Talon SRX Motion Profile Reference Manual

Revision 1.0

Cross The Road Electronics

www.crosstheroadelectronics.com

Table of Contents

TO OUR VALUED CUSTOMERS

It is our intention to provide our valued customers with the best documentation possible to ensure successful use of your CTRE products. To this end, we will continue to improve our publications, examples, and support to better suit your needs.

If you have any questions or comments regarding this document, or any CTRE product, please contact support@crosstheroadelectronics.com

To obtain the most recent version of this document, please visit www.ctr-electronics.com.

1. What is the Motion Profile Control Mode?

The Talon SRX supports a number of control modes…

- Percent Voltage
- Voltage Compensation
- Position Closed-Loop
- Velocity (Speed) Closed-Loop
- Current (Draw) Closed-Loop

These modes are documented in the Talon SRX Software Reference Manual and allow a "Robot Controller" to specify/select a target value to meet. The target can simply be the percent output motor drive, or a target current-draw. When used with a feedback sensor, the robot controller may also simply set the target position, or velocity to servo/maintain.

However, for advanced motion profiling, the Talon SRX additionally supports a mode whereby the robot controller can *stream* a sequence of trajectory points to express an entire *motion profile*.

Each trajectory point holds the desired velocity, position, and time duration to honor said point until moving on to the next point. These points can be sent to the Talon before executing the motion profile ensuring that enough points are buffered for smooth transitions.

Alternatively, the trajectory points can be streamed into the Talon *as the Talon is executing the profile*, so long as the robot controller sends the trajectory points faster than the Talon consumes them. This also means that there is no practical limit to how long a profile can be.

1.1. What is the benefit?

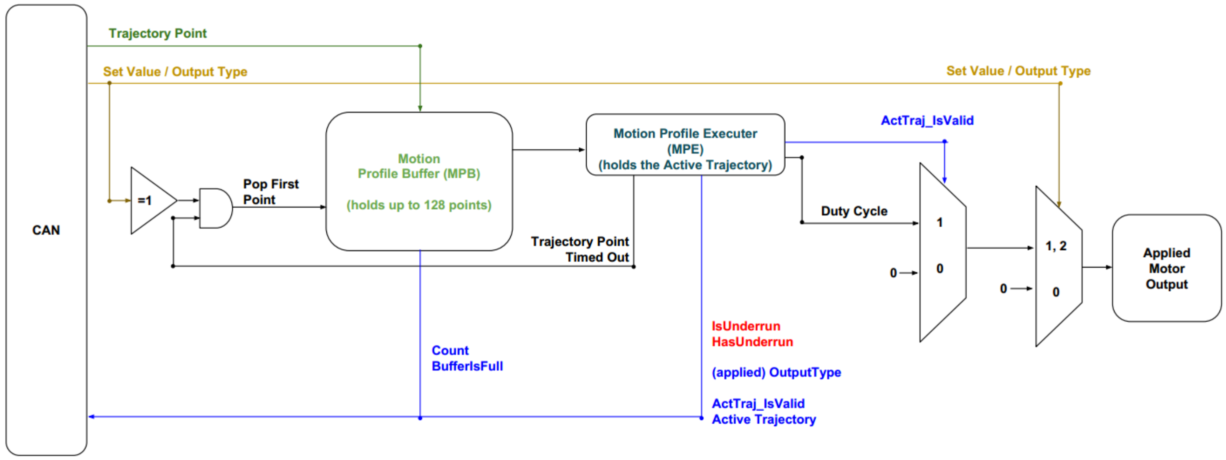
Leveraging the Motion Profile Control Mode in the Talon SRX has the following benefits…

- Direct control of the mechanism *throughout* the entire motion (as opposed to a single PID closed-loop which directly servos to the end target position).
- Accurate scheduling of the trajectory points that is not affected by the performance of the primary robot controller.
- Improved repeatability despite changes in battery voltage.
- Improved repeatability despite changes in motor load.
- Provides a method to synchronously gain schedule.

Additionally, this mode could be used to schedule several position servos in advance with precise time outs. For example, one could map out a collection of positions and timeouts, then stream the array to the Talon SRX to execute them.

2. Functional Diagram

Below is a simplified functional diagram of the motion profile firmware in the Talon SRX.



3. Signal List and Terms

3.1. Motion Profile Buffer (MPB)

The firmware buffer in the Talon SRX. The MPB can hold up to 128 trajectory points at once (with the current firmware, see Section 9.2 for requirements).

3.2. Motion Profile Executer (MPE)

The Motion Profile Executer is the firmware that calculates the motor-output based on the active trajectory point. It updates the motor update every 1ms by summing the velocity feed-forward term with the output of a position servo. The MPE holds one trajectory point at a time. Should the Talon Motion Control features be activated when there is no trajectory point buffered into the MPE, the motor output will be neutral.

3.3. Trajectory Point

This term refers to the trajectory point that is being sent to the Talon SRX buffering. This is not the point that is currently be used for calculating the throttle output.

3.3.1. The Last Trajectory Point

When generating the trajectory points, the final trajectory point in the profile is referred to as the "Last Trajectory Point". This point typically will have the following properties…

- The velocity to feed-forward is set to zero.
- The "Is Last" flag is set to "true".

This ensures that when the MPE processes the last point, it will hold position indefinitely (or until the robot controller decides the next course of action).

3.3.2. Trajectory Point Duration (Milliseconds)

The time to servo this trajectory point in milliseconds. The minimum value is 1ms and the maximum is 255ms. If '0' is sent, it will be interpreted as 1ms.

3.3.3. Trajectory Point Velocity
This is the velocity to feed-forward when this trajectory point is loaded into the MPE.

3.3.4. Trajectory Point Position
This is the position to target for the PID closed loop portion of the MPE inner loop when this trajectory point is loaded into the MPE.

3.3.5. Trajectory Point Profile Slot Select
Selects which slot to pull the closed-loop gains when this trajectory point is processed by the MPE. Talon persistently stores two sets of closed-loop parameters. This allows for atomic switching between two unique sets of gain-constants, ramping, and I-Zone.

3.3.6. Trajectory Point Is Last
Set to “true” if trajectory point is the final point of the motion profile. This signals the MPE to continue to servo this point even after it’s time duration expires. Be sure to zero the position of this trajectory point so that MPE will closed-loop to the final intended position.

3.3.7. Trajectory Point Vel (Velocity) Only
If a trajectory point has this flag set, then the target position is ignored, and only the velocity term is used. This can be used to conveniently “turn off” the position PID portion.

For example, a sequence of points that are Velocity-only provide a way to stream a custom-throttle-waveform, without the need of a sensor.

Additionally, turning off the position PID port can be diagnostically useful in seeing how well the system responds with just feed-forward. Note that if the “last” point is vel-only, then it will not servo position.

3.3.8. Trajectory Point Zero Position
If a trajectory point has this flag set, the MPE will zero the position value of the selected sensor when this trajectory point is processed. Typically, this would be done only with the first trajectory point of the motion profile. The goal is to clear the position so that the position values of the following trajectory points are relative to the start position of the mechanism.

The generated trajectory points could be calculated to not require re-zeroing the sensor. This flag merely provides the option to express a profile relative to the current physical position of the mechanism.

3.4. Motion Profile Set Value (Set Output, or Output Type)

The set-value is interpreted as follows:

0 - Motor output is neutral.

1 - Motor output is driven by the MPE. MPE will pop the "first" point from the MPB. If the MPE does not have a point (is empty) Motor output is neutral and "Is Underrun" is set.

2 - Motor output is in "hold". The active trajectory point inside the MPE will be driven indefinitely (while set-value is '2' and control mode is "Motion Profile"). This is useful when a profile has finished and the user needs to disconnect the MPE from the MPB to begin buffering the next action, meanwhile the MPE will continue to servo/drive the loaded point.

This set-value is typically only useful when the "Last" trajectory point has been reached, which typically has a target velocity of '0', and simply will servo/maintain the final position.

3.5. Active Trajectory

The active trajectory is the trajectory point loaded into the MPE. This holds the velocity, position, and flags used by the MPE to calculated the motor output.

3.5.1. Active Trajectory Is Valid

Since it is possible for the MPE to be "empty", there must be a flag to instrument this. For example, if the robot controller sends a nonzero Motion Profile Set Value when there are no trajectory points buffered, this will cause the MPE to be active when no trajectory point is available to shift into the MPE. When this happens, this flag will be false, and motor output will be neutral.

When reading the other member signals of Active Trajectory, this flag should be checked first.

3.5.2. Active Trajectory Velocity

The velocity the MPE will feed-forward if activated.
Only valid if "Active Trajectory Is Valid" is **true**.

3.5.3. Active Trajectory Position

The position the MPE will target if activated.
Only valid if "Active Trajectory Is Valid" is **true**.

3.5.4. Active Trajectory Profile Slot Select

The selected slot that the MPE will pull closed-loop parameters from if activated.
Only valid if "Active Trajectory Is Valid" is **true**.

3.5.5. Active Trajectory Is Last

The "Is Last" signal of the active trajectory point currently in the MPE. MPE will continue to servo this point indefinitely, allowing robot application to prepare next profile or change modes. Only valid if "Active Trajectory Is Valid" is **true**.

3.5.4. Active Trajectory Vel (Velocity) Only

The "Vel Only" signal of the active trajectory point currently in the MPE.
Only valid if "Active Trajectory Is Valid" is **true**.

3.5.5. Active Trajectory Zero Position

The "Zero Pos" signal of the active trajectory point currently in the MPE.
Only valid if "Active Trajectory Is Valid" is **true**.

3.6. Is Underrun

If the MPE is ready to process a new active trajectory point, but one is not available, this flag is set. The underrun behavior of the Talon depends on which three situations caused the underrun…

- Robot application has signaled MPE to start, but there is no buffered trajectory point to start with. Motor-output is neutral until a point is available.
- Robot application has signaled MPE to hold, but there is no active trajectory point in the MPE. Motor-output is neutral until a point is available.
- During the execution of a profile, the MPE timed out the active trajectory point, and was ready for the next point, but one was not available (MPB was empty). When this happens MPE will continue to use the active trajectory point until a new point is available in the buffer. In other words, MPE will not release the active trajectory point if the next point is not available.

