

PARTs Subsystem

Explanation & History

Keep in Mind

This includes IPARTsSubsystem as it acts as the recipe for the subsystem.

We have all these tools that we use but now we run in the issue of repeating all this code just to use them. We also want a good custom logging and telemetry solution. This is what inspired us to make our custom subsystem too.

With that, our subsystem currently features:

- Telemetry
- Logging
- Stop
- Reset

We included both stop and reset features for convenience and simplicity.

Code Example

Please note that the PARTs Subsystem is meant to be extended, not used directly.

```
package;  
  
// Import the subsystem so we can extend it.  
import org.parts3492.partslib.command.PARTsSubsystem;  
  
public class MySubsystem extends PARTsSubsystem {  
  
    private boolean publishTelemetry = true;  
    private boolean enableLogging = true;  
  
    // My awesome logged string.  
    public String myAwesomeString = "Hello world!";  
}
```

```
public MySubsystem() {
    super();
}

@Override
public void outputTelemetry() {
    partsNT.putString("My Awesome Telemetry", myAwesomeString, publishTelemetry);
}

@Override
public void stop() {
    // Stop the logging and telemetry since the subsystem is stopped.
    publishTelemetry = false;
    enableLogging = false;
}

@Override
public void reset() {
    // Reset the telemetry and logging to be published again when we reset the subsystem.
    publishTelemetry = true;
    enableLogging = true;
}

@Override
public void log() {
    // Log our string.   The log name.   The log value.   Whether to log or not.
    partsLogger.logString("My Awesome Log", myAwesomeString, enableLogging);
}
}
```

Revision #2

Created 30 March 2026 18:35:26 by Rowan Buchecker

Updated 10 April 2026 11:55:58 by Rowan Buchecker