What is Modular Programming?
Strategic Compartmentalization

It is breaking an application into its most fundamental components.
Then using those as building blocks to tackle a larger problem.
Why?

Pros. To be more efficient in the long run.
  1. It is faster to debug smaller, simpler code.
  2. You prove the code only once.
  3. No changes needed for each use.
  4. No copy and paste errors.

Cons. Larger effort up front
  1. Think of all of the uses for the code
  2. Design it for all the uses
  3. Revision control
Recent updates to make this easier
New Additional Non-Volatile Extended Memory

5370 / 5570 Controller

CPU

User Memory (Volatile/RAM)

New Memory (Non-Volatile)

Storage is Completely Independent of User Memory Space

New Memory (Non-Volatile)

- Comments Descriptions
- Alarm Log 10K Alarms
- Extended Tag Properties
Logix Extended Tag Properties

Decreased Troubleshooting Time & Faster Time to Market

- Simplify your Logix application
  - No need to create custom data structures for commonly used tag information
  - Develop your application faster and reduce errors
  - Use Min/Max tag limits programmatically, or propagate to third party HMI after moving to a local tag.
Logical Organizer

- **NEW Logical Organizer View**
  - Logical organization of programs to represent your process flow.
  - Easier to locate code even though programs span task scheduling
  - Create your own hierarchy
Library

- Library
  - Use .ACD as Library or XML files
  - Increased transportability
  - Data maintained and collisions managed
Program Parameters

- Program Parameters
  - Clearly defined interface for programs
  - Direct communication between programs
  - Increased modularity
Scalable and Flexible Solutions

Use modular code to simplify the application

V23 & Prior

Prog_01
Program Tags
Tag_4
Tag_5
Tag_6
Code Routines

Data

Prog_02
Program Tags
Tag_4
Tag_5
Tag_6
Code Routines

V24

Prog_01
Parameter
In
InOut
Out
Code Routines

Data

Prog_02
Parameter
In
Out
Code Routines

New capability (Program Parameters)
Scalable and Flexible Solutions

Recap...Modularity advancement

Logical Organizer

Program

Add-on Instruction

Routine

New in Version 24

Time
Putting It All Together

Easily Deploy Library Content

Easier to Locate

Maintains Schedule
What does it look like?

- Graphical representation of the area. Controllers, I/O, Switches, HMs.
- Organize Multi-Controller & Multi-HMI Projects in one place.
- Configure Server to Controller Network Paths and RSLinx Enterprise Shortcuts.
- Hardware Library of Controllers, I/O, Switches, HMs, and Computers.
- User Libraries include AOI, UDT and HMI content.
- Produce/Consume Tag list.
- Find Results Log.
- I/O Alias tag List.
Key Points...
FactoryTalk View 8.1 integrated with Architect

Studio 5000 Architect

FactoryTalk View Studio
Key Points…
IAB export to RSLogix Architect

- Reduces design time by matching the BOM with the actual project
- Allows for the export of control systems created in Integrated Architecture Builder to Architect and the automatic creation of the corresponding Logix Designer controller project files
- Eliminates data entry redundancy
- Reduces design time by matching the BOM with the actual project
Key Points
EPLAN Bi-Directional synchronization

Synchronizes data transfer of Hardware configuration and Point Tags (Alias Tags)
Allows for automated schematic updates in EPLAN
Improves start-up time ensuring electrical drawings are perfectly matched to I/O assignments
Creating Logix and View Programs

Studio 5000 Architect

- Create Logix and FactoryTalk View Programs
Adding library content to projects

Studio 5000 Architect

- Add content from the Rockwell Automation Library of Process Objects
**DESIGN**
- Library Designer is integrated into Logix Designer
- Flexible library authoring, structuring reusable content the way you want

**PUBLISH**
- Combine control, visualization, historical and alarm management into one library object
- Centralized repository for easy access and version management

**BUILD**
- Easily generate your standardized application code
- Bulk configure projects
- Leverage Rockwell Automation provided libraries (Process)

Maximize reuse and quickly build projects
Design & Publish Library
Static content transforms into dynamic library content

Step 1:
Create Content to be libraried

Step 2:
Group and Parametrize Library Content

Step 3:
Publish
## Build Project Content
Easily deploy project content through configuration – No Programming!