All three situations are caused by the robot controller not providing trajectory points to keep pace with the execution, or enabling the MPE before enough trajectory points are buffered.

This flag is automatically cleared when the problem condition is resolved.

3.7. Has Underrun

When Is Underrun is set, Has Underrun is also set. However, this flag only is cleared by the robot application using the robot API. This ensures the robot application can poll for underrun behavior infrequently without risking missing intermittent buffer underruns.

3.8. (Bottom, Firmware-level) Buffer Count

The number of trajectory points buffered in the MPB.

3.9. (Bottom, Firmware-level) Buffer Is Full

Flag indicating the firmware trajectory buffer is MPB.

3.10. Feed-Forward Gain

When used in Motion Control Mode, this value is used to feed-forward the target velocity gain. In other words, this is the K_V value in the Motion Profile inner loop equation.

3.11. Proportional Gain

When used in Motion Control Mode, this value is used as the proportional gain for the position closed-loop portion of the Motion Profile inner loop equation.

3.12. Integral Gain

When used in Motion Control Mode, this value is used as the integral gain for the position closed-loop portion of the Motion Profile inner loop equation.

3.13. Derivative Gain

When used in Motion Control Mode, this value is used as the derivative gain for the position closed-loop portion of the Motion Profile inner loop equation.

3.14. (Top, API-level) Buffer Count

The language-based robot APIs (C++, Java, HERO C#) utilize a “top level” buffer to hold trajectory points as they funnel into the Talon SRX’s MPB over CAN Bus. This helps simplify the robot application by allowing the program to one-shot generate the entire motion, buffer it at once into the robot API, and resume to other tasks while the Talon’s MPB fills.

4. Theory of Operation

The intent of the motion control features of Talon is allow the robot controller to generate and stream the trajectory points into the Talon SRX while the Talon honors the currently-selected control mode. This means that the Talon SRX can be in any control-mode (PercentVoltage for example) while motion profile buffering occurs. When the robot controller has detected that enough of the trajectory points have been funneled into the Talon, the robot controller can then select the motion profile control mode, and enable the MPE by setting a ‘1’,

When the motion profile is complete, the robot controller can disconnect the firmware trajectory buffer by setting a ‘0’ or ‘2’. A ‘0’ will signal the Talon to neutral the motor-output. A ‘2’ will signal the Talon to “hold” the current trajectory point, however this should only be done if the active trajectory point has a zero feed-forward velocity (last point).

The inner loop of the motion profile executer can be simplified as…

```
MotorOutput = PositionClosedLoop(targetPosition) + ( Kv ) x ( targetVelocity )
```

…where `targetPosition` and `targetVelocity` are measured in Talon native units, and the motor output ranges from -1023 (full reverse) to +1023 (full forward). The feedforward gain is used for the kV constant.

Although the math inside the firmware uses Talon native units, the robot API uses floating point rotations for position, and RPM for velocity. This is explained in Section 17.2 in the Talon SRX Software Reference Manual.

The I-Zone, nominal/peak closed-loop outputs, ramping, and allowable closed-loop error signals are all in-effect. More information on how these signals impact the motor-response can be found in the Talon SRX Software Reference Manual.

5. New Motion Profile API

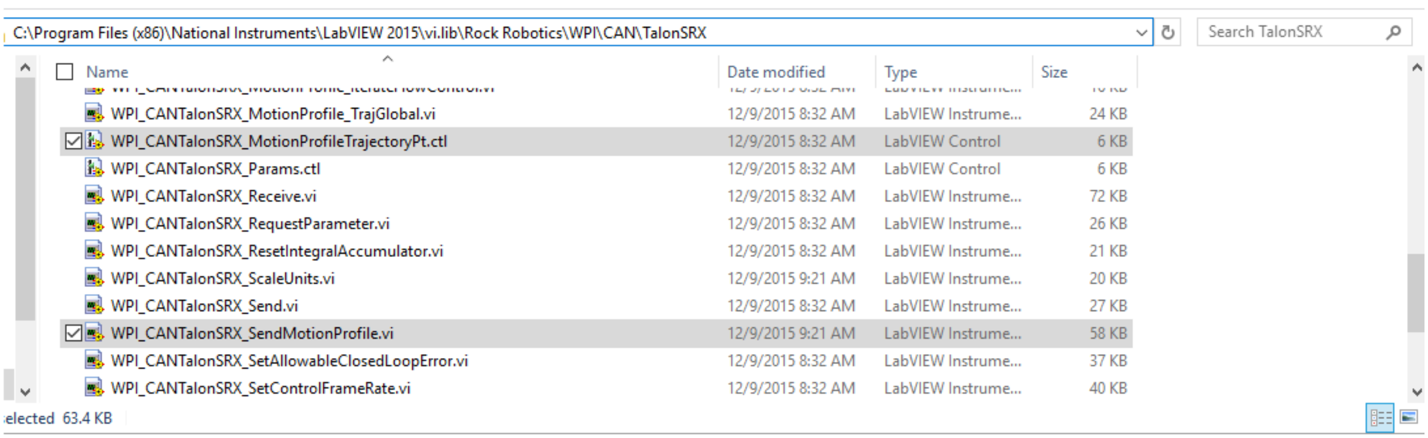
5.1. New Motion Profile API – LabVIEW

The only additional VIs necessary for Motion Profile are…

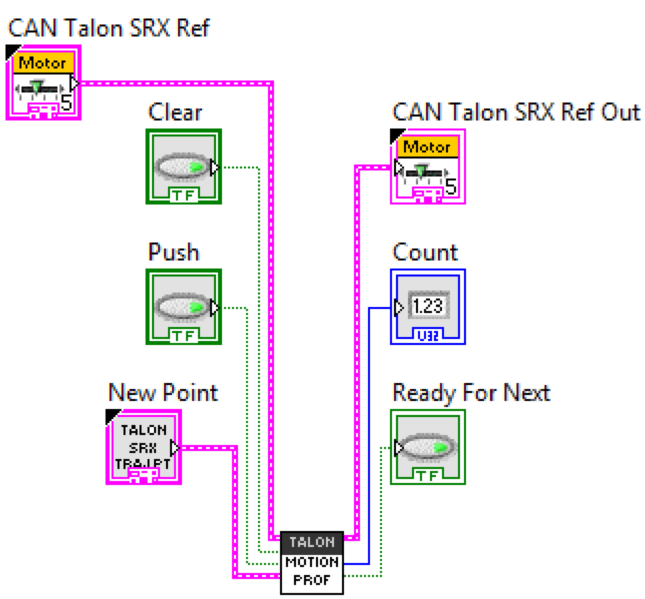
WPI_CANTalonSRX_SendMotionProfile.vi

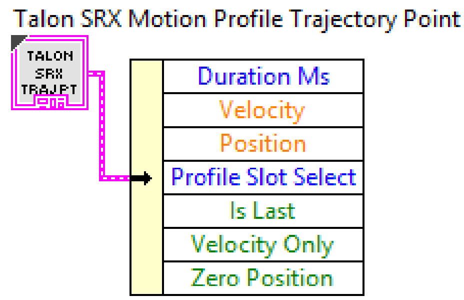
Talon SRX Motion Profile Trajectory Point

These can be used by drag-dropping the VI and CTL file into the LabVIEW block diagram view.



Breakout of signals of the new VI and CTL.

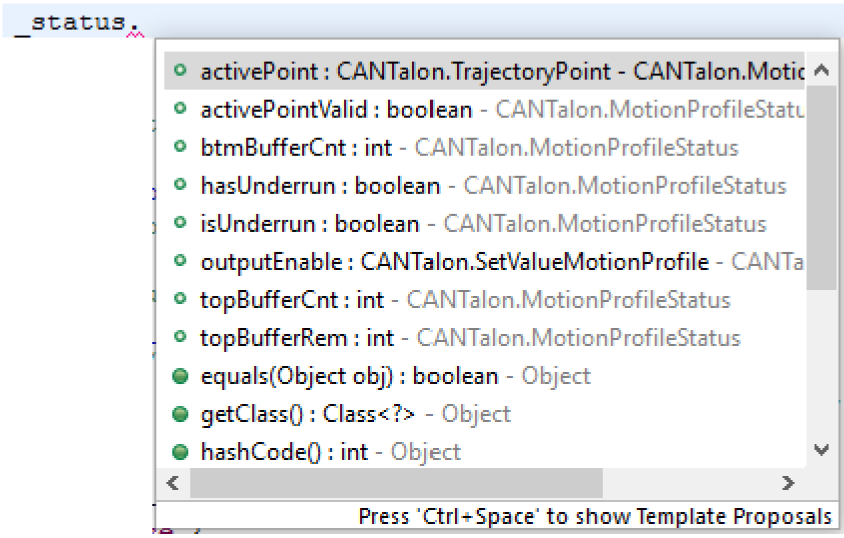




5.2. New Motion Profile API – Java

Motion Profile Status (MPB and MPE status) can be polled with this function.

```java
/* Get the motion profile status every loop */
_talon.getMotionProfileStatus(_status);
```



The hasUnderrun flag can be polled and cleared using these functions. Replace the instrumentation line with your application's handler or print statement.

```java
/* did we get an underrun condition since last time we checked ? */
if (_status.hasUnderrun) {
    /* better log it so we know about it */
    instrumentation.OnUnderrun();
    /*
     * clear the error. This flag does not auto clear, this way
     * we never miss logging it.
     */
    _talon.clearMotionProfileHasUnderrun();
}
```

Motion profile trajectory points can be cleared and pushed with these functions….

