Industrial Automation

Tech Note 24 Crimson Control Introduction

Abstract:

The document serves as an introduction to Crimson Control.

Products:

Graphite Controllers & HMIs with Control Module

Use Case: Crimson Control Introduction

Understanding what Crimson Control is, and how to use it

Required Software:

Crimson® 3.0

Required Firmware: Heading3

Build 692.000+

Prerequisite Reading:

Crimson 3.0 Quick Start Guide: http://www.redlion.net/C3quickstart

Associated Reading:

Crimson 3.0 Runtime Overview: http://www.redlion.net/TNIA16

IEC61131

- Vendor independent standardized programming for industrial automation
- Defined by International Electrotechnical Commission (IEC)
- Driven by need to control increasing complexity and costs of meeting automation requirements
- Defines data types, configuration resources and tasks, program organization and languages/commands
- Transferrable source code between vendors
- Actively advocated by PLCOpen
- IEC61131 contains five parts:

  1. General Overview
  2. Hardware
  3. **Programming Languages**
  4. User Guidelines
  5. Communication

- Part 3 defines five languages:

  1. **LD – Ladder Diagram***
  2. SFC – Sequential Function Chart
  3. **FBD – Function Block Diagram***
  4. **ST – Structured Text***
  5. **IL – Instruction List***

  ***** indicates Crimson Control supported language

Crimson Programming vs Crimson Control

Crimson Programming

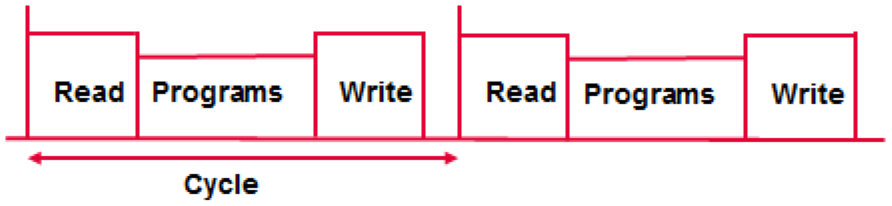
- Event driven
- Proprietary, C-like language
- Multiple External Data retrieval options
- Lack of communications may halt execution

Crimson Control

- Cyclically driven
- IEC61131 based languages
  - o Ladder
  - o Function Block
  - o Structured Text
  - o Instruction List
- External Data is automatically added to the comms scan
- Last know values ALWAYS used
  - Lack of communications will NOT break execution
- Cycle is NOT transactional

Cycle Execution

- A project is a list of programs executed sequentially according to the following model:
  - Begin cycle
    - *Read I/O*
    - *Execute first program*
    - …
    - *Execute last program*
    - *Write I/O*
  - Wait for cycle time to elapse
  - End cycle



- To change the execution order click on Project (Execution tab), then rearrange the programs

Navigation Pane – Control

When developing a new Crimson application, the workflow typically starts with the Modules or Communications section, then moves on to the Data Tags. When developing a new Crimson application that will include Control, the Control section should be configured prior to moving on to the Data Tags; data tags can be created from the Control area, saving the steps of creating folders and tags then later linking them to Control.

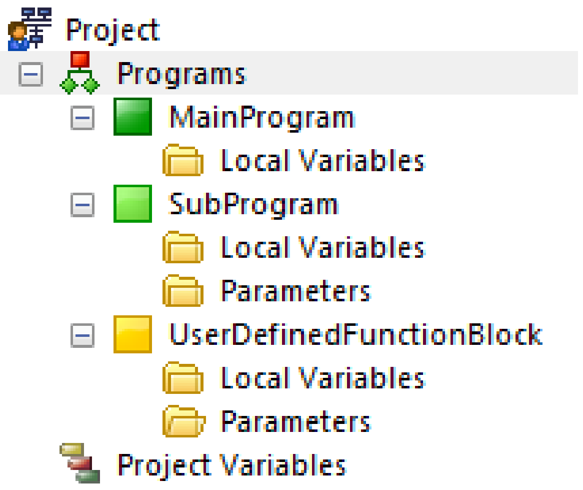
The Project tree has two main categories:

- Programs
  - o Local Variables (for the given program)
  - o Parameters: for Sub-Programs and User Defined Function Blocks
- Project Variables
  - o Project Variables (can be used in any CONTROL program)

Program Types

Main Program

- Can ONLY be called from the Project Scan
- Their Execution Order within the cycle can be changed
- Do NOT need to run every cycle
  - o Period: Run every X cycles
  - o Phase: Offset of Period
    - - *3* - *0*
    - - - *1*
    - - - *2*
    - - *3* - *0*
    - - - *1*
    - - - *2*


Project
Programs
MainProgram
Local Variables
SubProgram
Local Variables
Parameters
UserDefinedFunctionBlock
Local Variables
Parameters
Project Variables

Sub-program

- Can be called from any other program
- Not instantiated
  - o The same memory space is used for the local variables every time the sub-program is called
- Can accept Inputs and provide Outputs

User-defined Function Block

- Can be called from any other program
- Instantiated
  - – Each *instance* of a User Defined Function Block gets its own memory space for local variables
- Can accept Inputs (arguments) and provide Outputs (returns)

Program Execution

In order for the Project to execute its programs, the Program Execution needs to be set to *Execute periodically*. **Program execution is DISABLED by default**. To access the Program execution property, click on Control in the bottom of the Navigation Pane, and then select Project at the top of the Navigation Pane.

