector <vector object>
Description: Return the length of the given vector object.

<numeric value> ← VectorF.Magnitude vector <vector object>
Description: Return the magnitude of the given vector object.

<numeric value> ← call VectorF.get vector <vector object> index <numeric value>
Description: Return the component of given vector specified by the index value.

call VectorF.put vector <vector object>
index <numeric value>
value <numeric value>
Description: Update the given vector by update the component specified by the index value and the given value.

<text value> ← call VectorF.toText vector <vector object>
Description: Return a text representation of the given vector object.

Velocity

<distance unit> ← Velocity.DistanceUnit velocity <velocity object>
Description: Return the distance unit used in the velocity object.

<numeric value> ← Velocity.XVeloc velocity <velocity object>
Description: Return the x attribute of the velocity object.

<numeric value> ← Velocity.YVeloc velocity <velocity object>
Description: Return the y attribute of the velocity object.

<numeric value> ← Velocity.ZVeloc velocity <velocity object>
Description: Return the z attribute of the velocity object.

<time value> ← Velocity.AcquisitionTime velocity <velocity object>
Description: Return the acquisition time of the velocity object.

<velocity object> ← call Velocity.toDistanceUnit velocity <velocity object>
distanceUnit DistanceUnit [CM or INCH or METER or MM]
Description: Return a velocity object based on the given velocity object after converting to the indicated units.

<text value> ← call Velocity.toText velocity <velocity object>
Description: Return the velocity attributes of the velocity object as a formatted string. The format of the string is “Velocity: (x y z)units/s” where x, y and z are the attributes of the velocity in the indicated units per second. For example if x=2.5 cm/s, y=3.0 cm/s and z=5.0 cm/s the string is “Velocity: (2.500 3.000 5.000)cm/s”

<velocity object> ← new Velocity

Description: Return a new velocity object.

<velocity object> ← new Velocity
distanceUnit DistanceUnit [CM or INCH or METER or MM]
XVeloc <numeric value> YVeloc <numeric value> ZVeloc <numeric value>
acquisitionTime call System.nanotime

Description: Return a new velocity using the x, y and z values provided in those units and the current system time as the acquisition time.

Vuforia

Introduction: Vuforia gives the ability to use the either of the cameras on the Robot Controller phone to locate the four FTC targets on the field walls and use them to determine your robot's position on the field. For information on the FTC Field Coordinate System see http://tinyurl.com/FTCFieldCo

Optimized for Rover Ruckus

Introduction: Later sections provide complete access to the Vuforia features. This section provides an optimized interface for discovering the location of your robot or the relative locations of the four FTC trackable targets. The blocks in this section should provide faster performance than those in the later sections.

Call Vuforia.initialize
vuforiaLicenseKey <text value> (As of Version 3.3, no longer needed)
cameraDirection CameraDirection [BACK or FRONT]
useExtendedTracking <logic value>
enableCameraMonitoring <logic value>
cameraMonitorFeedback CameraMonitorFeedback [DEFAULT or NONE or AXES or TEAPOT or BUILDING]
phoneLocationOnRobot translation dx <numeric value>
phoneLocationOnRobot translation dy <numeric value>
phoneLocationOnRobot translation dz <numeric value>
phoneLocationOnRobot rotation x <numeric value>
phoneLocationOnRobot rotation y <numeric value>
phoneLocationOnRobot rotation z <numeric value>
useCompetitionFieldTargetLocations <logic value>

Description: Initializes Vuforia using the values provided. The license key is obtained by creating a free account on Vuforia.com and requesting a key. Copy the long text string and paste it into the first parameter of this block. Note: Starting with Version 3.3 of the Robot Controller app, a license key is no longer required to use this block. CameraDirection allows you to specify which camera on the Robot Controller phone you wish to use. Specify BACK if you wish to use the camera on the back of the Android device, FRONT if use the camera on the front of the device. The useExtendedTracking parameter allows you to request that Vuforia attempt to continue its tracking even when a trackable (FTC target) is not visible. The enableCameraMonitoring parameter determines whether the camera image and any tracking information is displayed on the Robot Controller Android device. cameraMonitorFeedback allows you to specify what type of monitoring you want on the Driver Station. Specify DEFAULT or NONE or AXES or