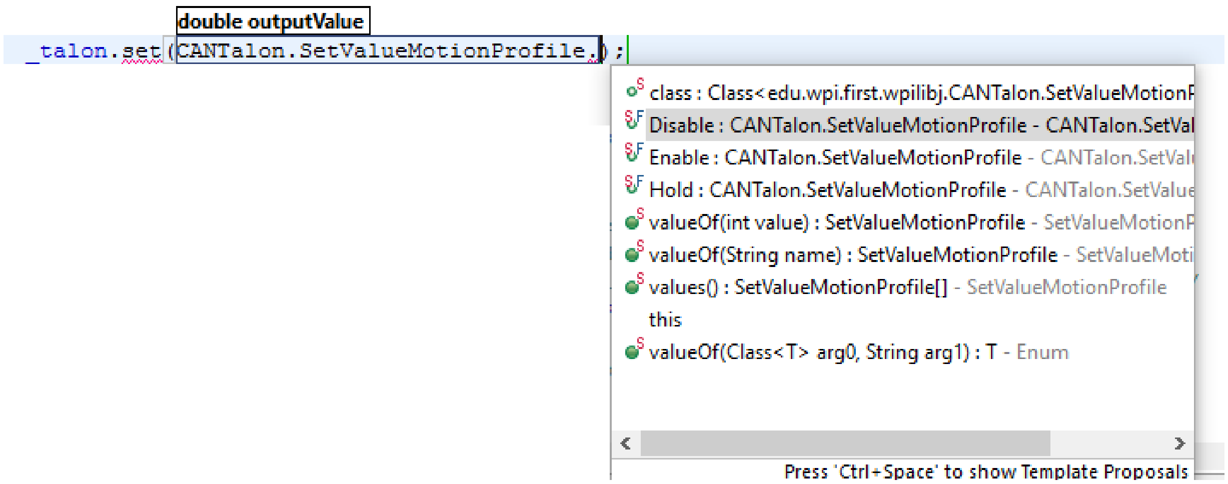
```java
_talon.pushMotionProfileTrajectory(point);

_talon.clearMotionProfileTrajectories();
```

Process function should be called periodically at a rate faster than the profile execution.

```java
_talon.processMotionProfileBuffer();
```

Controlling the Motion Profile Executer can be done by first entering the control mode, then using the set method

```
_talon.changeControlMode(TalonControlMode.MotionProfile);
```

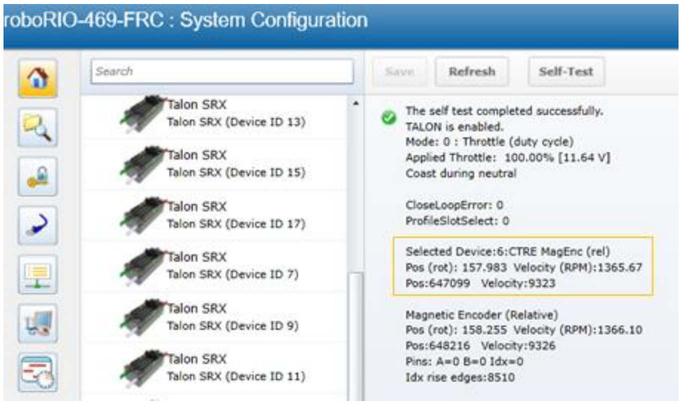


6. Software Integration Steps

This section describes the necessary pieces for leveraging the motion profile control mode of the Talon SRX. The sections below reference code segments from the LabVIEW and Java code examples, which are available for download (see Section 7). Although C++ is not referenced directly, the C++ example is line-for-line comparable to the Java example, therefore C++ users can still follow through the sections to understand the integration requirements, and to learn how the C++ Motion Profile Example works.

6.1. Direct Drive the Talon SRX and Check Sensor

The first step is to ensure that the Talon's sensor is functional and is in-phase with motor. The simplest method to do this is to directly control the motor-output of the Talon in PercentVoltage (or similar) mode. Select the sensor programmatically and drive the Talon with positive motor output (green Talon LEDs) while measuring the sensor position and velocity. If the Talon's selected sensor position is **not moving in a positive direction, then use the robot API to reverse the sensor output** (see section 7.4 in the Talon SRX software reference manual).



While driving the mechanism ensure…

- The position increases with positive throttle and…
- That there are no strange or erroneous samples. Remember, a motion profile executer is only as good as the sensor!
- Then measure the speed of the sensor at a given output (100% or approximately the max motor-output expected to use).

In this example, we see at 100% throttle, the sensor's measured velocity is **1365 RPM** or **9323** native units per 100ms.

The referenced C++, Java, and LabVIEW examples (available on GitHub) all provide a method to directly drive the Talon for the purpose of checking sensor direction and sampling the velocity and throttle.

6.1.1. Direct Drive the Talon SRX and Check Sensor - Java

In the Java example, we accomplish this by entering Voltage Control Mode when button5 is let go (motion profile is not activated). Typically, PercentVoltage Control Mode should be used, however this example is also meant to demonstrate one of the new feature-additions for 2016 (Voltage Compensation mode).

```
/**  function is called periodically during operator control */
public void teleopPeriodic() {
    /* get buttons */
    boolean [] btns= new boolean [_btnsLast.length];
    for(int i=1;i<_btnsLast.length;++i)
        btns[i] = _joy.getRawButton(i);

    /* get the left joystick axis on Logitech Gampead */
    double leftYjoystick = -1 * _joy.getY(); /* multiple by -1 so joystick forward is positive */

    /* call this periodically, and catch the output.  Only apply it if user wants to run MP. */
    _example.control();

    if (btns[5] == false) { /* Check button 5 (top left shoulder on the logitech gamepad). */
        /*
         * If it's not being pressed, just do a simple drive.  This
         * could be a RobotDrive class or custom drivetrain logic.
         * The point is we want the switch in and out of MP Control mode.*/

        /* button5 is off so straight drive */
        _talon.changeControlMode(TalonControlMode.Voltage);
        _talon.set(12.0 * leftYjoystick);
```

By using the left gamepad Y-axis, we can drive the mechanism while measuring the velocity.

6.2. Measure Peak RPM to Calculate F-gain

Using the values measured in Section 6.1, we can calculate our F-gain.
The example measurement is at 100% throttle, the sensor's measured velocity is **1365 RPM** or **9323** native units per 100ms.

To sanity check this, knowing the resolution of our Magnetic Encoder is 4096 (Section 17.2.1 in Talon Software Reference Manual) we could have also calculated the velocity in native units per 100ms.

Velocity is measured in change in native units per $T_{velMeas}$= 100ms.

(1365.67 Rotations / min) X (1 min / 60 sec) X (1 sec / 10 $T_{velMeas}$) X (4096 native units / rotation) = **9323** native units per 100ms

Now let's calculate a Feed-forward gain so that 100% motor output is calculated when the requested speed is **9323** native units per 100ms.

F-gain = (100% X 1023) / 9323
F-gain = 0.109728

Let's check our math, if the target speed is 9326 native units per 100ms, Closed-loop output will be (**0.109728** X **9323**) => 1023 (full forward).

Now by applying this F-gain, our Talon can perform the velocity feed-forward portion of the motion profile inner-loop correctly.

```
_talon.setF(0.1097);
_talon.setP(0);
_talon.setI(0);
_talon.setD(0);
```

Next we will set the calculated gain. This can also be done in the roboRIO web-based configuration or programmatically (example here is for Java). See section 12.1 in Talon SRX Software reference manual for how to programmatically set gains in all languages.

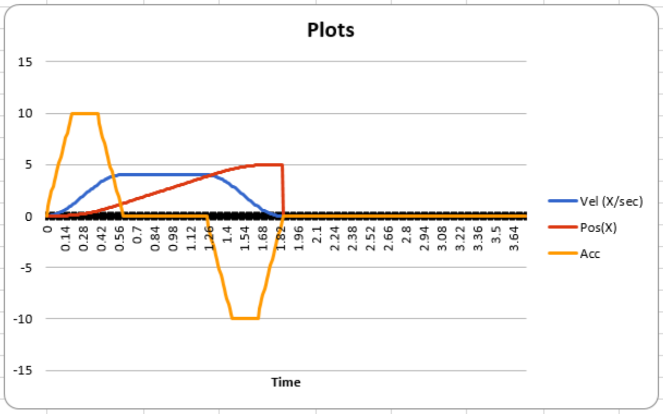
6.3. Generating the trajectory points

Trajectory points can be generated using a number of techniques (trapezoidal, s-curve, etc.). An excel sheet is provided to perform this generation to get started. "Motion Profile Generator.xlsx" can be downloaded at…
http://www.ctr-electronics.com/talon-srx.html#product_tabs_technical_resources
…and is available at our GitHub account.

Of course many users will choose to utilize their own motion profile generators, which is acceptable as the trajectory point requirements are meant to be generic.

Opening this Excel file, we see the following view in the first sheet.

Vprog(max speed)	4.00	rotations/sec
Dist	5	rotations
T1	400	ms
T2	200	ms
itp	10	ms
T4	1250	ms
FL1	40	
FL2	20	
N	125	



Step	Time	Input	Filter 1 sum	Filter 2 Sum	1 if zero pt	Output Included	Output Vel (X/sec)	Output Pos(X)	Output Acc	Output Is First	Output Is Last
1	0	0	0	0	1	TRUE	0	0	0	TRUE	0
2	0.01	1	0.025	0.025	0	TRUE	0.00952381	4.76E-05	0.952381	FALSE	FALSE
3	0.02	1	0.05	0.075	0	TRUE	0.02380952	0.000214	1.428571	FALSE	FALSE
4	0.03	1	0.075	0.15	0	TRUE	0.04285714	0.000548	1.904762	FALSE	FALSE
5	0.04	1	0.1	0.25	0	TRUE	0.06666667	0.001095	2.380952	FALSE	FALSE
6	0.05	1	0.125	0.375	0	TRUE	0.0952381	0.001905	2.857143	FALSE	FALSE
7	0.06	1	0.15	0.525	0	TRUE	0.12857143	0.003024	3.333333	FALSE	FALSE
8	0.07	1	0.175	0.7	0	TRUE	0.16666667	0.0045	3.809524	FALSE	FALSE
9	0.08	1	0.2	0.9	0	TRUE	0.20952381	0.006381	4.285714	FALSE	FALSE

The parameters (green cells in file) to configure are…

- **Vprog**: This is the maximum target velocity in rotations per second. (Note if your desired max speed is in RPM, you must multiply by 60).

- **Dist**: The final target position to servo to in rotations.

