1 Device: Device

Users and Groups

Users:

Groups

Access Rights

View
Modify
Execute
Add/remove children

Symbol Rights

Information

Name:	CODESYS Control Win V3
Vendor:	CODESYS
Categories:	PLCs
Type:	4096
ID:	0000 0001
Version:	3.5.20.40
Order number:	???
Description:	CODESYS V3 Soft-PLC for Windows with non realtime capabilities (CODESYS Control Win V3)

1.1 PLC Logic: Plc Logic

1.1.1 Application: Application

1.1.1.1 Folder: Days

1.1.1.1.1 Interface: IPuzzle

```
1       INTERFACE  IPuzzle
2
```

1.1.1.1.1.1 Interface method: Solve

```
METHOD Solve

```

1.1.1.2 Folder: Utility

1.1.1.2.1 POU: ArrayInsertSorted

```
FUNCTION ArrayInsertSorted : BOOL
VAR_INPUT
    Element : DINT;
    AreaStart : DINT;
    AreaEnd : DINT;
END_VAR

VAR_IN_OUT
    TargetArray : ARRAY[*] OF DINT;
END_VAR

VAR_OUTPUT
    ErrorCode : STRING := 'OK';
END_VAR

VAR
    i : DINT;
    arrayStart : DINT := LOWER_BOUND(TargetArray, 1);
    arrayEnd : DINT := UPPER_BOUND(TargetArray, 1);
END_VAR

```

```
IF AreaStart < arrayStart OR AreaEnd > arrayEnd THEN
    ErrorCode := 'Target area outside of array bounds.';
    ArrayInsertSorted := FALSE;
    RETURN;
END_IF

FOR i := AreaStart TO AreaEnd DO
    // Insert before the first larger element
    IF TargetArray[i] > Element THEN
        ShiftArray(
            Direction := ShiftDirection.RIGHT,
            ShiftAmount := 1,
            Start := i,
            End := AreaEnd,
            TargetArray := TargetArray);

        TargetArray[i] := Element;
        ArrayInsertSorted := TRUE;
        RETURN;
    END_IF
END_FOR

// All elements are smaller, append at the end.
TargetArray[AreaEnd + 1] := Element;
ArrayInsertSorted := TRUE;
RETURN;

```

1.1.1.2.2 POU: LineReader

```
FUNCTION_BLOCK LineReader
VAR CONSTANT
    MAX_LINES : __XWORD := 999;
    MAX_LINE_LENGTH : INT := 255;
END_VAR
VAR_INPUT
    FilePath : STRING;
END_VAR
VAR_OUTPUT
   Done : BOOL := FALSE;
   Error : BOOL := FALSE;
   ErrorCode : STRING := '';
   ReadLines : ARRAY[0..MAX_LINES] OF STRING(MAX_LINE_LENGTH);
   LineCount : WORD := 0;
END_VAR
VAR
    CurrentLine : STRING(MAX_LINE_LENGTH) := '';
    FileHandle : SysFile.RTS_IEC_HANDLE;
    BytesRead : __XWORD := 0;
    CharBuffer : STRING(1);
    pResult : POINTER TO SysFile.RTS_IEC_RESULT;
    i : __XWORD := 0;
    Initialized : BOOL := FALSE;
END_VAR

```

```
IF Initialized = FALSE THEN
    FOR i := 0 TO MAX_LINES DO
        ReadLines[i] := 'NOT WRITTEN TO';
    END_FOR
    Initialized := TRUE;
END_IF

FileHandle := SysFileOpen(szFile := FilePath, am := SysFile.AM_READ, pResult := pResult);

IF FileHandle = SysFile.RTS_INVALID_HANDLE THEN //File not found
    ErrorCode := 'File not found, is it on the device?';
    Error := TRUE;
    Done := TRUE;
ELSE // File open
    WHILE NOT Done DO
        BytesRead := SysFileRead(hFile := FileHandle, pbyBuffer := ADR(CharBuffer), ulSize := 1, pResult := pResult);

        IF BytesRead > 0 THEN
            // A little bit of cheating, only read files with unix line endings
            IF CharBuffer = '$N' THEN
                IF LineCount <= MAX_LINES THEN
                    ReadLines[LineCount] := CurrentLine;
                    LineCount := LineCount + 1;
                    CurrentLine := '';
                ELSE
                    ErrorCode := 'Input file is larger than max line count.';
                    Error := TRUE;
                    Done := TRUE;
                END_IF

            ELSE
                IF LEN(CurrentLine) <= MAX_LINE_LENGTH THEN
                    CurrentLine := CONCAT(CurrentLine, CharBuffer);
                ELSE
                    ErrorCode := 'Line exceedes max line length';
                    Error := TRUE;
```

```
                    Done := TRUE;
                END_IF
            END_IF
        ELSE
            Done := TRUE;
            EXIT;
        END_IF
    END_WHILE;
END_IF

SysFileClose(FileHandle);