TEAPOT or BUILDING. AXES indicates you want X, Y, and Z axes displayed. Specify TEAPOT or BUILDING if you want the corresponding image to be used. If you specify Default, AXES will be used. Provide dx, dy and dz values in millimeters to indicate the position of the camera lens relative to the center of your robot. Provide rotation x (pitch), y (roll) and z (yaw) angle values to indicate how your Android device is mounted on your robot. x=0 indicates it is mounted vertically. y=0 indicates the device is mounted in portrait orientation. z=0 indicates the lens being used is facing forward relative to the robot. useCompetitionFieldTargetLocations allows you to indicate that you want the tracking to assume that the four FTC target images are located in their standard locations on the field walls. If so, specify True. Otherwise, the calculations will be done as if all four of the targets are mounted at the center of the field. [x = 0, y = 0, z = 0] and navigation information provided by the "call Vuforia.track" block (described below) will be relative to the current target.
Note: During the RELIC RECOVERY season, useCompetitionFieldTargetLocations should be set to False. This is because the four targets are the same during each round and thus Vuforia cannot be used to determine the position of the Robot Controller phone and the robots location on the field. If you wish to use targets for navigation you should set useCompetitionFieldTargetLocations to false and use the "call Vuforia.track" block (described below) to obtain the location of the robot relative to the current target (i.e. as if the targets are located at the center of the field). Alternatively and easier, you can use the "call Vuforia.trackPose" block to obtain the location of the robot relative to the robot. If you use trackPose the setting of useCompetitionFieldTargetLocations does not matter.

call Vuforia.activate
Description: Activates Vuforia system. This should be used once during the initialization portion of your op mode.

call Vuforia.deactivate
Description: Deactivates Vuforia system. This should be used at the end of your op mode.

<vuforia tracking results object> ← call Vuforia.track
trackableName TrackableName [RELIC]
Description: Returns a tracking results object using the specified FTC target. During the RELIC RECOVERY season, the only valid target name is RELIC. This block should be called each time an op mode needs an update on where the robot is on the field. If useCompetitionFieldTargetLocations is True the results are given as the position and orientation the robot on the competition field. Otherwise, the position is given relative to the target. Note: As discussed above, setting useCompetitionFieldTargetLocations to True during RELIC RECOVERY season will not work.

<vuforia tracking results object> ← call Vuforia.trackPose
trackableName TrackableName [RELIC]
Description: Returns a tracking results object of the pose of the specified FTC target. The pose is the location of the target in the phone's coordinate system, i.e. relative to the current location of the Robot Controller phone. During the RELIC RECOVERY season, the only valid target name is RELIC.

<logic value> ← VuforiaTrackingResults.IsVisible
vuforiaTrackingResults <vuforiaTrackingResults variable>
Description: Returns true if trackable described by TrackingResults variable is visible. Returns false otherwise.

<VuMark value> ← VuforiaTrackingResults.RelicRecoveryVuMark
vuforiaTrackingResults <vuforiaTrackingResults object>
Description: Returns VuMark value corresponding to any VuMark pattern that is being tracked.

<numeric value> ← VuforiaTrackingResults.X
vuforiaTrackingResults <vuforiaTrackingResults object>
Description: If the TrackingResults object was returned by Vuforia.track block, this block returns the position of the robot in the X dimension. X is millimeters (mm) along wall the red alliance wall, increasing towards the blue alliance. If useCompetitionFieldTargetLocations is True, 0 is center of wall and X is -1790 at the wall opposite the blue alliance and 1790 at the blue alliance wall.

If the TrackingResults object was returned by Vuforia.trackPose block and the phone is mounted horizontally on the robot, the x value is the up-down location of the target relative to the Robot Controller Android device with positive values being millimeters (mm) below the camera lens and negative values above.

<numeric value> ← VuforiaTrackingResults.Y
vuforiaTrackingResults <vuforiaTrackingResults object>
Description: If the TrackingResults object was returned by Vuforia.track block, this block returns the position of the robot in the Y dimension. Y is mm along the blue alliance wall. If useCompetitionFieldTargetLocations is True, 0 is the center of the wall and Y is -1790 at the red alliance wall and 1790 at the wall opposite the red alliance.

If the TrackingResults object was returned by Vuforia.trackPose block and the phone is mounted horizontally on the robot, the y value is the left-right location of the target relative to the Robot Controller Android device with negative values being millimeters (mm) to the left and positive being mm to the right.