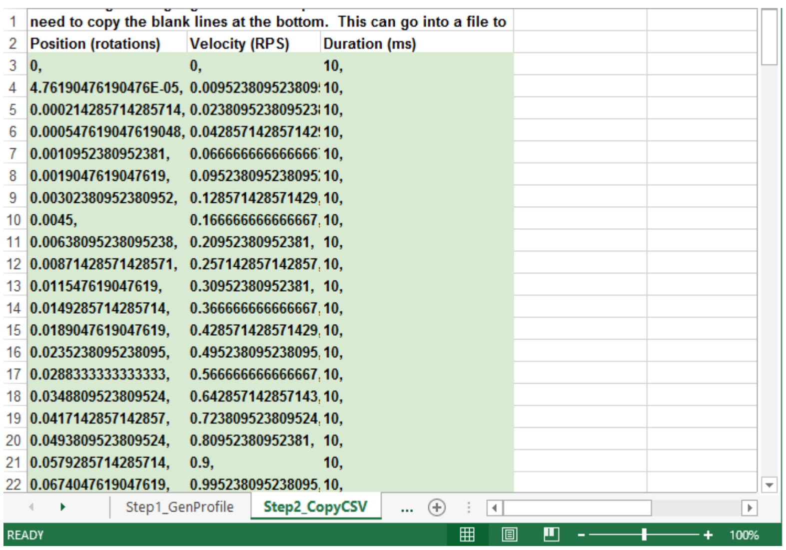
- **T1** and **T2**: This are the acceleration time constants. By tweaking T1, you can control how much of a ramp-up there is until reaching the peak velocity. By tweaking T2, you can control how much rounding there is during the transition between the ramp and the peak velocity. Watch the blue velocity curve and observe how it changes a T1 and T2 are modified.

- **Itp**: The duration of each trajectory point. Default is 10ms per point. This effectively determines how resolute each trajectory point is. Regardless of this value however, the Talon will perform the motion profile inner loop every 1ms.

When the curve seems reasonable, the generated trajectory points are serialized a number of ways.

TIP: if this profile is for drivetrain and you know what the max acceleration is before wheel-slip, you can tweak T1 and T2 until the values under the acceleration column are below your max acceleration.

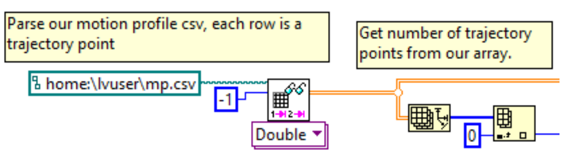
6.3.1. Using a CSV File (for LabVIEW)

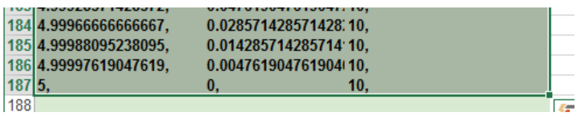


1	need to copy the blank lines at the bottom. This can go into a file to		
2	Position (rotations)	Velocity (RPS)	Duration (ms)
3	0,	0,	10,
4	4.76190476190476E-05,	0.009523809523809	10,
5	0.000214285714285714,	0.023809523809523	10,
6	0.000547619047619048,	0.042857142857142	10,
7	0.0010952380952381,	0.066666666666666	10,
8	0.0019047619047619,	0.095238095238095	10,
9	0.00302380952380952,	0.128571428571429	10,
10	0.0045,	0.166666666666667	10,
11	0.00638095238095238,	0.20952380952381,	10,
12	0.00871428571428571,	0.257142857142857	10,
13	0.011547619047619,	0.30952380952381,	10,
14	0.0149285714285714,	0.366666666666667	10,
15	0.0189047619047619,	0.428571428571429	10,
16	0.0235238095238095,	0.495238095238095	10,
17	0.0288333333333333,	0.566666666666667	10,
18	0.0348809523809524,	0.642857142857143	10,
19	0.0417142857142857,	0.723809523809524	10,
20	0.0493809523809524,	0.80952380952381,	10,
21	0.0579285714285714,	0.9,	10,
22	0.0674047619047619,	0.995238095238095	10,

The CSV tab can be used to copy the values into a CSV file, which could then be placed into a filesystem if the robot controller supports it.

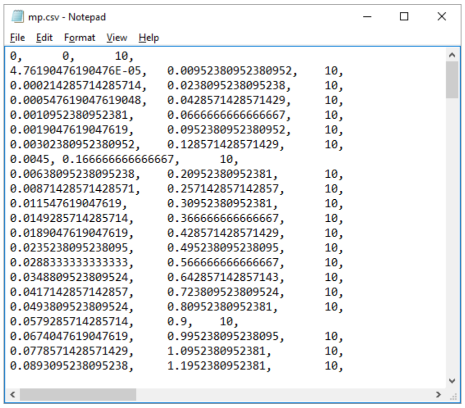
The roboRIO, for example, could read the CSV from its filesystem easily using the LabVIEW





184	4.99966666666667,	0.028571428571428	10,
185	4.99988095238095,	0.014285714285714	10,
186	4.99997619047619,	0.004761904761904	10,
187	5,	0,	10,
188			

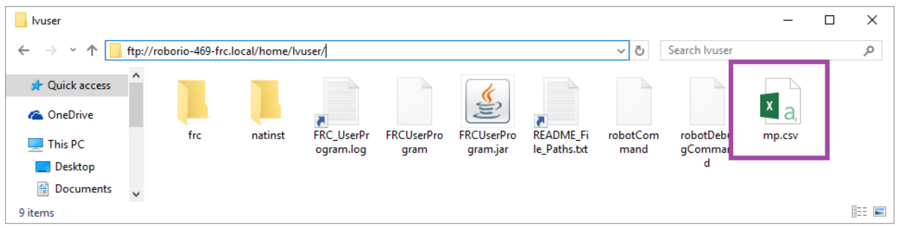
Only copy to the final point, avoid copying blank lines.


mp.csv - Notepad
File Edit Format View Help

```
0,	0,	10,
4.76190476190476E-05,	0.00952380952380952,	10,
0.000214285714285714,	0.0238095238095238,	10,
0.000547619047619048,	0.0428571428571429,	10,
0.0010952380952381,	0.0666666666666667,	10,
0.0019047619047619,	0.0952380952380952,	10,
0.00302380952380952,	0.128571428571429,	10,
0.0045,	0.166666666666667,	10,
0.00638095238095238,	0.20952380952381,	10,
0.00871428571428571,	0.257142857142857,	10,
0.011547619047619,	0.30952380952381,	10,
0.0149285714285714,	0.366666666666667,	10,
0.0189047619047619,	0.428571428571429,	10,
0.0235238095238095,	0.495238095238095,	10,
0.0288333333333333,	0.566666666666667,	10,
0.0348809523809524,	0.642857142857143,	10,
0.0417142857142857,	0.723809523809524,	10,
0.0493809523809524,	0.80952380952381,	10,
0.0579285714285714,	0.9,	10,
0.0674047619047619,	0.995238095238095,	10,
0.0778571428571429,	1.0952380952381,	10,
0.0893095238095238,	1.1952380952381,	10,
```

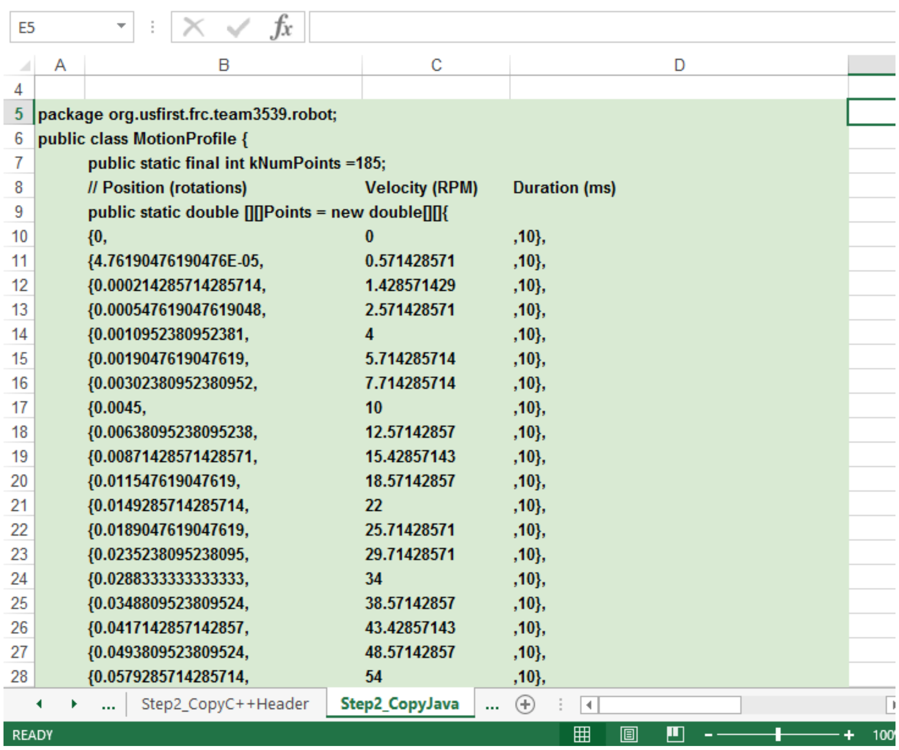
The cells can then be pasted into a simple text file and saved as a csv.

Access over FTP can be done with Windows Explorer, or your favorite FTP client.



6.3.2. Using an array in a script language. (C++, Java, HERO C#, etc.).

The supplemental sheets will create a double-precision array (N X 3) where each row represents a trajectory point, for a total of N trajectory points. For each row, the first cell is the position in rotations, the second cell is the velocity in RPM, and the third parameter is durationMs (though this could be optimized out as this is generally constant).



	A	B	C	D
4				
5	package org.usfirst.frc.team3539.robot;			
6	public class MotionProfile {			
7		public static final int kNumPoints =185;		
8		// Position (rotations)	Velocity (RPM)	Duration (ms)
9		public static double [][]Points = new double[][]{		
10		{0,	0	,10},
11		{4.76190476190476E-05,	0.571428571	,10},
12		{0.000214285714285714,	1.428571429	,10},
13		{0.000547619047619048,	2.571428571	,10},
14		{0.0010952380952381,	4	,10},
15		{0.0019047619047619,	5.714285714	,10},
16		{0.00302380952380952,	7.714285714	,10},
17		{0.0045,	10	,10},
18		{0.00638095238095238,	12.57142857	,10},
19		{0.00871428571428571,	15.42857143	,10},
20		{0.011547619047619,	18.57142857	,10},
21		{0.0149285714285714,	22	,10},
22		{0.0189047619047619,	25.71428571	,10},
23		{0.0235238095238095,	29.71428571	,10},
24		{0.0288333333333333,	34	,10},
25		{0.0348809523809524,	38.57142857	,10},
26		{0.0417142857142857,	43.42857143	,10},
27		{0.0493809523809524,	48.57142857	,10},
28		{0.0579285714285714,	54	,10},

Here is an example of the “CopyJava” tab, which produces a Java-style array which can be copied into an FRC Java application.