```

1.1.1.2.3 POU: LineToNumbers

```
FUNCTION LineToNumbers : INT
VAR_INPUT
    line            : STRING;
    separator       : STRING;
END_VAR
VAR_IN_OUT
    numberBuffer    : ARRAY[*] OF DINT;
END_VAR
VAR_OUTPUT
    numberCount     : INT           := 0;
END_VAR
VAR
    separatorLength : INT           := LEN(separator);
    bufferStart     : DINT              := LOWER_BOUND(numberBuffer, 1);
    bufferEnd       : DINT              := UPPER_BOUND(numberBuffer, 1);
    bufferIndex     : INT           := 0;
    endReached      : BOOL              := FALSE;
    separatorIndex  : INT           := 0;
    tmp             : STRING        := '';
    startRight      : INT           := 0;
    emergencyBreak  : INT           := 0;
END_VAR

```

```
WHILE endReached = FALSE DO
    // Just in case something goes wrong,
    // endless loops are annoying.
    emergencyBreak := emergencyBreak + 1;
    IF emergencyBreak > 100 THEN
        LineToNumbers := -1;
        RETURN;
    END_IF

    separatorIndex := FIND(line, separator);
    IF separatorIndex > 0 THEN
        tmp := LEFT(line, separatorIndex - 1);
        startRight := LEN(line) - (separatorIndex + separatorLength - 1);
        line := RIGHT(line, startRight);
    ELSE
        tmp := line;
        endReached := TRUE;
    END_IF

    bufferIndex := DINT_TO_INT(bufferStart + numberCount);
    IF bufferIndex <= bufferEnd THEN
        numberBuffer[bufferIndex] := STRING_TO_INT(tmp);
        numberCount := numberCount + 1;
    ELSE
        LineToNumbers := -2;
        RETURN;
    END_IF
```

```
28      END_WHILE
29
30      LineToNumbers  :=  numberCount ;
31
```

1.1.1.2.4 POU: ShiftArray

```
1       FUNCTION  ShiftArray  :  BOOL
2       VAR_INPUT
3           Direction    :  ShiftDirection ;
4           ShiftAmount  :  DINT ;
5           Start        :  DINT ;
6           End          :  DINT ;
7       END_VAR
8       VAR_IN_OUT
9           TargetArray  :  ARRAY [ * ]  OF  DINT ;
10      END_VAR
11      VAR_OUTPUT
12          ErrorCode    :  STRING  :=  'OK' ;
13      END_VAR
14      VAR
15          i  :  DINT ;
16          arrayStart   :  DINT  :=  LOWER_BOUND ( TargetArray ,  1 ) ;
17          arrayEnd     :  DINT  :=  UPPER_BOUND ( TargetArray ,  1 ) ;
18          target       :  DINT ;
19      END_VAR
20
```

```
1       IF  Start  <  arrayStart  OR  End  >  arrayEnd  THEN
2           ErrorCode  :=  'Shift start/end outside of array bounds' ;
3           ShiftArray  :=  FALSE ;
4           RETURN ;
5       END_IF
6
7       // Shift logic: Always start at the outer bound to prevent overriding values.
8       IF  Direction  =  ShiftDirection . RIGHT  THEN
9           FOR  i  :=  End  TO  Start  BY  - 1  DO
10              target  :=  i  +  ShiftAmount ;
11              IF  target  <=  arrayEnd  AND  target  >=  arrayStart  THEN
12                  TargetArray [ target ]  :=  TargetArray [ i ] ;
13              END_IF
14          END_FOR
15      END_IF
16
17      IF  Direction  =  ShiftDirection . LEFT  THEN
18          FOR  i  :=  Start  TO  End  BY  + 1  DO
19              target  :=  i  -  ShiftAmount ;
20              IF  target  <=  arrayEnd  AND  target  >=  arrayStart  THEN
21                  TargetArray [ target ]  :=  TargetArray [ i ] ;
22              END_IF
23          END_FOR
24      END_IF
25
26      ShiftArray  :=  TRUE ;
27      RETURN ;
28
```

1.1.1.2.5 POU: SIGN

```
FUNCTION SIGN : INT
VAR_INPUT
    number : DINT;
END_VAR

```

```
IF number > 0 THEN
    SIGN := 1;
ELSIF number < 0 THEN
    SIGN := -1;
ELSE
    SIGN := 0;
END_IF

```

1.1.1.3 POU: PLC_PRG

```
PROGRAM PLC_PRG
VAR_OUTPUT
    Decoration      : BOOL := TRUE;
END_VAR
VAR
    day1            : Day1;
    day2            : Day2;
    puzzles         : ARRAY[0..1] OF IPuzzle := [day1, day2];
    puzzleEnabled   : ARRAY[0..1] OF BOOL;
    i               : DINT := 0;
END_VAR

```

```
// File not found? Put the input on the device with:
// Double click Device-> Files-> into the PlcLogic/inputs folder
FOR i := 0 TO UPPER_BOUND(puzzles, 1) DO
    IF puzzleEnabled[i] = TRUE THEN
        puzzles[i].Solve();
    END_IF
END_FOR

```