### Step 1:
Select objects from library

- V1.0
- V1.1
- V2.0
- V1.0
- V1.1
- V2.0

### Step 2:
Configure via Parameters

**Analog Input Channel Object**
- Name: PT100
- Description: Ambient pressure
- Units: bar
- Minimum
- Maximum: 10

**Motor Control Object**
- Name: M101
- Description: AC motor
- Motor Type:
  - Single
  - Reversing
  - Two Speed
  - Hand Operated
- Electronic Overload:
  - YES
  - NO

### Step 3:
Generate

![Diagram showing configuration process]

---

**ACM Motors**
- M101
- M102
- M103
- M104

**Frequency Table**

- **Parameter**
  - **Value**
  - **Unit**

**Input Values**

- **Value**
  - **Unit**

---

**Build Project Content**
Easily deploy project content through configuration – No Programming!
These are all great features! But where do I start?
Lets break it down

- **System:** Mixing Tank
- **What does it do?**
- **Functions:** Add/Remove/Mix/Heat/Cool
- **How does it do those?**
- **Controls components:** valves/motors/steam/chilled water
  - Valve block. Options, control only? Include limit switches? Encapsulate alarms?
    - Type of valve? Analog? 2 position? Spring return? How long to open?
  - Steam. Just a few valve blocks and a temp control loop?
  - Chilled water. See above

- What about level control?
To AOI or not to AOI?

- AOI’s the good, the bad, and the lack of spaghetti.
  - + Have their own Data type
  - + Can have enable in false code (does a one shot work with out this?)
  - - AOI can not be modified on line.
  - + Revision history
  - + Can be locked down
  - + Great for small code
  - - Can become unruly at large sizes
AOI/AOP Tag Types

- Input
- Output
- InOut
- Local
- Public Parameter (AOP only)

- How does each behave scan to scan?
- If your program acts as a UDT, do you still need a UDT?
How to interconnect programs

- Options:
  - Link tags directly in code
  - Link tags in parameter list
    - Link inputs
    - Link outputs
  - Have mapping code
  - How do you handle extra code between programs?
    - Interlocks, alarms, other

- Input
- Output
- Inout
- Local
- Public Parameter
Modules, So Many Modules
New Ways to Access the Module Object

What's the Module Object?

- The Module Object provides status information relevant to a module

Attributes –
- Entry Status
- Fault Code
- Fault Info
- FW Supervisor Status
- Force Status
- INSTANCE
- Status Indicator
- Mode

Path

FYI
That's New

The Module

How to Access

The Info

Get System Value
Class Name
Instance Name
Attribute Name
Dest

Module
Lcl_S02
EntryStatus
Lcl_S02_EntrySts
6182
New Ways to Access the Module Object

New Attribute to the Module Object: Path

- Use GSV to retrieve the Module Communication Path
  - SINT Array

- Use an SSV to Set the Msg Path

- Possible Re-use of Message for Multiple modules in an Add-On Instruction
  - Coding required
Access to Module Object
Inside an Add-On Instruction

- Full Access to the Module Object inside the Add-On Instruction
  - “Module” Data type as InOut Parameter (Add-On Instruction & Programs)
  - Allows you to point to the module from within an Add-On Instruction Definition

I/O Configuration
- 1756 Backplane, 1756-A7
  - 0 [0] 1756-L72S LIBRARY
  - 1 [1] 1756-L75P LIBRARY:Partner
  - 2 [2] 1756-JB16 Lcl_S02
  - 3 [3] 1756-OB16Lc1_S03
  - 4 [4] 1756-JF0 Lcl_S04
  - 5 [5] 1756-EN8TR Lcl_S05

Add-On Instruction Definition

Full Access
Inhibit a Module
Configure a module in programming

- Use SSV to change module mode to 4 to inhibit.
- Set back to zero to un-inhibit
- Use the card_name.c.tags to change the card configuration.
  - Changes take effect on
    - power cycle
    - inhibit-un-inhibit.
    - Message instruction of type “module reconfigure”
Don’t be Alarmed
Traditional Alarming Method

This is polled data, consuming valuable network bandwidth.