6.4. Sending the trajectory points

The robot API includes functions to clear and push trajectory points into the Talon. The status can be polled periodically to determine if enough trajectory points have been buffered to start the motion profile.

If the motion profile is large or if the motion profile needs to start quickly (before buffering fills Talon completely), the application should set the process periods to keep pace with the rate of the motion profile. In other words, if your profile has trajectory points that have 10ms durations, then the application task that processes the profile should process at least as fast. A conservative recommendation is to process at half the period (so twice as fast).

6.4.1. Sending the trajectory points – Java

```java
private void startFilling(double[][] profile, int totalCnt) {

    /* create an empty point */
    CANTalon.TrajectoryPoint point = new CANTalon.TrajectoryPoint();

    /* did we get an underrun condition since last time we checked ? */
    if (_status.hasUnderrun) {
        /* better log it so we know about it */
        instrumentation.OnUnderrun();
        /*
         * clear the error. This flag does not auto clear, this way
         * we never miss logging it.
         */
        _talon.clearMotionProfileHasUnderrun();
    }
    /*
     * just in case we are interrupting another MP and there is still buffer
     * points in memory, clear it.
     */
    _talon.clearMotionProfileTrajectories();

    /* This is fast since it's just into our TOP buffer */
    for (int i = 0; i < totalCnt; ++i) {
        /* for each point, fill our structure and pass it to API */
        point.position = profile[i][0];
        point.velocity = profile[i][1];
        point.timeDurMs = (int) profile[i][2];
        point.profileSlotSelect = 0; /* which set of gains would you like to
        point.velocityOnly = false; /* set true to not do any position
                                     * servo, just velocity feedforward
                                     */

        point.zeroPos = false;
        if (i == 0)
            point.zeroPos = true; /* set this to true on the first point */

        point.isLastPoint = false;
        if ((i + 1) == totalCnt)
            point.isLastPoint = true; /* set this to true on the last point

        _talon.pushMotionProfileTrajectory(point);
    }
}
```

For example, this routine takes the double-array of trajectory points and passes it into the Talon object. The routine `clearMotionProfileHasUnderRun()` is called first just in case we are interrupting a previous MP. Then `pushMotionProfileTrajectory()` is called once per point. These functions return immediately as the points are stored in the RIO initially. This buffer is referred to as the "Top-level" or API-level buffer.

If the profile is very large (2048 points or more) the function may return a nonzero error code. In which case caller can periodically call `pushMotionProfileTrajectory` () to stream the profile into the API, or use larger trajectory point durations, or modifying the library to increase the capacity.

Periodic calls to `processMotionProfileBuffer()` then empty the data points into the Talon's low-level (firmware) buffer. For this reason, this routine should be called quickly enough to keep pace with the execution of the profile, if the MP is firing before buffering is finished.

A conservative approach is to call the routine twice as fast as the MP. For example, if the MP uses 10ms trajectory points, therefore the `notifier` task that calls `processMotionProfileBuffer()` is set to fire every 5ms to ensure it has sufficient opportunity to funnel trajectory points into the Talon.

Typically, this can be done by creating a thread or task that calls the `processMotionProfileBuffer()`member function of the `CANTalon` object.

The function is re-entrant and does not require any “locking” strategy.

```
/**
 * Lets create a periodic task to funnel our trajectory points into our talon.
 * It doesn't need to be very accurate, just needs to keep pace with the motion
 * profiler executer.  Now if you're trajectory points are slow, there is no need
 * to do this, just call _talon.processMotionProfileBuffer() in your teleop loop.
 * Generally speaking you want to call it at least twice as fast as the duration
 * of your trajectory points.  So if they are firing every 20ms, you should call
 * every 10ms.
 */
class PeriodicRunnable implements java.lang.Runnable {
    public void run() {  _talon.processMotionProfileBuffer();    }
}
Notifier _notifer = new Notifier(new PeriodicRunnable());
```

Here’s where the period is set for our notifier. To be conservative, the transmit rate of the motion profile control CAN frame is set to match to ensure the communication is optimal.

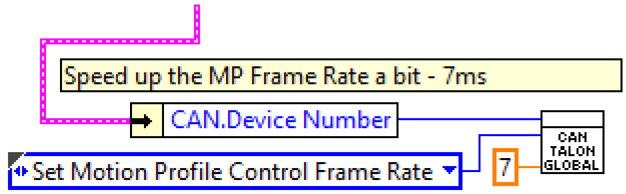
The benefit of this is that the precision of the notifier isn’t a factor in how smooth the motion profile executes. This means spikes in CPU don’t adversely affect the motion profile.

```
/*
 * since our MP is 10ms per point, set the control frame rate and the
 * notifer to half that
 */
_talon.changeMotionControlFramePeriod(5);
_notifer.startPeriodic(0.005);
```

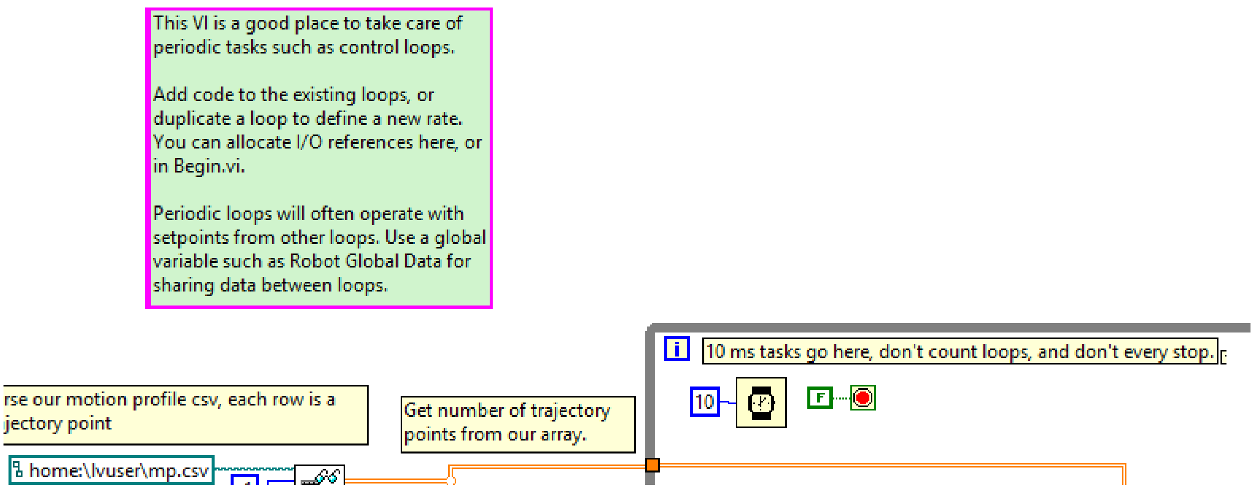
6.4.2. Sending the trajectory points - LabVIEW

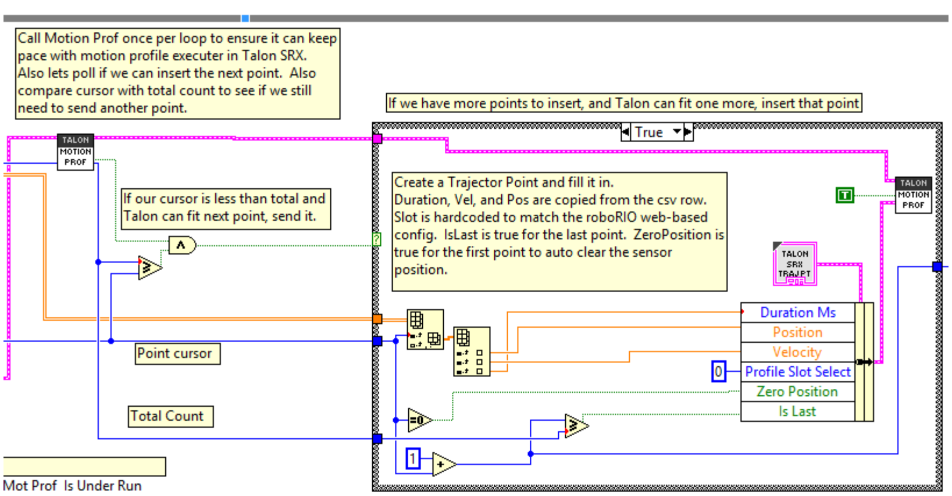
Similar to the API in the script-based languages, LabVIEW has a method for controlling the Motion Profile Control Frame Rate and a method to schedule tasks in a period fashion.

The CAN Global VI is used to change the motion profile control frame rate from the default value of 10ms to 7ms.



In the LabVIEW example, the Periodic Tasks VI is used for motion-profile tasking. It is ideal since it is timed and runs in parallel to the rest of robot application.





Here is a case structure that conditionally inserts the next trajectory point into in the CAN control frame if there is room for the next point, and if the next point is available.