<numeric value> ← VuforiaTrackingResults.Z
vuforiaTrackingResults <vuforiaTrackingResults object>
Description: If the TrackingResults object was returned by Vuforia.track block, this block returns the position of the robot in the Z dimension. Z is mm from floor.

If the TrackingResults object was returned by Vuforia.trackPose block and the phone is mounted horizontally on the robot, the z value is distance to the target relative from the Robot Controller Android device with negative values being millimeters in front of the camera lens.

<numeric value> ← VuforiaTrackingResults.XAngle

vuforiaTrackingResults <vuforiaTrackingResults variable>
Description: If the TrackingResults object was returned by Vuforia.track block, this block returns the x angle to the target described by the variable. X is the pitch angle in degrees relative to the target. 90 means the target is perpendicular to the robot and parallel to the camera lens. Less than 90 means target is below. Greater than 90 means the target is above.

If the TrackingResults object was returned by Vuforia.trackPose block and the phone is mounted horizontally on the robot, the x angle is 0 degrees when the target is parallel to to the Robot Controller Android device. When the center the target is above the camera lens, the x angle is negative. When the center of the target is below, the x angle is positive.

<numeric value> ← VuforiaTrackingResults.YAngle
vuforiaTrackingResults <vuforiaTrackingResults variable>
Description: If the TrackingResults object was returned by Vuforia.track block, this block returns the y angle relative to the target described by the variable. Y angle is robot's and camera's roll angle in degrees relative to the target. 0 means robot is level if phone is mounted vertically (camera is in portrait orientation). Positive angles mean the robot and camera have rolled clockwise relative to target. Negative angles mean the robot and camera have turned counterclockwise relative to the target.

If the TrackingResults object was returned by Vuforia.trackPose block and the phone is mounted horizontally on the robot, the y angle is 0 degrees when the target is straight ahead of the Robot Controller Android device's camera lens. When the target is to the left relative to the camera lens, the y angle is negative. When the target is to the right, the y angle is positive.

<numeric value> ← VuforiaTrackingResults.ZAngle
vuforiaTrackingResults <vuforiaTrackingResults variable>
Description: If the TrackingResults object was returned by Vuforia.track block, this block returns the Z angle of the robot on the field. The Z angle is the yaw angle in degrees relative to field where 0 means the wall opposite red alliance is straight ahead. Positive angles mean the robot has turned away from the blue alliance wall (i.e. it has yawed left – counterclockwise relative to that wall). Negative angles mean the robot has turned toward the blue alliance wall.

If the TrackingResults object was returned by Vuforia.trackPose block and the phone is mounted horizontally on the robot, the z angle is 90 degrees when the roll angle of the target is same as the Robot Controller Android device. When the target is to the rolled left the angle is greater than 90. When it is rolled to the right it is less than 90.

<text value> ← call VuforiaTrackingResults.formatAsTransform
vuforiaTrackingResults <vuforiaTrackingResults variable>
Description: Returns text value that is a formatted version of the tracking results for the trackable described by the tracking results variable.

RelicRecoveryVuMark

Introduction: During the 2017-18 RELIC RECOVERY season, Blocks Programming has four special values. Three of them correspond to the three VuMark patterns for the season – LEFT, CENTER and RIGHT. The fourth value – UNKNOWN – is the value that is returned by Vuforia Tracking Results.RelecRecoveryVuMark block when none of the three patterns has been recognized.

<VuMark value> ← RelicRecoveryVuMark.UNKNOWN

<VuMark value> ← RelicRecoveryVuMark.LEFT

<VuMark value> ← RelicRecoveryVuMark.CENTER

<VuMark value> ← RelicRecoveryVuMark.RIGHT

VuforiaLocalizer

<localizer object> ← new VuforiaLocalizer
vuforiaLocalizerParameters <vuforia variable>

Description: Return new Vuforia localizer object using the provided Vuforia parameters variable .See the next section for functions for creating such variables.

<trackables object> ← call VuforiaLocalizer.loadTrackablesFromAsset
vuforiaLocalizer <vuforia localizer variable>
assetName <text value>

Description: Return a Vuforia trackables object after loading a trackable dataset from the asset indicated by the assetName text value into the provided localizer variable. The assetName must describe an XML and DAT pair with the indicated assetName. This operation can take a few seconds to execute. Assets are bound into the application APK file when it is compiled. The standard Robot Controller application available from Google Play Store includes the four FTC targets as assets. To load these assets specify "FTC_2016-17" as the assetName.