Traditional alarming uses a lot of network bandwidth, as each client needs to talk to the data server.

Time stamps are relative to each individual clock in this case.

This is polled data, consuming valuable network bandwidth.

Each individual HMI is “acking” their own alarms, an acked alarm in one doesn’t ack in another.

All of these extra connections to the controller limit availability.
Logix-Based Alarms & Events Method

HMI clients subscribe to alarm events

When alarms are detected an A&E controller event is pushed to FactoryTalk® Linx

FactoryTalk® Linx data server hosts A&E component to subscribe to controller-based alarms

The FactoryTalk® Alarm & Events pushes alarm notifications to the HMI clients
Server Based Alarms

- Managed in FactoryTalk® View Studio
- Designed for non-Rockwell Automation & legacy controllers
- Step forward from RSView®32 or legacy HMI Alarms
Logix Based Alarms

Instruction Based Alarms

Tag Based Alarms

Version 16
Except v21 – 23

Version 31
5580, 5480, 5380 Only
Logix Instruction Based Alarms

- Managed in LogixDesigner
- Designed for exclusively for Rockwell Automation controller.
- Zero HMI design required, it just works!
This is Instruction Alarm example only...

Rhonda Benoit, 4/4/2018
Why use Instruction Alarms?
Logix Instruction Based Alarms

**Advantages**

- Alarms are detected at the same time logic is executed
- Alarms events are buffered in controller memory
- Alarms events are pushed to the HMI only on state changes
## Value of Logix Instruction Based Alarms

### Use Cases

<table>
<thead>
<tr>
<th>USE CASE</th>
<th>Alarming Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>I want to use alarms with <strong>legacy</strong> or <strong>third-party</strong> devices</td>
<td><strong>Server</strong> Tag Based Alarms</td>
</tr>
<tr>
<td>I want <strong>maximum scan time</strong> performance from my controller</td>
<td><strong>Server</strong> Tag Based Alarms</td>
</tr>
<tr>
<td>I want to integrate a <strong>skid or new controller</strong> with new alarms into an existing system <strong>without modifying the HMI</strong></td>
<td><strong>Logix</strong> Instruction Based Alarms</td>
</tr>
<tr>
<td>I want time stamps <strong>applied by the controller</strong></td>
<td><strong>Logix</strong> Instruction Based Alarms</td>
</tr>
</tbody>
</table>
## Challenges - Instruction Based Alarms

### Cons

<table>
<thead>
<tr>
<th></th>
<th>Logix Instruction Based Alarms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Time</td>
<td>Impacts scan time</td>
</tr>
<tr>
<td>Footprint</td>
<td>Uses a portion of controller memory</td>
</tr>
<tr>
<td>Modularity</td>
<td>Not possible</td>
</tr>
<tr>
<td>Grouping Roll-Ups</td>
<td>Not possible w/o programming</td>
</tr>
<tr>
<td>Bulk Generation</td>
<td>Challenging</td>
</tr>
</tbody>
</table>
Logix Tag-based Alarming

Simple alarm setup and configuration

- Single environment – deploy alarms when creating control elements in Studio 5000 Logix Designer®
- One place to go to add and configure alarms
- One click to add alarms from the objects definition
- Pre defined alarms on every object
- Allow users to select alarms for each instance.

