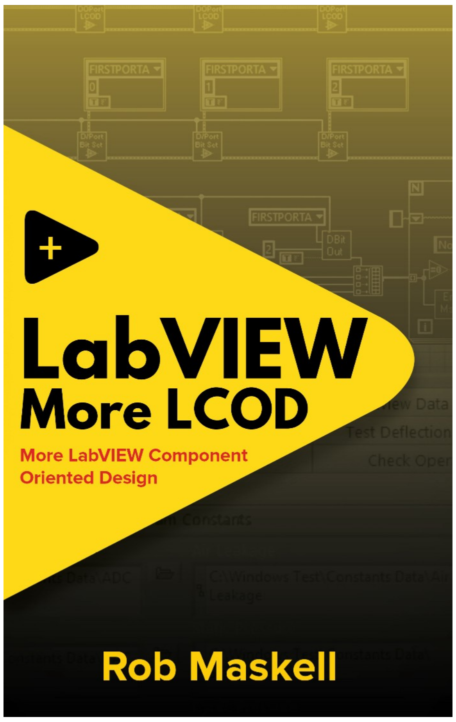
2024 Expansion Notes for the book LabVIEW:- more LCOD

First of all a big “thank you” for reading my latest book “LabVIEW :- more LCOD”.

If you haven’t read the book yet, I just wanted to let you know a little about it.

This book is published at the end of a very long career of 50 + years.



The LabVIEW : more LCOD Name is a reflection of what is in the book and it covers Component Building using State engines. That is while loops with shift registers, named cases, the case states controlled by type definition enums running off the shift registers.

No where in the book do I deviate from the State Engine Component design and so there are no flat sequence programming examples shown, as this was the way that LabVIEW was programmed when it first came out. I invented the first state engines using the technique above in 1988. The first state engine was built for a production engineer from the Jaguar car company at the first MAC World show in London using the MAC LabVIEW 1 interpreter version.

Actually the first state engine was driven by numerics, later by strings and finally with enums.

The main purpose of the book is to show off the four main design criteria for any software development. (Cohesion, Coupling, Abstraction and Hidden Data).

Cohesion and Coupling often go hand in hand in most programs and the use of private data and private components solve most of these issues.

Abstraction is covered in a lot of detail and a method called double abstraction is very useful. State Engines as Components is show in detail examples.

The book focuses largely on “Hidden Data” techniques and shows off a new method to remove any constant type from inside a diagram and store them on disk as their graphical objects. This is in contrast to the very limited ini file method which allows storing only numerics, strings, booleans and arrays.   Recovery of these graphical constant items is shown and how to re-embed the data back into the diagram.

This is avery big step forward for program management. Data in the graphical files can also be edited by an off line in line Editor.

In addition to these programming chapters are several chapters showing off the technology I have played with over the many years.

The first chapter on Technology history going right back into the early 1950s and the first use of transistors. It covers a little on radio back then and early computers both analog and digital versions that I used and developed on.

The second chapter on technology covers digital computers from the 1960/70s. By the time I was 20 years old I was developing a type of virtual instrument system for Biomedical research, building the hardware and software for the first real time metabolic analyser for athletes and research. I called them multivariable instruments. Plus other projects.

There is a chapter on technology covers early DEC mini computers and the early OSs they used.

Also a chapter on technology covers my time when I started the NI in Europe, starting in the UK and finishing in Italy. Also looking at the various advances made in LabVIEW methods and uses during that time. This is pre LabVIEW 2.

A short chapter is devoted to Calculator technology. When I started working there were no calculators, we used Slide rules for all our design work. Calculators were the stepping stones to the development of the micro processors.

The last technology chapter is devoted to a few interesting projects under taken from the 60s to the 80s.

The book is finished off with a method showing how to insert algorithms into LabVIEW diagrams by using the same technique as hidden data.

The biggest difficulty in making a paper back book like this is the resolution of the printing and is it possible to show the Front Panels and Diagrams well enough to read easily. I spent quite a bit of time on this and I think I managed this. Changing the default font size on all examples was the key step.

So I hope you enjoy the book even if you do not use all that is in it. It is quite a step for many programmers to make.