<trackables object> ← call VuforiaLocalizer.loadTrackablesFromFile
vuforiaLocalizer <vuforia localizer variable>
absoluteFileName <text value>

Description: Return a Vuforia trackables object after loading a trackable dataset from the asset indicated by the assetName text value into the provided localizer variable. The absoluteFileName must describe a pair of files and provide the full path to these files. The first file is an XML with a name that starts with file at the end of the path given by the indicated absolute File name and ends in .XML. The XML file specifies which images are described by the DAT file. The DAT file must be in the same directory as the XML file and be a Vuforia data file containing one or more images. This operation can take a few seconds to execute.

VuforiaLocalizerParameters

<vuforia localizer parameters object> ← new VuforiaLocalizer.Parameters

Description: Return Vuforia localizer parameters object with default values.

call VuforiaLocalizer.Parameters.setVuforiaLicenseKey
vuforiaLocalizerParameters <vuforia localizer parameters variable>

vuforiaLicenseKey <text value>

Description: Set the Vuforia License Key in the provided Vuforia localize parameters variables to the given text value. Obtain a license key by creating a free account on Vuforia.com and requesting a license key.

call VuforiaLocalizer.Parameters.setCameraDirection
vuforiaLocalizerParameters <vuforia localizer parameters variable>
cameraDireciton CameraDirection [BACK or FRONT]

Description: Set the camera direction in the provided Vuforia localizer parameters variable. Specify BACK if you wish to use the camera on the back of the Android device, FRONT if use the camera on the front of the device.

call VuforiaLocalizer.Parameters.setUseExtendedTracking
vuforiaLocalizerParameters <vuforia localizer parameters variable>
useExtendedTracking [true or false]

Description: Set the Use Extended Tracking attribute in the provided Vuforia localizer parameters variable to true or false. Extended tracking allows tracking to continue even when a target image is no longer in view of the camera.

call VuforiaLocalizer.Parameters.setEnableCameraMonitoring
vuforiaLocalizerParameters <vuforia localizer parameters variable>
enableCameraMonitoring [true or false]

Description: Set the EnableCameraMonitoring attribute of the provided Vuforia localizer parameters variable to true or false. This attribute determines whether the camera image and any tracking information is displayed on the Robot Controller Android device.

call VuforiaLocalizer.Parameters.setCameraMonitoringFeedback
vuforiaLocalizerParameters <vuforia localizer parameters variable>
CameraMonitoringFeedback <camera monitor feedback value>

Description: Set the CameraMonitoringFeedback attribute of the provided Vuforia localizer parameters. Valid values are CameraMonitorFeedback.DEFAULT, CameraMonitorFeedback.NONE, CameraMonitorFeedback.AXES, CameraMonitorFeedback.TEAPOT, CameraMonitorFeedback.BUILDINGS. The AXES value will cause the x, y and z axes to the current target to be displayed.

call VuforiaLocalizer.Parameters.setFillCameraMonitorViewParent
vuforiaLocalizerParameters <vuforia localizer parameters variable>
fillCameraMonitorViewParent [true or false]

Description: Set the FillCameraMonitorViewParent attribute in the provided Vuforia localizer parameters variable to true or false. True calls the camera image to fill the monitor screen possibly preventing some data to be displayed. False assures that all available data is displayed.

<text value> ← VuforiaLocalizer.Parameters.VuforiaLicenseKey
vuforiaLocalizerParameters <vuforia localizer parameters variable>

Description: Return the license key from the provided Vuforia Localizer parameters variable.

<camera direction object> ← VuforiaLocalizer.Parameters.CameraDirection
vuforiaLocalizerParameters <vuforia localizer parameters variable>

Description: Set the CameraDirection attribute of the provided Vuforia localizer parameters variable. Valid values are

<logic value> ← VuforiaLocalizer.Parameters.UseExtendedTracking
vuforiaLocalizerParameters <vuforia localizer parameters variable>

Description: Return the current UseExtendedTracking value – either true or false.

<logic value> ← VuforiaLocalizer.Parameters.EnableCameraMonitoring
vuforiaLocalizerParameters <vuforia localizer parameters variable>

Description: Returns the current value of EnableCameraMonitoring attribute – either true or false.