- Available in the Logix 5380, 5480 & 5580 Family of Controllers – V31

Benefits

- Seamless integration to FTAE system and HMI display controls
- No additional software configuration
Logix Tag-Based Alarms

- FactoryTalk® View SE
- FactoryTalk® ViewPoint
- FactoryTalk® Alarms & Events
- FactoryTalk® Linx™

Supported in 5580, 5480 and 5380 Only

Tag Alarm Syntax

FactoryTalk® Linx™ extended to support direct access to alarm data members

- To configure user selects base reference and manually types member names (future tag browser extension)
- Access to alarm and alarm set values (shelve, disable, state and configuration)

* Not supported with FactoryTalk® View ME / PanelView® Plus or FactoryTalk® Linx Gateway
Simplify, remove FTInx emphasis
Rhonda Benoit, 4/4/2018
Logix Tag-Based Alarms

Adding an Alarm to Any Structure of tags – Ex. UDTs and AOIs

Every structure instance will now have an alarm. Alarm instances can now be customized!
New Alarming Capability
Logix Tag Based Alarms

- Alarms now defined on “Tags” with periodic evaluation
- Simplified design workflows inside Studio 5000®
- Standards-based design
- Leverages existing FactoryTalk® Alarms & Events infrastructure

Supported in 5580, 5480 and 5380 Only

PanelView™ 5500
With v5 in 2018
Minimize controller scan time impact using **multicore** Technology

- Alarm evaluation is done separately.
- Uses the same tag to evaluate alarm conditions
Instruction based alarms take a larger portion of a controller’s runtime memory.

With Logix tag based alarms, the average size of each alarm is significantly smaller:
- Thus reducing alarms’ footprint on controller memory.

- 1700 Bytes (ALMD)
- 1220 Bytes (ALMA/4)
- 850 Bytes (Logix Tag-Based Alarm)
Modularity via “Alarm Definition.”

- Alarm definitions reside on larger programming constructs like AOIs and UDTs.
- Unlike individual tags, alarm definitions allow you to create a set of alarms that are defined for all instances of that object.

Alarm Definitions can be placed on:
- User-defined data types
- Add-On Instructions
- System-defined data types (such as PIDE)
- Module-defined data types (such as PowerFlex®)
Faster Time to Market
Bulk Alarm Generation

- Simple Import/Export operations to XML
  - Includes SCHEMA for easy interpretation by a variety of packages
  - Targets Microsoft Excel for easy manipulation in bulk.

<table>
<thead>
<tr>
<th>Name</th>
<th>ConditionType</th>
<th>Event</th>
<th>Expression</th>
<th>Limit</th>
<th>TargetTag</th>
<th>OriginTag</th>
<th>Severity</th>
<th>AssociatedTag</th>
<th>AssociatedTag?</th>
<th>AssociatedTag2</th>
<th>AssociatedTag3</th>
<th>AssociatedTag4</th>
<th>AssociatedTag5</th>
<th>AssociatedTag6</th>
<th>AssociatedTag7</th>
<th>AssociatedTag8</th>
<th>AssociatedTag9</th>
<th>AssociatedTag10</th>
<th>AssociatedTag11</th>
<th>AssociatedTag12</th>
</tr>
</thead>
<tbody>
<tr>
<td>HHH</td>
<td>TRIP</td>
<td>my_boot</td>
<td>&lt;= 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>idfaf</td>
<td>TRIP</td>
<td>udsaf1.</td>
<td>&lt;= 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HHH</td>
<td>HE</td>
<td>MainProgram:instance1:Input</td>
<td>&lt;= 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>idfaf</td>
<td>HE</td>
<td>MainProgram:instance1:Input</td>
<td>&lt;= 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HHH</td>
<td>HE</td>
<td>MainProgram:instance1:Input</td>
<td>&lt;= 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>idfaf</td>
<td>HE</td>
<td>MainProgram:instance1:Input</td>
<td>&lt;= 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GOS</td>
<td>TRIP</td>
<td>OOTService</td>
<td>&lt;= 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HHH</td>
<td>HHH</td>
<td>MainProgram:cows</td>
<td>&lt;= 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Info</td>
<td>HE</td>
<td>MainProgram:cows</td>
<td>&lt;= 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Info</td>
<td>HE</td>
<td>MainProgram:instance1:Output</td>
<td>&lt;= 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Info</td>
<td>ME</td>
<td>MainProgram:instance1:Output</td>
<td>&lt;= 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HHH</td>
<td>HHH</td>
<td>MainProgram:cows</td>
<td>&lt;= 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Info</td>
<td>ME</td>
<td>MainProgram:instance1:Output</td>
<td>&lt;= 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Info</td>
<td>ME</td>
<td>MainProgram:instance1:Output</td>
<td>&lt;= 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Info</td>
<td>ME</td>
<td>MainProgram:instance1:Output</td>
<td>&lt;= 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Info</td>
<td>ME</td>
<td>MainProgram:instance1:Output</td>
<td>&lt;= 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Editor Improvements: Structured Text

