Application example: Automation of an index punching machine with NanoJ

For a customer who controls the precision feed control of the punching knife in an index punching machine with a SMCI47-S-2 and Nanopro, the solution should be automated so that a PLC integrated in the line can move to the individual punching positions. The teach-in for a new order should continue to be carried out on a laptop, although the function of the inputs and outputs has been programmed in the SMCI47-S with NanoJ so that the program could be very simply configured on the superordinate PLC.
Index punching requires the possibility to move to up to 31 individual index positions, each with a separately adjustable length. These positions are saved by the machine setter in individual positioning records of the controllers via Nanopro.
The precision feed control of the punch is controlled by the PLC over five digital inputs of the SMCI47-S and the reference switch is connected to the sixth input of the controller:

INPUT	
1-	“Emergency off”
2-	“Next position”
3-	“Start reference run”
4-	“Move back to last position”
5-	“Reset to target position 1”
6-	Reference switch

After switching on the machine, the PLC uses Input 3 to first carry out a reference travel to the reference switch connected to Input 6. The machine is then moved to the next punching position by a signal on Input 2. If the operator notices that the punching is unclean, the machine can be moved back one position each time by Input 4 and the punching repeated.
If all indexes of a book are punched, the counter is set back to the first index position again by Input 5.
If the operator presses the emergency stop button on the machine, the motor is brought to a controlled stop by Input 1.

Safeguarded against scrap by encoder monitoring
An AP8918M6404-E was selected as the motor in protection class IP65 that is not adversely affected by the dust that occurs during the punching process. The encoder integrated in the motor is monitored by the SMCI47-S and ensures that the controller registers an error if there is any mechanical stiffness or blockage, for example, to prevent the index from being brought to the wrong position.

```
import nanotec.io;                                          // verwendete Includes einbinden//
import nanotec.util;
import nanotec.drive;

class neu_kommentiert
{
    public static void main ()                              //Hauptprogramm aufrufen//
    {
        int satz = 1;                                       //Zähler für die Fahrprofilwahl definieren//

        while(true){                                        //Endlosschleife starten//

        //Funktion für Eingang 2, Schritt starten und Nächsten laden//
            if(( io.GetDigitalInput () == 2)&&(satz==32 ))  //Wenn Eingang 2 aktiviert und
            {                                               //Zähler gleich 32, dann...//
                drive.LoadDataSet(satz);                    //Datensatz entsprechend Zähler aus EEPROM laden//
                drive.StartDrive();                         //Fahrprofil starten//
                satz =1;                                    //Wert von Zähler auf 1 setzen//
                while(io.GetDigitalInput () == 2){};        //Warten bis Eingang 2 nicht mehr aktiv//
                util.Sleep(200);                            //zeitliche Sicherheit (Endprellzeit)//
            }

        //Funktion für Eingang 1, Not-Aus//
            if( io.GetDigitalInput( ) == 1)                 //Wenn Eingang 1 aktiviert, dann...//
            {
                drive.StopDrive(0);                         //Fahrt sofort stoppen//
                while(io.GetDigitalInput () == 1){};        //Warten bis Eingang 1 nicht mehr aktiv//
                util.Sleep(200);                            //zeitliche Sicherheit (Endprellzeit)//
            }

        //Funktion für Eingang 2 wenn letzter Schritt erreicht, Schritt starten und Schritt 1 laden//
            if(( io.GetDigitalInput () == 2)&&(satz!=32 ))  //Wenn Eingang 2 aktiviert und
            {                                               //Zähler ungleich 32, dann...//
                drive.LoadDataSet(satz);                    //Datensatz entsprechend Zähler aus EEPROM laden//
                drive.StartDrive();                         //Fahrprofil starten//
                satz =satz+1;                               //Wert von Zähler um 1 vergrößert//
                while(io.GetDigitalInput () == 2){};        //Warten bis Eingang 2 nicht mehr aktiv//
                util.Sleep(200);                            //zeitliche Sicherheit (Endprellzeit)//
            }

        //Funktion für Eingang 5, wieder auf Schritt 1 stellen (RESET)//
            if( io.GetDigitalInput( ) == 16)                //Wenn Eingang 5 aktiviert, dann...//
            {
```