<camera monitor feedback value>
← VuforiaLocalizer.Parameters.CameraMonitorFeedback
vuforiaLocalizerParameters <vuforia localizer parameters variable>

Description: Return the current value of the ParametersCameraMonitorFeedback attribute. Valid values are CameraMonitorFeedback.DEFAULT, CameraMonitorFeedback.NONE, CameraMonitorFeedback.AXES, CameraMonitorFeedback.TEAPOT, and CameraMonitorFeedback.BUILDINGS

<logic value> ← VuforiaLocalizer.Parameters.FillCameraMonitorViewParent
vuforiaLocalizerParameters <vuforia localizer parameters variable>

Description: Return the current value of the FillCameraMonitorViewParent attribute – either true or false.

VuforiaTrackable

<matrix object> ← VuforiaTrackable.Location
vuforiaTrackable <vuforia trackable variable>

Description: Return a matrix object containing the location of the specified trackable in the FTC coordinate system.

<numeric value> ← VuforiaTrackable.UserData
vuforiaTrackable <vuforia trackable variable>

Description: Return user data previously associated with the speified trackable.

<trackables object> ← VuforiaTrackable.Trackables
vuforiaTrackable <vuforia trackable variable>

Description: Return trackables object that is associated with the specified trackable.

<text value> ← VuforiaTrackable.Name
vuforiaTrackable <vuforia trackable variable>

Description: Return the user-specified name for the specified trackable.

<listener object> ← VuforiaTrackable.Listener vuforiaTrackable <vuforia trackable

variable>

Description: Return the listener object associated with specified trackable.

call VuforiaTrackable.setLocation
vuforiaTrackable <vuforia trackable variable>
matrix <matrix variable>

Description: Set location of the specified trackable in FTC field coordinate system based on the specified matrix.

call VuforiaTrackable.setUserData
vuforiaTrackable <vuforia trackable variable>
userData <numveric value>

Description: Set user data attribute in specified trackable to the specified value.

call VuforiaTrackable.setName
vuforiaTrackable <vuforia trackable variable> name <text value>

Description: Set the name of the specified trackable to the specified text. This can be useful for debugging.

VuforiaTrackableDefaultListener

call VuforiaTrackableDefaultListener.setPhoneInformation
vuforiaTrackableDefaultListener <vuforia trackable default listener variable>
phoneLocationOnRobot <matrix variable>
cameraDirection CameraDirection [BACK or FORWARD]

Description: Set the phone information of the specified default listener to the location on the robot given by the matrix and camera direction to BACK or FORWARD. The format of the matrix is a OpenGLMatrix. Default values are dx=0, dy=0, dz = 0, X angle = 0, Y angle = 0 and Z angle = 0. (Camera lens is at center of robot, mounted facing forward in portrait orientation.)

<logic value> ← call VuforiaTrackableDefaultListener.isVisable
vuforiaTrackableDefaultListener <vuforia trackable default listener variable>

Description: Return true if the trackable associated with the default listener is current visible, false otherwise.

<matrix object> ← VuforiaTrackableDefaultListener.getUpdatedRobotLocation
vuforiaTrackableDefaultListener <vuforia trackable default listener variable>

Description: Return a matrix that gives updated robot location but only if the location has changed since the last call to this function.

<matrix object> ← VuforiaTrackableDefaultListener.getPose
vuforiaTrackableDefaultListener <vuforia trackable default listener variable>

Description: Returns the pose as an OpenGLMatrix of the associated trackable if it is currently visible. If not, null is returned. The pose is the location of the trackable in the phone's coordinate system, i.e. relative to the phone.

VuforiaTrackables

<numeric value> ← VuforiaTrackables.Size
vuforiaTrackables <vuforia trackables variable>

Description: Return the size of specified trackable.

<text value> ← VuforiaTrackables.Name
vuforiaTrackables <vuforia trackables variable>

Description: Return the name of the specified trackable.

<localizer object> ← VuforiaTrackables.Localizer
vuforiaTrackables <vuforia trackables variable>

Description: Return the localizer that manages specified trackable.