Variables

- **ALL external data used as the Source of writable variables will be written to the variable's initial value, which will be 0 or the last retained value, in the case of a Retentive Project Variable.**
- Project Variables
  - Variables available to all programs within the Project
  - Can be mapped to comms items OR data tags
  - Can be single items or single dimensional arrays
  - Can be set to retentive
- Local Variables
  - Variables available only to the program that they are defined in
  - Can be mapped to comms items OR data tags
  - Can be single items or single dimensional arrays

Type	Description	Values	Prefixes	Example
BOOL	Boolean (bit)	FALSE or TRUE	BOOL#	BOOL#TRUE
SINT	Small signed integer on 8 bits	-128 to +127	SINT#	SINT#-1
USINT (BYTE)	Small unsigned integer on 8 bits	0 to 255	USINT#	USINT#2
INT	Signed integer on 16 bits	-32768 to +32767	INT#	INT#-3
UINT (WORD)	Unsigned integer on 16 bits	0 to 65535	UINT#	UINT#4
DINT	Signed integer on 32 bits	-2147483648 to +2147483647	DINT#	DINT#-5
REAL	Single precision floating point	stored on 32 bits	REAL#	REAL#0.6
TIME	Duration - Accuracy is 1ms	0ms to 24h	T# or TIME#	T#1h2m3s4ms
STRING	Variable length string, ASCII not Unicode	Not to exceed 255 characters		

Crimson Control Variable Data Type	Crimson Data Tag Type
BOOL	Flag
SINT	Numeric
USINT (BYTE)	Numeric
INT	Numeric
UINT (WORD)	Numeric
DINT	Numeric
REAL	Numeric
TIME	Numeric (1ms resolution)
STRING	String

Naming Conventions for Programs and Variables

- First character must be a letter ('A' .. 'Z') or an underscore
- Following characters must be letters, digits or underscores
- You cannot enter 2 consecutive underscore characters
- Names are case insensitive
- Maximum length is 255 characters

Constant Expressions

- Boolean (BOOL):
  - Reserved keywords **TRUE** and **FALSE**.
- Integers:
  - Prefixed with type name and '#' sign
  - Example: USINT#123
- Reals
  - Must contain a dot '.'
  - Prefixed with type name and '#' sign
  - Scientific notation available
  - Example: REAL#0.123
  - Example: REAL#1.23E-5
- Duration (TIME):
  - Time constant expressions represent duration that must be less than 24 hours
  - Must be prefixed with "T#" or "TIME#"
  - They are expressed as a number of hours followed by "**h**"
  - A number of minutes followed by "**m**"
  - A number of seconds followed by "**s**"
  - And a number of milliseconds followed by "**ms**".
  - The order of units (hour, minutes, seconds, milliseconds) must be respected.
    - Example: *T#5h32m42s10ms*
- Character string (STRING):
  - Must be written between single quote marks – no prefix.
  - The length of the string cannot exceed 255 characters.
  - These sequences represent a special or non-printable character within a string:

Use	For
$$	"$" character
$'	Single quote
$T	Tab stop (ASCII 9)
$R	Carriage return (ASCII 13)
$L	Line feed (ASCII 10)
$N	Carriage return/Line feed (ASCII 13 and 10)
$P	Page break (ASCII 12)
$xx	Any ASCII character (ASCII xx)

Programming Languages

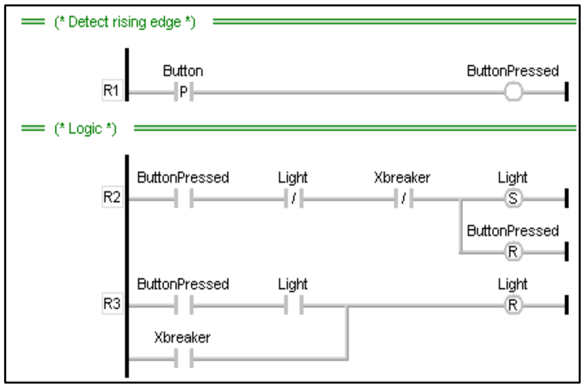
- Text Based
  - Instruction List
    - Looks like AWL (Anweisungsliste) by SIEMENS
    - Similar to assembly language

```
LD   Input1
ORN  Alarm2
ST   ABC1

CAL  TM1 (IN := cm1, T:=t#1s)
LD   TM1.Q
&    bAck
ST   bDone
```

