**ASCOM Focuser Variables and Methods**

public bool Connected {

get {

LogMessage("Connected", "Get {0}", IsConnected);

return IsConnected;

}

set {

tl.LogMessage("Connected", "Set {0}", value);

if (value == IsConnected) return;

if (value) {

connectedState = true;

LogMessage("Connected Set", "Connecting to port {0}", comPort);

// TODO connect to the device

} else {

connectedState = false;

LogMessage("Connected Set", "Disconnecting from port {0}", comPort);

// TODO disconnect from the device

}

}

}

public string Description {

// TODO customise this device description

get {

tl.LogMessage("Description Get", driverDescription);

return driverDescription;

}

}

public string DriverInfo {

get {

Version version = System.Reflection.Assembly.GetExecutingAssembly().GetName().Version;

// TODO customise this driver description

string driverInfo = "Information about the driver itself. Version: " + String.Format(CultureInfo.InvariantCulture, "{0}.{1}", version.Major, version.Minor);

tl.LogMessage("DriverInfo Get", driverInfo);

return driverInfo;

}

}

public string DriverVersion{

get {

Version version = System.Reflection.Assembly.GetExecutingAssembly().GetName().Version;

string driverVersion = String.Format(CultureInfo.InvariantCulture, "{0}.{1}", version.Major, version.Minor);

tl.LogMessage("DriverVersion Get", driverVersion);

return driverVersion;

}

}

public short InterfaceVersion { // set by the driver wizard

get {

LogMessage("InterfaceVersion Get", "2");

return Convert.ToInt16("2");

}

}

public string Name {

get {

string name = "Short driver name - please customise";

tl.LogMessage("Name Get", name);

return name;

}

}

public bool Absolute {

get {

tl.LogMessage("Absolute Get", true.ToString());

return true; // This is an absolute focuser

}

}

public void Halt() {

tl.LogMessage("Halt", "Not implemented");

throw new ASCOM.MethodNotImplementedException("Halt");

}

public bool IsMoving {

get {

tl.LogMessage("IsMoving Get", false.ToString());

return false; // This focuser always moves instantaneously so no need for IsMoving ever to be True

}

}

public bool Link {

get {

tl.LogMessage("Link Get", this.Connected.ToString());

return this.Connected; // Direct function to the connected method, the Link method is just here for backwards compatibility

}

set {

tl.LogMessage("Link Set", value.ToString());

this.Connected = value; // Direct function to the connected method, the Link method is just here for backwards compatibility

}

}

public int MaxIncrement {

get {

tl.LogMessage("MaxIncrement Get", focuserSteps.ToString());

return focuserSteps; // Maximum change in one move

}

}

public int MaxStep {

get {

tl.LogMessage("MaxStep Get", focuserSteps.ToString());

return focuserSteps; // Maximum extent of the focuser, so position range is 0 to 10,000

}

}

public void Move(int Position) {

tl.LogMessage("Move", Position.ToString());

focuserPosition = Position; // Set the focuser position

}

public int Position {

get {

return focuserPosition; // Return the focuser position

}

}

public double StepSize {

get {

tl.LogMessage("StepSize Get", "Not implemented");

throw new ASCOM.PropertyNotImplementedException("StepSize", false);

}

}

public bool TempComp {

get {

tl.LogMessage("TempComp Get", false.ToString());

return false;

}

set {

tl.LogMessage("TempComp Set", "Not implemented");

throw new ASCOM.PropertyNotImplementedException("TempComp", false);

}

}

public bool TempCompAvailable {

get {

tl.LogMessage("TempCompAvailable Get", false.ToString());

return false; // Temperature compensation is not available in this driver

}

}

public double Temperature {

get {

tl.LogMessage("Temperature Get", "Not implemented");

throw new ASCOM.PropertyNotImplementedException("Temperature", false);

}

}