The next few notes and examples in this PDF show off further development of the book’s original ideas.

For many beginners this info may be a challenge or not understand why as they have not yet hit LabVIEW development issues with large programs but hang in there if you are a beginner.

What is offered may be helpful or not depending on how you view programming.

Since the release of this book last year, I have been using the LCOD COC programming method in a new “Motorcycle Engine Performance Analysis” software & hardware project.

While working away, I started reflecting a little more on the LCOD’s four design features: Hidden Data, Abstraction, Cohesion and Coupling.

I have now come to the realisation that I may achieved a little more than I explained in the book, hence the need for additional notes.

This realisation are mainly focusing on expanding the “Hidden Data” ideas & methods explained in the book.

Plus a smattering of extra Abstraction and extra Self documentation help also.

This 2024 PDF will show off these new awareness points and you can decide if these could be helpful also.

I am hoping to make these additional PDF notes into Web pages at some point for the site www.lcodcoc.com

Additional seeds for this new awareness were also realised when I started looking over some old C Code written in the early 90s that I was using in a new project.

The seed thought that popped out was based around this realisation.

C program design always requires the building of Data Declaration blocks and Definition blocks for Functions that will be used in the main code.

I must declare though, that I am no expert in C.

In almost all coded programming languages before you can start writing the App’s code you have to make Data Declaration and Definition blocks for: variables and constants that will be used in the code and assign a data type (Float, Int, Array, String, Logic etc). Plus any functions to be made.

The compilers has to have this info to make sense of all the main body code to be able to compile to machine code.

This declaration and definition work also becomes part of the programs early design and usually the programmer/s are backwards and forwards to this declaration and definition sections of code adding more info as they progress deeper into the coding. It is often iterative and not something that is complete right at the beginning of coding.

Just to illustrate this I will share some different coding definition and declaration methods just in case you are not aware of these requirements in most coded languages.

Starting with an example using Basic A, the original interpreter Basic for the IBM PC (1981).

This IBM PC Basic A code is from about 1984 and is an example of an Opto22 OptoMUX programs declaration block.

```
1220 '*********************************************************************
1230 '*
1240 '*     DIMENSION AND INITIALIZE VARIABLES
1250 '*
1260 '*********************************************************************
1270 '                  This work is achieved with a Subroutine call to line 1300
1280 '
1290 '                  Initialize OPTOMUX Driver Parameters
1300                    ADDR% = 255
1310                    COMMAND% =0
1320                    DIMPOSI%(15),MODI%(1),INFO%(15)
1330  '
1340                    FOR I% =0 TO 15
1350                    POS I%(I%) = 0
```

```
1360            INFO%(I%) = 0
1370            NEXT
1380 '
1330            MODI%(0) = 0
1400            MODI%(1) = 0
1410 '
1420 '          Define Address Of OPTOMUX Driver (Optoware)
1430
1440            OPTOWARE = 4
1450            RETURN
1460 '
1470 '
1480 '*********************************************************************
```

Not much is required really as Basic A is sort of like LabVIEW in that it allows on the fly or dynamic making of variables etc.