<trackable object> ← call VuforiaTrackables.get
vuforiaTrackables <vuforia trackables variable>
index <numeric value>

Description: Return the specific trackable among the set of trackables based on the specified index.

call VuforiaTrackables.setName
vuforiaTrackables <vuforia trackables variable>
name <text value>

Description: Set the name of the specified set of trackables or individual trackable.

call VuforiaTrackables.activate vuforiaTrackables <vuforia trackables variable>

Description: Activate the localizer so that it actively seeks the locations of the trackables associated with the given trackables object.

call VuforiaTrackables.deactivate vuforiaTrackables <vuforia trackables variable>

Description: Deactivate the localizer so that it no longer seeks the locations of the trackables associated with the given trackables object.

Logic

The Logic menu contains blocks that allow you to specify conditions when blocks should be executed and to create logic expressions that produce logic values, i.e. True or False values.

If <logic value>

do <steps>

Description: The series of steps is executed if the logic value is true. The logic value is usually a comparison or other logic equation. Place blocks in the steps sections to specify what you want to be executed when the condition is true.

If <logic value>

do <steps if true>

else <steps if false>

Description: The series of steps in the <steps if true> section is executed if the logic value

is true and those in the <steps if false> section is executed if the logic value is false. The logic value is usually a comparison or other logic equation. Place blocks in the <steps if false> section to specify what you want to be executed when the condition is true. Place blocks in the <steps if false> section to specify what you want to be executed when the condition is false.

If <first logic value>

do <steps if first condition is true>

else if <second logic value>

do <steps if second condition is true>

else <steps if none of the conditions are true>

Description: The series of steps in the <steps if first condition is true> section is executed if the first logic value is true. The series of steps in the <steps if second condition is true> section is executed if the second logic value is true. If neither logic value is true, the steps in the <steps if none of the conditions are true> section is executed. If you would like to add additional conditions to the list of conditions, use the blue icon in the upper left to drag *else if* blocks into the complex block.

<logic value> ← <1st numeric value> [= or < or <= or > or >=] <2nd numeric value>

Description: Return a logic value based on comparison of 1st numeric value to the 2nd numeric value. Either or both values can be variables or formulas. For instance, if you specify *A < 5*, true will be returned if variable A contains a value of 5 or less and false otherwise.

<logic value> ← <1st logic value> [and OR or] <2nd logic value>

Description: Return a logic value based on the logical combination of two other logic values. If "and" is selected, true will be returned if both of <1st logic value> and <2nd logic value> are true and false otherwise. If "or" is selected, true will be returned if <1st logic value> and/or <2nd logic value> are true and false will be returned if both are false.

<logic value> ← not <logic value>

Description: Return true if the logic value to the right of *not* false. Return false if the logic value to the right of *not* is true.

<logic value> ← [true OR false]

Description: Return true if *true* is selected. Return false if *false* is selected.

null

Description: Return the special value of null, which can be tested for to distinguish it vs other values using *call isNull* OR *call isNotNull*.

<numeric value> ← test <logic value>
if true <1st numeric value>
if false <2nd numeric value>

Description: Return 1st numeric value if the logic value is true. Return the 2nd numeric value if the logic value is false.

Loops

repeat <count> times

do <steps>

Description: Repeat a series of steps the number of times indicated by the count value. Place blocks in the steps area to specify what you want to be repeated.

repeat [while OR until] <logic value>
do <steps>

Description: Repeat a series of steps while the specified logic value is true. Place blocks in the steps area to specify what you want to be repeated. When the logic value is no longer true, any blocks after this repeat block will be executed. If "until" is selected instead of "while" the steps will be executed until the logic value becomes true. Thus selecting "until" is equivalent to using while and negating the logic value by inserting a logic "not" before the logic value.

count with <variable> from <1st value> to <last-value> by <increment-value>
do <steps>

Description: Repeat a series of steps a specific number of times equal to

(<last-value>-<1st value>+<increment value>) / <increment value>.

The first time the variable will be set to <1st value>. The second time it will be <1st value> + <increment value>. The variable will continue to increase by increment value until last value is reached. For example, *count with x from 1 to 5 by 1 do <steps>* will execute the steps 5 times with x=1 the first time, 2 the second time and 5 the last time.

for each item <variable> in list <list object>
do <steps>

Description: Repeat a series of steps for each item in a list. The variable is set to the value of the first item in the list the first time the steps are executed, the 2nd item the second time, etc.

[break out OR continue with next iteration] of loop

Description: If "break out" is selected the loop will no longer be executed when this block is encountered. This block will typically be used with an if-do block or an if-do-else block. If "continue with next iteration" is selected the loop will continue be executed by the next block to be executed will be the first block in the loop.