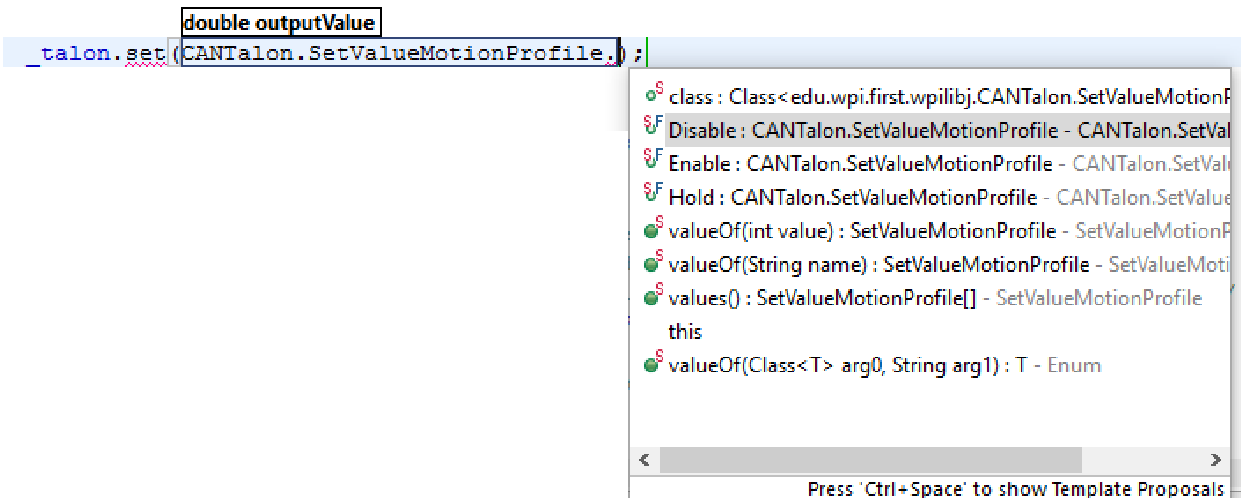
6.5. Activating the Motion Profile

Once the robot application has confirmed that there are points in the Talon buffer, the application can “fire” the buffered Motion Profile by setting the Talon output to ‘1’.

Care should be taken to not activate the executer until the robot application has confirmed there are trajectory points in the firmware buffer by polling the MPB status.

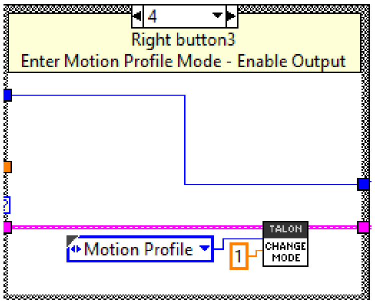
6.5.1. Activating the Motion Profile - Java

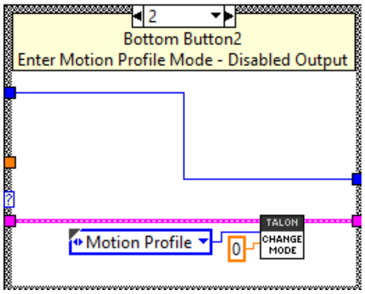
Pass a ‘1’ or `CANTalon.SetValueMotionProfile.Enable` to signal the Talon to start executing the buffered profile.

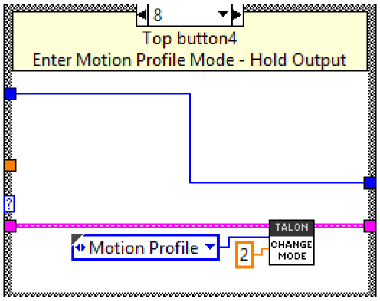


6.5.2. Activating the Motion Profile – LabVIEW

The motion profile executer can be controlled with the set value parameter of the Change Mode VI, or using the general motor set VI. In this example the control mode and Set Value are set at the same time.







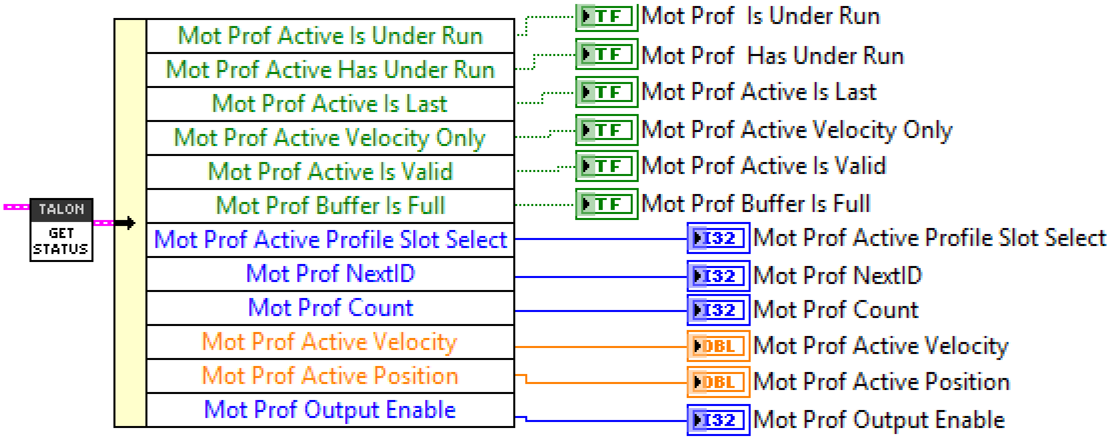
Care should be taken to only "hold" the active trajectory point if target velocity is zero.

6.6. Checking the Motion Profile Status

Robot application should check on the MP's status to determine if/when the MP is finished.

6.6.1. Checking the Motion Profile Status – LabVIEW

In LabVIEW, the status signals relating to Motion Profile are available in the general `GET STATUS` VI. Use the "Unbundle By Name" object and look for the "Mot Prof" prefix.



6.6.2. Checking the Motion Profile Status – Java

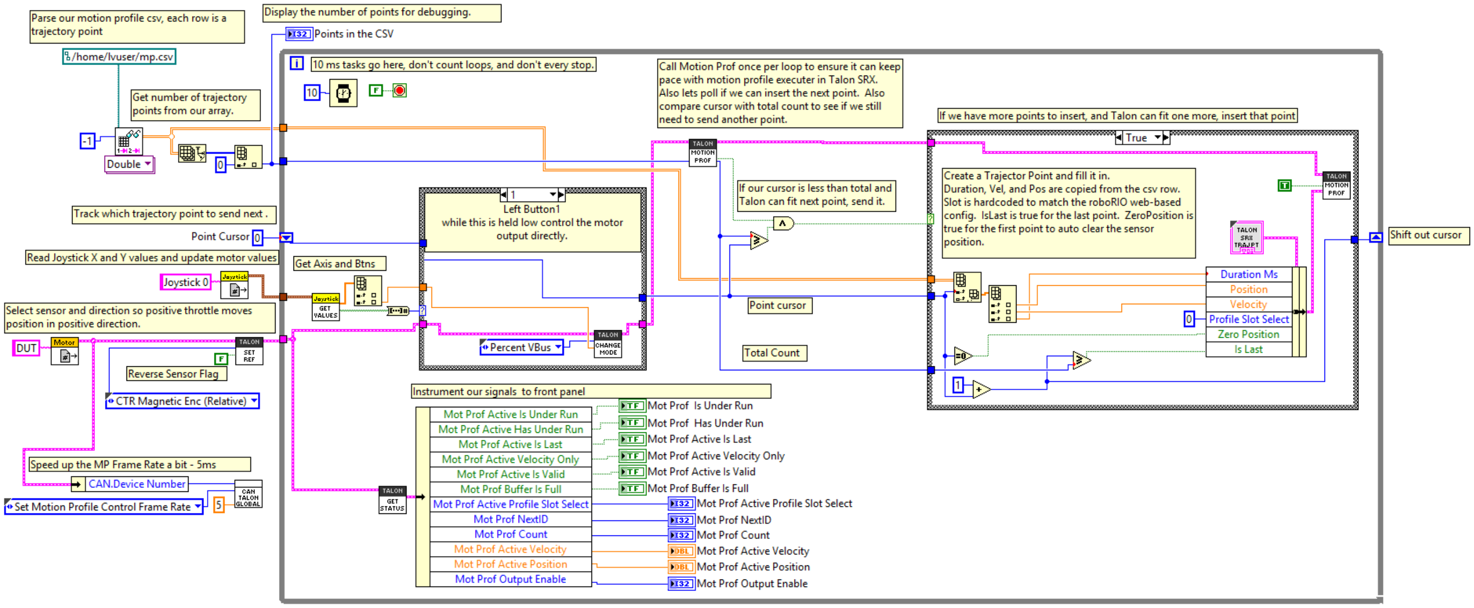
See Section 5.2 for example function call. Checking the status is necessary for…

- Determining that a sufficient number of trajectory points are in the MPB before activating the MPE.
- Determining when the MPE is in enable/disable/hold, after robot application has changed the desired state using set().
- Confirming MPE is in disable/hold before calling the clear and push routines for buffering trajectory points for the next motion profile. It is important to confirm that MPE is no longer interacting with MPB, before inserting new points into the MPB.

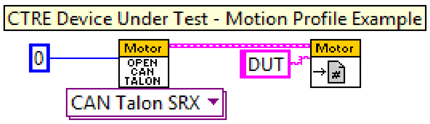
6.7. Complete Example Overview

6.7.1. Complete Example - LabVIEW

The LabVIEW example has all of the software integration steps completed in the Periodic Tasks VI. See Section 7 for download link.