Overview

- Level Setting Usability
  - Zoom In & Out
  - Bookmarks

- Enhancements
  - Line Numbers
  - More Descriptive Tool Tips
  - Syntax Highlighting
  - Multi-Line / Column Select (Alt + Mouse)
  - Change and Verify Bars
  - Code Outlining (collapsible)
  - Completion Prompt
  - Code Snippets
  - Smart Indent
  - Inline Value monitoring
Logix Designer®: Structured Text Editor
Enhancement - Line Numbers

Line Numbers

Workstation Options
Categories:

- Application
- Display
- Verification
- Tag Editor/Data Monitor
- Equipment Sequence Editor
- Ladder Display
- SFC Editor
- FBD Editor
- Structured Text Editor

Change Structured Text Editor Preferences
- Show Line Numbers
- Semantic Outlining
- Show Simplification

Inline Values
- Do Not Show
- Show When Monitoring Online
- Show Alway

Indentation
- None
- Block
- Smart

Tabs
- Tab Size: 4
- Insert Spaces
- Keep Tabs
- Include Header on Create

Copyright © 2016 Rockwell Automation, Inc. All Rights Reserved.

```plaintext
if (xuTest_Mode = 3) then
  ![Structured Text Code]
endif
```

PUBLIC
Logix Designer®: Structured Text Editor
Enhancement - Syntax Highlighting

- Syntax Highlighting

- Before

```
New_InputParameter1:= My_NewLocal;
New_OutputParameter1 := MyOtherNewLocal;

new_Tag1:= My_NewLocal;
```

- After

```
22 New_InputParameter1:= My_NewLocal;
23 New_OutputParameter1 := MyOtherNewLocal;
24
25 new_Tag1:= My_NewLocal;
26
```

*Quick & Clear identification of undefined tags and incorrect syntax*
More descriptive Tool Tips

```
ST_Degrees := DEG(Radians);
ST_Radians := RAD(Degrees);
/* ST ASCII Conversion */
```

```
FIND(last, search_string, sint, num1);
FIND(source, search_string, start, result);
MID(search_string, sint1, sint2, dest_string);
DELETE(first, sint1, sint2, dest_string)
```
Logix Designer®: Structured Text Editor
Enhancement – Completion Prompt

- Completion Prompt
  - Tag Look Ahead (i.e Type “Prod”)
Intelligent Completion Prompt

- “Type” based Filtering
Logix Designer®: Structured Text Editor
Enhancement – Change Bars

- Change and Verify Bars

```plaintext
yyTstM := 1;
If yyW2 Then
    yyI := 0;
    yySample_W2 := 0;
    yyTest_Spd_Ref := xxTest_Spd_Set / xxConstant_RPMPerFPM * xxBuildUpRatio2;
End_If;
```

- Changes detected

```plaintext
yyTstM := 1;
If yyW2 Then
    yyI := 1;
    yySample_W2 := 1;
    yyTest_Spd_Ref := xxTest_Spd_Set / xxConstant_RPMPerFPM * xxBuildUpRatio2;
End_If;
```