Math

The Math menu contains blocks that allow you to create formulas.

<numeric value> ←<constant>
Description: Return the indicated constant numeric value.

<numeric value> ← <1st numeric value> + <2nd numeric value>
Description: Add the 2nd numeric value to the 1st numeric value.

<numeric value> ← <1st numeric value> - <2nd numeric value>
Description: Subtract the 2nd numeric value from the 1st numeric value.

<numeric value> ← <1st numeric value> x <2nd numeric value>
Description: Multiply the 1st numeric value by the 2nd numeric value

<numeric value> ← <1st numeric value> / <2nd numeric value>
Description: Divide the 1st numeric value by the 2nd numeric value.

<numeric value> ← <1st numeric value> ^ <2nd numeric value>
Description: Raise the 1st numeric value to the power given by the 2nd numeric value

<numeric value> ← - <numeric value>
Description: Negate the numeric value.

<numeric value> ← absolute <numeric value>
Description: If the numeric value is negative, negate it to make it positive.

<numeric value> ← square root <numeric value>
Description: Calculate the square root of the numeric value.

<numeric value> ← [sin OR cos OR tan OR asin OR acos OR atan] <numeric value>
Description: Calculate the trigonometric function of the numeric value.

<numeric value> ← atan2 y <numeric value> x <numeric value>

Description: Returns a numeric value between -180 and 180 degrees, representing the counterclockwise angle between the positive X axis and the point (x, y).

<numeric value> ← [pi OR e OR sigma OR sqrt(2) OR sqrt(1/2) OR infinity]
Description: Produce the indicated numeric constant.

<logic value> ← <numeric value> is [even OR odd OR prime OR whole OR positive OR negative OR divisible by]
Description: Return true if the numeric value satisfies the indicated condition, false otherwise.

<numeric value> ← [round OR round up OR round down] <numeric value>
Description: Round the numeric value in the way indicated.

<numeric value> ← [sum OR min OR max OR average OR median OR modes OR standard deviation OR random item] of list <list object>
Description: Calculate the indicated value from the values in the specified list.

<numeric value> ← remainder of <1st numeric value> / <2nd numeric value>Description:
Determine the remainder resulting from the division of the 1st numeric value by 2nd.

<numeric value> ← constraint <1st numeric value> low <2nd numeric value> high <3rd numeric value>
Description: Compare the 1st numeric value to the 2nd and 3rd. If the 1st is less than the 2nd, return the 2nd. If the 1st is greater than the third, return the 3rd. Otherwise return the 1st.

<numeric value> ← random integer from <1st numeric value> to <2nd numeric value>
Description: Produce a random integer between the 1st and 2nd numeric values, inclusive.

<numeric value> ← random fraction

Description: Produce a random fraction between 0 and 1.

Text

The Text menu provides way of creating text strings referred to as text values. It also provides blocks that allow you to access characteristics of text values.

<text value> ← <text constant>
Description: Return the indicated text constant.

<text value> ← create text with <text value>
<text value>
Description: Return a text value which is the concatenation of two or more text values.

to Description: text variable> append text <text value>
Description: Concatenate the text value to the text held by the text variable.

<numeric value> ← length of <text value>
Description: Return the length of the text value.

<logic value> ← <text value> is empty
Description: Return true if the text value is empty, false otherwise.

<numeric value> ← in text <text variable> find [first OR last] occurrence of <text value>
Description: Return the index of the first or last occurrence of the text value in the text variable. Return -1 if the text value is not found in the text variable.

<text value> ← in text <text variable> get [letter # OR letter # from end
OR first letter OR last letter OR random letter]
Description: Return the letter in the text variable described by the part after "get". If a letter number is provided and it is 0, the last letter is returned.

<text value> ← in text <text variable> get substring
from [letter # OR letter # from end OR first letter]] <numeric value>
to [letter # OR letter # from end OR first letter] <numeric value>
Description: Return the substring of the text variable described between "from" and "to" values.

Lists

<list object> ← create empty list
Description: Return a new list object which is empty.

<list object> ← create list with <item>
<item>
<item>
Description: Return a new list object containing a series of items. Use the blue settings icon to adjust the number of items in the list. Items can be numeric values, text values or any object.

<list object> ← create list with item <item> repeated <numeric value> times

Description: Return a new list object with the item repeated the indicated number of times.