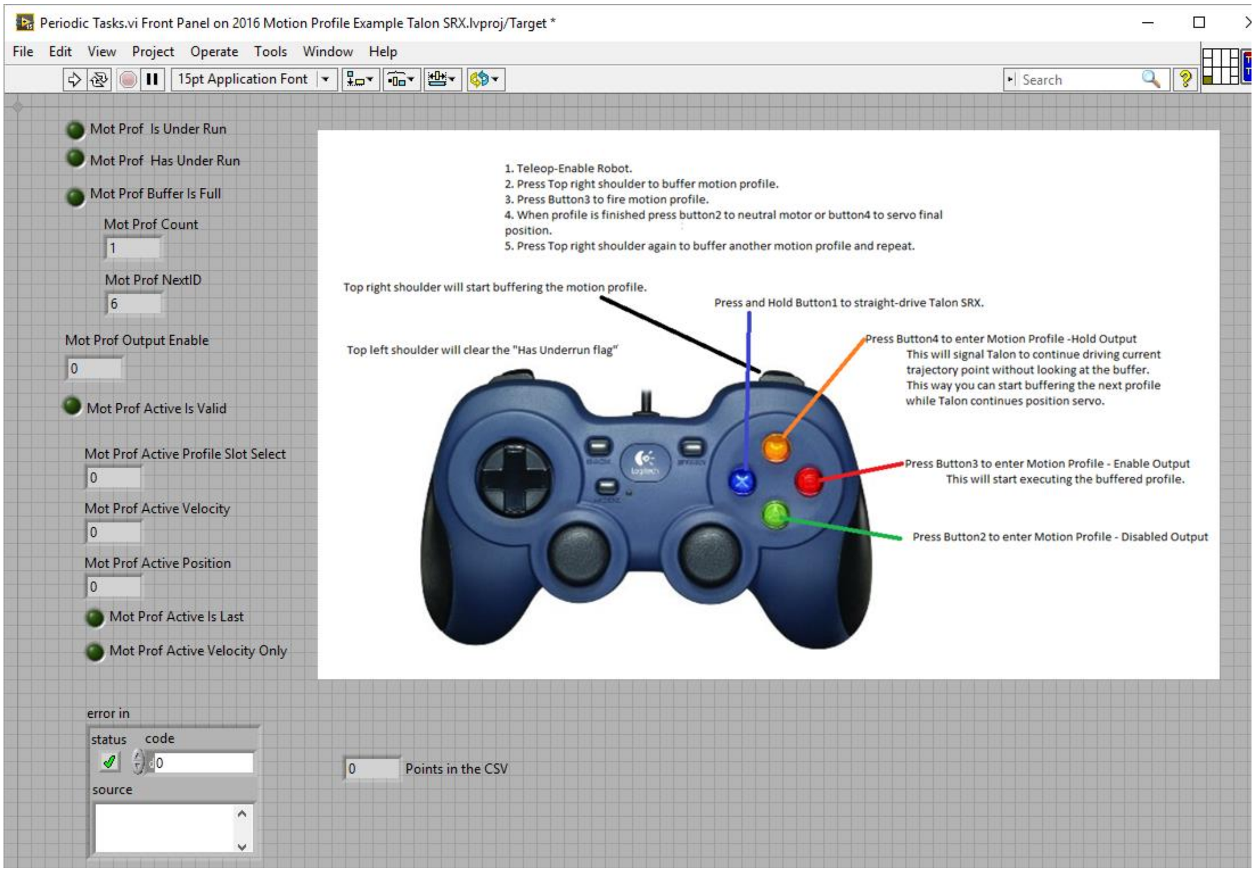
In Begin.vi, the Talon SRX reference is created with a CAN Device ID of '0'. See Talon SRX Software Reference Manual for more information on CAN Device IDs.



The string "DUT" is used to reference the Talon SRX. DUT stands for "Device Under Test", however most developers name the motor controller to something more specific: "arm, shooterWheel, LeftFrontDrive, etc.).

Instructions for testing are on the front-panel (below).
For example, manual control of the Talon can be be done by holding down Button 1 and using the left y-axis. If using another input-device, generally the “first” y-axis will control the Talon SRX.

Watching the intrumented signals on the left side while testing the example can also help users learn more about how this feature works.



6.7.2. Complete Example - Java

See Section 7 for download link. The project primary depends on two classes.

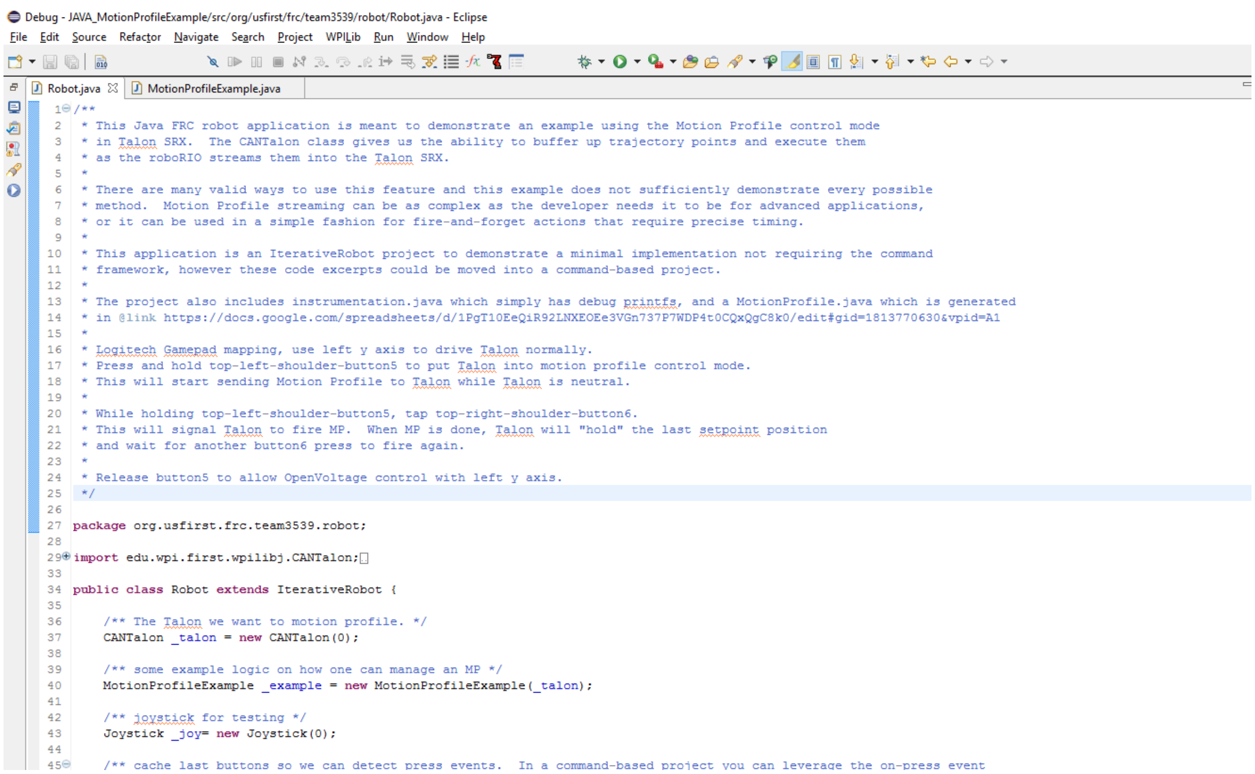
MotionProfileExample implements the integration steps in Section 6, including polling motion profile status and deciding when to "fire" the Motion Profile.

Robot.java creates a `MotionProfileExample` object and uses `startMotionProfile`() and `reset`() to signal the `MotionProfileExample` object what to do.

Be sure to look at the Output window to watch the changes in state of the Motion Profiler Executer.

Note that `MotionProfileExample` doesn't actually change the control mode or the set value. That is done in Robot.java so that logic for changing modes can be done in one place.

The CANTalon is created in Robot.java and uses the Device ID '0'. See Talon SRX Software Reference Manual for more information on CAN Device IDs.




```java
/**
 * This Java FRC robot application is meant to demonstrate an example using the Motion Profile control mode
 * in Talon SRX.  The CANTalon class gives us the ability to buffer up trajectory points and execute them
 * as the roboRIO streams them into the Talon SRX.
 *
 * There are many valid ways to use this feature and this example does not sufficiently demonstrate every possible
 * method.  Motion Profile streaming can be as complex as the developer needs it to be for advanced applications,
 * or it can be used in a simple fashion for fire-and-forget actions that require precise timing.
 *
 * This application is an IterativeRobot project to demonstrate a minimal implementation not requiring the command
 * framework, however these code excerpts could be moved into a command-based project.
 *
 * The project also includes instrumentation.java which simply has debug printfs, and a MotionProfile.java which is generated
 * in @link https://docs.google.com/spreadsheets/d/1PgT10EeQiR92LNXEOEe3VGn737P7WDP4t0CQxQgC8k0/edit#gid=1813770630&vpid=A1
 *
 * Logitech Gamepad mapping, use left y axis to drive Talon normally.
 * Press and hold top-left-shoulder-button5 to put Talon into motion profile control mode.
 * This will start sending Motion Profile to Talon while Talon is neutral.
 *
 * While holding top-left-shoulder-button5, tap top-right-shoulder-button6.
 * This will signal Talon to fire MP.  When MP is done, Talon will "hold" the last setpoint position
 * and wait for another button6 press to fire again.
 *
 * Release button5 to allow OpenVoltage control with left y axis.
 */

package org.usfirst.frc.team3539.robot;

import edu.wpi.first.wpilibj.CANTalon;[]

public class Robot extends IterativeRobot {

    /** The Talon we want to motion profile. */
    CANTalon _talon = new CANTalon(0);

    /** some example logic on how one can manage an MP */
    MotionProfileExample _example = new MotionProfileExample(_talon);

    /** joystick for testing */
    Joystick _joy= new Joystick(0);

    /** cache last buttons so we can detect press events.  In a command-based project you can leverage the on-press event
```

6.7.3. Complete Example - C++

See Section 7 for download link. The project primary depends on two classes.

MotionProfileExample implements the integration steps in Section 6, including polling motion profile status and deciding when to "fire" the Motion Profile.

Be sure to look at the Output window to watch the changes in state of the Motion Profiler Executer.

Robot.cpp creates a `MotionProfileExample` object and uses `startMotionProfile`() and `reset`() to signal the `MotionProfileExample` object what to do.

Note that `MotionProfileExample` doesn't actually change the control mode or the set value. That is done in Robot.java so that logic for changing modes can be done in one place.

The CANTalon is created in Robot.cpp and uses the Device ID '0'. See Talon SRX Software Reference Manual for more information on CAN Device IDs.