- Verification of changes

```plaintext
yyTstM := 1;
If yyW2 Then
    yyI := 1;
    yySample_W2 := 1;
    yyTest_Spd_Ref := xxTest_Spd_Set / xxConstant_RPMPerFPM * xxBuildUpRatio2;
End_If;
```

Copyright © 2016 Rockwell Automation, Inc. All Rights Reserved.
Logix Designer®: Structured Text Editor
Enhancement – Code Snippets

- **What does a Code snippet do for me?**
  - Inserts a Predefined Syntax with template parameters
  - Reduces the amount of work for entering *commonly used constructs*

- **Supported Constructs in Logix**
  - IF... THEN
  - ELSEIF... THEN
  - CASE... OF
  - FOR... DO
  - WHILE... DO
  - REPEAT... UNTIL
How does this work?

- Type in a Key word like `for`
  
  - Press ‘Tab’ Key.
    - The associated code snippet will be inserted with placeholders
    - Hover Tool Tips provide placeholder information

- More productive Structured Text design experience

```plaintext
for count := initial_value to final_value do
  end_for;

for count := initial_value to final_value do
  Tag to store count position as the FOR...DO executes
  end_for;
```
Logix Designer®: Structured Text Editor
Enhancement – Code Outlining

- Collapsible Code Segments
- Tool Tips show collapsed content
- Nesting of Constructs is also supported
Tool Tip Outline

- Cursor over the various Collapsible Indicators
- Clear indication of the construct & comment segmentations.

```plaintext
IF (xxTest_Mode=1) Then
  ***************
  * Warming up   *
  * TEST_MODE1% = 1 *
  *******************
  yyTstM := 1;
  IF xxTest THEN
    yyTest_Spd_Ref := xxTest_Spd_Set / xxConstant_RPMPerFPM * xxBuildUpRatio2;
    Else
    yyTest_Spd_Ref := 0;
  End_IF;
End_IF;
```
Logix Designer®: Structured Text Editor
Enhancement – User Defined Code Segments

- User defined region of code

- Syntax
  - `#region`
  - `#endregion`
  - Documentation immediately following that describes the region

- Collapsed Content

- Tool-tip Contents
Zoom In and Out capability

50-300%
Logix Designer®: Structured Text Editor
Enhancement – Multi-Line Column Select

- **Multi-Line Select (Alt-Mouse)**

```plaintext
If xxTest then
  yyTest_Spd := xxLineSpdRef_FPM * xxConstant_BMPPerFPM / xxBuildUpRatio1;
  If xxTest_Spd < xxHSLS_TBL_Spd[0] then yyHSLS_TBL[0] := xxAveraged_Trq_PU - yyHSLS_TBL_Loss[0]; End_IF;
  If xxTest_Spd < xxHSLS_TBL_Spd[1] then yyHSLS_TBL[1] := xxAveraged_Trq_PU - yyHSLS_TBL_Loss[1]; End_IF;
  If xxTest_Spd < xxHSLS_TBL_Spd[8] then yyHSLS_TBL[8] := xxAveraged_Trq_PU - yyHSLS_TBL_Loss[8]; End_IF;
  If xxTest_Spd < xxHSLS_TBL_Spd[10] then yyHSLS_TBL[10] := xxAveraged_Trq_PU - yyHSLS_TBL_Loss[10]; End_IF;
  If xxTest_Spd < xxHSLS_TBL_Spd[12] then yyHSLS_TBL[12] := xxAveraged_Trq_PU - yyHSLS_TBL_Loss[12]; End_IF;
  If xxTest_Spd < xxHSLS_TBL_Spd[14] then yyHSLS_TBL[14] := xxAveraged_Trq_PU - yyHSLS_TBL_Loss[14]; End_IF;
  If xxTest_Spd < xxHSLS_TBL_Spd[16] then yyHSLS_TBL[16] := xxAveraged_Trq_PU - yyHSLS_TBL_Loss[16]; End_IF;
End_IF;
```