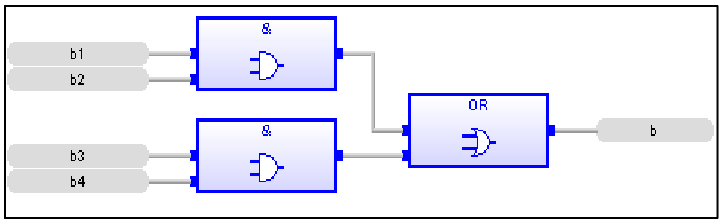
  - Structured Text (ST)
    - Looks like PASCAL
    - Loops (While, Until, For)
    - Condition (If then else...)

```
for i := 0 to 9 do

    if Arr[i].bEnable then
        Arr[i].sum := any_to_lint (Arr[i].di1 + Arr[i].di2);
    else
        Arr[i].sum := 0;
    end_if;

end_for;
```

- Graphical
  - Ladder Diagram (LD)
    - Similar to electrical diagram

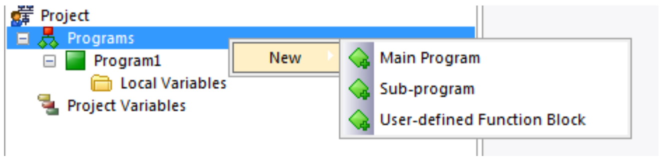


  - Function Block Diagram (FBD)
    - Programming using blocks
    - Data flow

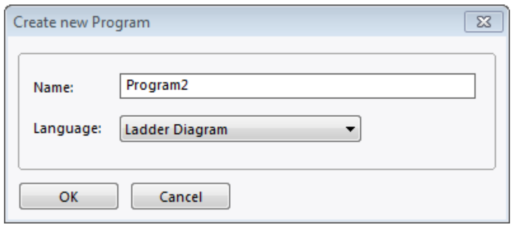


- Programs can be converted between all languages.

Adding Programs



1. Right-click Programs or a program name
2. Click *New*
3. Choose type of program to add
4. Assign a name to the new program
5. Choose the Language to create the program in


Create new Program

Name: Program2

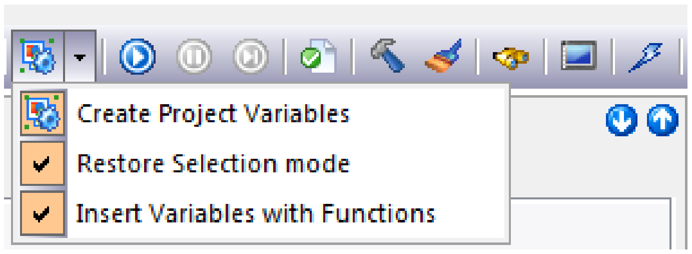
Language: Ladder Diagram

Adding and Mapping Variables

1. Right-click the Local Variables folder or Project Variables
2. Click *Variable*
3. Assign a name to the new variable
4. Choose the data Type for the new variable
5. Local and Project Variables can be mapped similar to Data Tags
   - In order to access the control variables (project and locals) outside of Crimson Control, their Source MUST be a Tag.
   - Changing the Source to *New Tag* will automatically recommend a helpful tag name and create it.

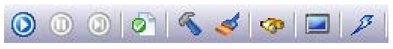
Programming

- Much like the rest of Crimson, function blocks and variables can be selected in the Resource Pane and dragged into the Program Code Window
- When dragging function blocks into a Function Block Diagram program, you can choose to Insert Variables with Functions.
  - o Saves dragging and connecting variables later.



- Ctrl-T checks Syntax
- Clean will delete all previously compiled code
- Build will compile the whole Project

Simulation

- The Project can be simulated by clicking the blue play button in the tool bar. 
- Double clicking a variable will allow you to edit its value
- The function blocks and variables will change color to show status

Programming Resources

- IEC61131 Overview
  - Control Engineering: https://www.controleng.com/industry-news/more-news/single-article/speaking-in-tongues-understanding-the-iec-61131-3-programming-languages/4123b0e66c3f2cb8bdd60d3cb20f944d.html
- Ladder
  - PLCS.net - Learn PLCS: http://www.plcs.net/contents.shtml
- Function Block Diagram
  - Many good examples on YouTube
- Structured Text
  - PLC Academy: http://www.plcacademy.com/structured-text-tutorial/

For more information: http://www.redlion.net/support/policies-statements/warranty-statement