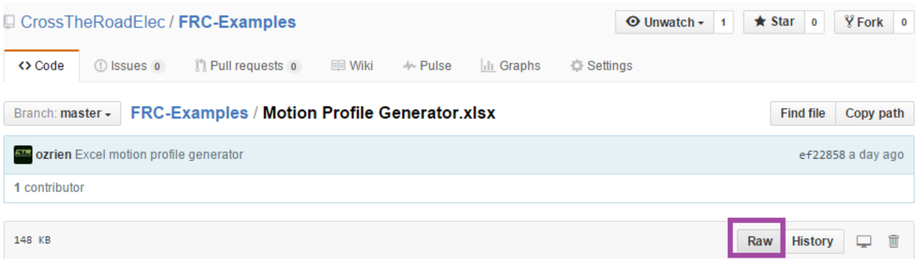
Robot.cpp | MotionProfileExample.h

```cpp
/**
 * This C++ FRC robot application is meant to demonstrate an example using the Motion Profile control mode
 * in Talon SRX.  The CANTalon class gives us the ability to buffer up trajectory points and execute them
 * as the roboRIO streams them into the Talon SRX.
 *
 * There are many valid ways to use this feature and this example does not sufficiently demonstrate every possible
 * method.  Motion Profile streaming can be as complex as the developer needs it to be for advanced applications,
 * or it can be used in a simple fashion for fire-and-forget actions that require precise timing.
 *
 * This application is an IterativeRobot project to demonstrate a minimal implementation not requiring the command
 * framework, however these code excerpts could be moved into a command-based project.
 *
 * The project also includes instrumentation.java which simply has debug printfs, and a MotionProfile.java which is generated
 * in @link https://docs.google.com/spreadsheets/d/1PgT10EeQiR92LNXEOEe3VGn737P7WDP4t0CQxQgC8k0/edit#gid=1813770630&vpid=A1
 *
 * Logitech Gamepad mapping, use left y axis to drive Talon normally.
 * Press and hold top-left-shoulder-button5 to put Talon into motion profile control mode.
 * This will start sending Motion Profile to Talon while Talon is neutral.
 * This will signal Talon to fire MP.  When MP is done, Talon will "hold" the last setpoint position
 * and wait for another button6 press to fire again.
 * Release button5 to allow OpenVoltage control with left y axis.
 */
#include <Instrumentation.h>
#include "WPILib.h"
#include "MotionProfileExample.h"

class Robot: public IterativeRobot
{
public:
    /** The Talon we want to motion profile. */
    CANTalon _talon;

    /** some example logic on how one can manage an MP */
    MotionProfileExample _example;

    /** joystick for testing */
    Joystick _joy;

    /** cache last buttons so we can detect press events.  In a command-based project you can leverage the on-press event
     * but for this simple example, lets just do quick compares to prev-btn-states */
    bool _btnsLast[10] = {false,false,false,false,false,false,false,false,false,false};


    Robot() : _talon(6), _example(_talon), _joy(0)
    {
        _talon.SetFeedbackDevice(CANTalon::CtreMagEncoder_Relative);
```

7. Download the Examples

Generally speaking, all source and generator files can be found in https://github.com/CrossTheRoadElec under the "FRC-Examples" repository.

7.1. Download a file "as is" from GitHub



When reviewing a non-text based file, press the "Raw" button to download the file as-is.

7.2. Download links

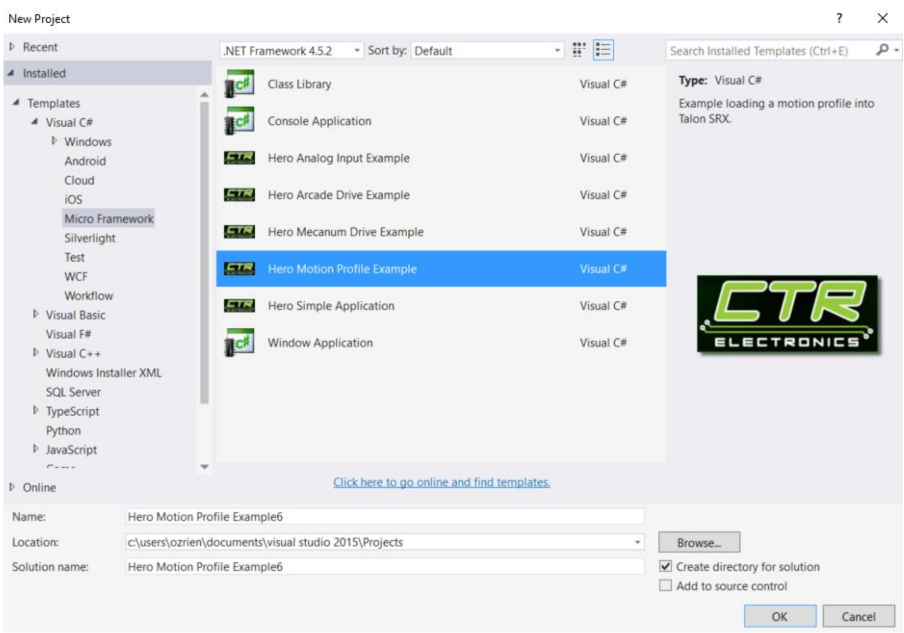
These links are tested at the time of writing. However, these resources can also be found by navigating through the CTRE GitHub account.

Motion Profile Generator Excel Sheet
https://github.com/CrossTheRoadElec/FRC-Examples/tree/master/Motion Profile Generator.xlsx
Java Motion Profile Example
https://github.com/CrossTheRoadElec/FRC-Examples/tree/master/JAVA_MotionProfileExample
LabVIEW Motion Profile Example
https://github.com/CrossTheRoadElec/FRC-Examples/tree/master/LabVIEW_MotionProfileExample
C++ Motion Profile Example
https://github.com/CrossTheRoadElec/FRC-Examples/tree/master/CPP_MotionProfileExample

7.3. Example – HERO C#

For HERO development board users, an example Motion Profile project can be found after installing the HERO-SDK-Installer and using the default Visual Studio HERO example project.

Example Visual Studio Project is also available at the CTRE GitHub Account.



8. Suggested Testing / General Recommendations

Additionally, testing is recommended to ensure robot responds in an expected fashion if the Talon motor controller is power cycled or disconnected from robot controller during a motion profile. The motion profile control mode is unique in that information is *streamed* to a motor controller, so be sure to test your robot's response to intermittent connections where the stream is momentarily or permanently severed (disconnected CAN wires or unpowered Talon).

As with all advanced control modes, it's often helpful to have an override mode to allow the human operator to manually control a mechanism (sensor failure or alignment, sensor disconnect, mechanical failures, gear-teeth skipping, software issue, etc.).

Having a method to "re-zero" or "re-tare" your sensors can also be helpful (see Section 16.19 in Talon Software Reference Manual).

9. Troubleshooting Tips and Common Questions

9.1. Where can I find the other resources mentioned? Software Reference Manual, Motion profile generator, example source?

Under the "Tech Resources" tab on the Talon SRX product page.
http://www.ctr-electronics.com/talon-srx.html#product_tabs_technical_resources
Programming examples are mentioned in Section 7.

9.2. What motor controllers, which firmware is required for this feature?

This document assumes that you are using **Talon SRX** wired to **CAN Bus**.
The firmware version requirements are…

FRC: equal to or **greater than 2.0**.

Non-FRC or general use: equal to or **greater than 10.0**.

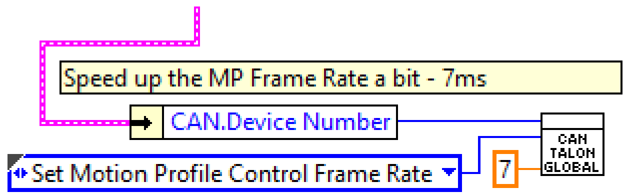
10. Functional Limitations

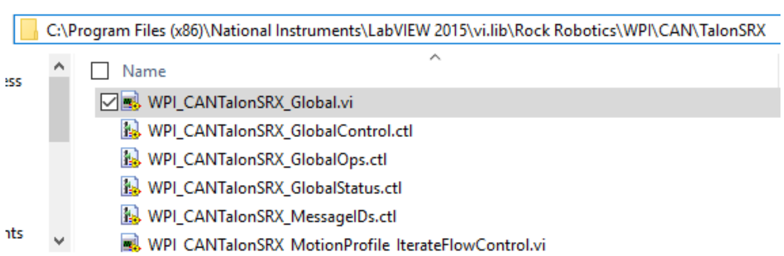
10.1. C++ References missing in document.

Because of how similar the C++ and Java examples are, this document references the Java example only. However, the C++ example is **nearly-line-for-line identical** outside of the obvious language differences between C++ and Java. The C++ example works identically to the Java example, therefore following the Java document references should be sufficient for C++ users.

10.2. LabVIEW: No VI for setting the Motion Profile Control Frame Rate

There is no specific VI for modifying the motion profile control frame rate. The frame rate can be controlled using the `CAN Talon GLOBAL VI`.

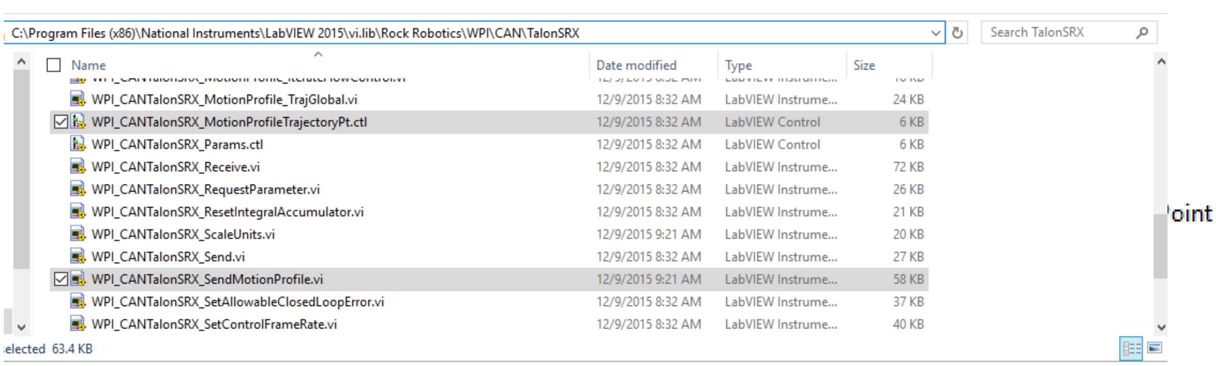




Simply drag-and-drop "WPI_CANTalonSRX_Global.vi" into your block diagram. This can be found in the install path below…

10.3. LabVIEW: Motion Profile Vis are not accessible in LabVIEW palette

VI and CTL can be drag and dropped into the LabVIEW block diagram view.



11. Revision History

Rev	Date	Description
1.0	19-Jan-2016	-Initial Release for 2016 FRC Season