- Allows users to quickly change a column of code without multiple edits.
Logix Designer®: Structured Text Editor
Enhancement – Inline Value Monitoring

- Inline Value Monitoring
  - 3 Display Options
    - Manual Toggle On/Off
    - When Monitoring Online
    - Show Always
  - Editable Value Fields
    - Same as RLL, FBD
Logix Designer®: Structured Text Editor
Enhancement – Smart Indent

- Smart Indent <Good Practice>
  - The result is clear indication of the statements and the constructs for the various levels of your code.
  - Indentation Style in Workstation Options

```plaintext
4 if MyBoolTag then
  x:=1;
5  elseif Mybooltag2 then
6      y:=6;
7      z:=8;
8    else
9      a:=12;
10   end_if;
11```

![Indentation Style in Workstation Options](image.png)
Smart Indent <Good Practice>

- Help provide a cleaner looking piece of code by automatically understanding constructs and indenting when nesting them.
- Type a Construct Expression press Enter
  - Editor recognizes Construct and Auto Indents
  - This sets you up for properly indented code.
  - When you are done with your statement and you end it with the `else` or `elsif`, the editor will actually outdent for you. This aligns your statements.
What does a Code snippet do for me?

- Inserts a Predefined Syntax with template parameters
- Reduces the amount of work for entering *commonly used constructs*

Supported Constructs in Logix

- IF... THEN
- ELSEIF... THEN
- CASE... OF
- FOR... DO
- WHILE... DO
- REPEAT.. UNTIL
Rounding error
Look at the bits, they’re all over the floor
Limited Precision Example 1 - Addition
If an ADD instruction is used to add 9,000,000 (nine million) to 1,234.56789, the result should be 9,001,234.56789. However, due to the limited precision of the float format (6 digits), the result for a ControlLogix is 9,001,232. In this example, the first 6 digits match, but due to rounding errors of any further operations, only the first five digits of the number should be considered reliable.

Limited Precision Example 2 - Addition
Another example would be adding 1 to 9,000,000. The result is 9,000,000. In this case, the number being added is small enough that it doesn't have any impact at all on the result.

Limited Precision Example 3 - Multiplication
The same holds true for the result of a multiply. An example for a multiply is 123,456,789 times .123456789. This would equal 15241578.75 on a 10 digit calculator, but the controller will show 15241579.2.
Extended Data Types
64 Bit Math

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>BOOL</td>
</tr>
<tr>
<td>Short Integer</td>
<td>SINT</td>
</tr>
<tr>
<td>Integer</td>
<td>INT</td>
</tr>
<tr>
<td>Double Integer</td>
<td>DINT</td>
</tr>
<tr>
<td>Real Number</td>
<td>REAL</td>
</tr>
<tr>
<td>Unsigned Short Integer</td>
<td>USINT</td>
</tr>
<tr>
<td>Unsigned Double Integer</td>
<td>UDINT</td>
</tr>
<tr>
<td>Unsigned Integer</td>
<td>UINT</td>
</tr>
<tr>
<td>Unsigned Long Integer</td>
<td>ULLINT</td>
</tr>
<tr>
<td>Long Real Number</td>
<td>LREAL</td>
</tr>
<tr>
<td>Long Integer</td>
<td>LINT</td>
</tr>
</tbody>
</table>

Overview
- New extended data types in CompactLogix™ 5380 and ControlLogix® 5580 controllers to calculate more precise or larger values for use in your calculations

Benefits
- Calculate more precise values with higher resolution
- More flexibility to comply with open/third-party communications standards (e.g. HART)
LReal Limitations
15 is all you get

With 15 significant digits
LREAL reduces inadvertent error
Totalizer
Minimal error
Demo Time
They don't make enough duct tape to fix this.
Thank You