<numeric value> ← length of <list object>
Description: Return the number of items in the list object.

<logic value> ← <list> is empty
Description: Return true if the list is empty, false otherwise.

<numeric value> ← in list <list variable> find [first OR last] occurrence of item <item>
Description: Return the list index of the item in the list variable. Return -1 if the item is not in the list.

<item> ← in list <list variable> [get OR get and remove OR remove]
[# OR # from end OR first OR last OR random] <numeric value>
Description: Return the item from the list variable in the indicated position in the list. The first item has index 1. If get and remove is specified, the item will be deleted from the list. If remove is specified, the item is removed and nothing is returned.

in list <list variable> [set OR insert at]
[# OR # from end OR first OR last OR random] as <item>
Description: Set a particular list item in the list variable in the indicated position using the given item. If "insert at", insert the item in the indicated position, moving the items after it down one slot.

<list object> ← in list <list variable> get sub-list from
[# OR # from end OR first] <> to [# OR # from end OR last]
Description: Return a list object that contains the sub-list created by taking the series of list items from the list variable starting at the index given after "from" and ending at the index given after "to".

<list object> ← make [list from text <text value> OR text from list <list object>]
with delimiter <text character>
Description: If "list from text" is selected, return a list object that contains a series of text items created by dividing the provided text value using the text character as the delimiter indicating the beginning and end of each item. For example, *make list "abc,d,ef,ghi" with delimeter ","* returns a 4-item list made up of "abc", "d", "ef" and "ghi".

If "text from list" is selected, return a text object that contains the items in the provided list object formatted as a series of text values separated by the indicated delimiter. For example, if the list object contains "j", "kl", "mno" and "p" and the delimiter is " " the text value "j kl mno p" is returned.

<list object> ← sort [numeric OR alphabetic OR alphabetic, ignore case]
[ascending OR descending] <list object>
Description: Return a list object by sorting the items in the provided list object. Based on the words after "sort" sort the list numerically, alphabetically using case, or alphabetically ignoring case and either ascending or descending.

Variables

Blocks is based on a technology called Blockly. Here's what the Blockly Wiki has to say about variables:

We use the term variable the same as it is used in mathematics and in other programming languages: a named value that can be changed (varies). Variables can be created in several different ways.

- Every count with and for each block uses a variable and defines its values. These values can only be used within the block. A traditional computer science term for these are loop variables.
- User-defined functions (also known as "procedures") can define inputs, which creates variables that can be used only within the function. These are traditionally called "parameters" or "arguments".
- Users may create variables at any time through the "set" block. These are traditionally called "global variables". Blockly does not support local variables.

The following items are found in the Variables sub menu. Once a variable is created, there are additional options like Rename available by clicking on the drop-down triangle in a variable block.

Create variable
Description: Create a new variable. A pop-up will allow you to give the variable a name.

set <variable> to <value or object>
Description: Set a variable to a specified value or object.

change <variable> by <numeric value>
Description: Add the numeric value to the variable. This function assumes the variable contains a numeric value.

<value or object> ← <variable>
Description: Return the value or object contained in the variable.

Functions

do <do something>
<steps>
Description: Create a function by replacing “do something” with the name of the function. Drag blocks into the function to create the steps that will be executed when the function is executed. To create parameters, use the blue icon in the upper left corner. Give each parameter a name.

do <do something>
<steps>
return <value>
Description: Create a function that returns a value or an object. Replace “do something” with the name of the function. Drag blocks into the function to create the steps that will be executed when the function is executed. To create parameters, use the blue icon in the upper left corner. Give each parameter a name.

if <logic formula> return <value or object>
Description: if logic formula is true the function should return the given value or object

<function name>
Description: Run the indicated function.

<function name> with <parameter value>
Description: Run the indicated function with one or more parameters.

runOpMode

Description: Execution of this block would run the current op mode but this would be a case of recursion – the op mode calling itself and that is not recommended.

Miscellaneous

<comment>
Description: Allows you to describe your program so you and others can better understand. Does change what the program does as comments are not executied.

Null
Description: Return the special value of null, which can be tested for to distinguish it versus other values.

<logic value> ← call [isNull OR isNotNull]
value <value>
Description: If “isNull” is selected, return true if the value is null and false otherwise. If “isNotNull” is selected return true if the value is not null and false if it null.