Next:- this is the 90's C code that also seeded my mind and it shows the definition and declaration blocks used in an embedded micro program we built to automate a "commercial fishing boat's set and recovery long line winch systems". It is quite comprehensive with a lot of details.

```
#define BOARD_TYPE    CPLC_BOARD

#define RXMAX         1024
#define TXMAX         1024
#define MODE          0x14   // 1 stop, no parity, 8 data, even, RTS/CTS
#define BAUD          19200
#define MODEM         1
#define ECHO          1
#define LCD_COLS      20
#define LCD_LINES     2

// function prototypes
void InitProfile(void);
void ProcMessage(void);
void NakMess(void);
void AckMess(void);
void PerformMess(void);
void DownloadProfile(void);
void JigHistory(void);
void JigDepth(void);
void BeginJig(void);
void PauseJig(void);
void RaiseLine(void);
void ContinueJig(void);
void StopJig(void);
void Direction(char direction);
void MotorSpeed(int speed);
int MotorCurrent(void);
void StartCounters(void);
void SetAlarm(char state);
void ClearAlarm(void);
void NewAddress(void);
int Extract(char **pch);
void LCDMessage(char *message);
void LCDString(char *message);
void ShowDepth(void);

// status
#define READY         0
#define LOWERLINE     1
#define JIGGING               2
#define RAISENOLOAD   3
#define RAISELOAD     4
#define PAUSE         5
#define ALARM         6

// direction
#define OFFDIR        0
#define FWDDIR        1
#define REVDIR        2

// alarm
#define ALARMOFF      0
```

```
#define ALARMON              1

// counter inputs
#define SPOOLCOUNT    0
#define MOTORCOUNT    1

// global variables
char ser0RxBuff[RXMAX];
char ser0TxBuff[TXMAX];
char ser0RxMess[RXMAX];
char *pMessage;
char messResp[TXMAX];
char error;

// costatements
CoData LowerLine;
CoData Jigging;
CoData RaiseNoLoad;
CoData RaiseLoad;
CoData LoadSense;
CoData CheckDepth;
CoData Slippage;

struct S_LOWERLINE
{
	int speed;
	int depth;
} lower;

struct S_RAISENOLOAD
{
	int speed;
	int endDepth;
} raiseNoLoad;

struct S_LOADSENSE
{
	int maxSlippage;
	int maxCurrent;
} loadSense;

struct S_JIGGING
{
	int downSpeed;
	int pauseTime;
	int downDistance;
	int maxDepth;
	int numCycles;
	int numDistances;
	int upDistance[101];
	int minDepth;
} jigging;

struct S_RAISELOAD
{
	int minSpeed;
	int maxSpeed;
	int startSpeedPerc;
	int speedIncPerc;
	int timePeriod;
	int stopDistance;
	int maxSlippage;
	int maxCurrent;
	int startSpeed;
	int speedInc;
} raiseLoad;

struct S_RIG
{
	int motorDistance;
	int spoolDistance;
	int minMotorCount;
} rig;

char status;
char start[5];
char direction;

int speed;
int depth;
```

```
unsigned int depthCount;
unsigned int spoolCount;
unsigned int motorCount;
unsigned int slippage;
char paused;
int motorSpeed;
char testing;
int address;
int addressCom;
```

You can see this is quite comprehensive and is the basis of the program development from then on.

Also I noticed, much of the “design thought” for the program is actually in those Declarations and Definitions

Just read them through and you can almost imagine the program running in your mind. (This is a very good thing, I feel)

However it is not graphical.

People often say LabVIEW is Graphical C however it has a lot of look and feel like the Basic A Interpreter also. Interactive builds, dynamic adding of variables and constants without declaration and easy definitions. It even makes dynamically dimensioned arrays.

Because LabVIEW doesn’t require all the Definitions and Declarations up front. This allows the programmer to ignore this part of building programs and to just charging straight into building the graphical diagrams. This may be a nice feature but I feel it misses out an essential part of the design and preparation cycle in the program build. The program can become haphazard in nature.

You will find this is very true when you start building large LV programs with many components. The free style method of LabVIEW programming usually starts to become a messy and it makes maintaining programs difficult, as all the defining and declaring is scattered through out the diagrams.

So back to the LabVIEW:- more LCOD book and this new realisation.

Looking inside the LabVIEW:- more LCOD book, towards the end of the book I actually started morphing the Hidden Data techniques COCs to include much of this definition and declaration information listed above and even more.

So I have now come to the conclusion the LCOD COC method can be also be said to be a type of methodology for LabVIEW Data Declaration and Function Definition Blocks.

At the end of the book the Hidden Data method was the morphed or stretched to include more than just the public COCs > file paths, operator display interface configuration, DAQ, DAC, DIO, Encoder, Sensors, Algorithms, Calibration, Messages, Copyrights, To Do List, Running Log. All of these are Public Data. The last parts of the book included a lot on Abstraction, self documentation and finally how to control Component States from a COC file. The addendum had a section on Initialisation COCs.

The App’s Components can use the Public Data COCs plus the Components all have their own private data declaration in the Component “Diagram COCs”.

The Morphing that happened was many more declarations, even down to being able to choose for the Components, the State steps order and settings. This enabled the modifying of the state engine operation in an editor.

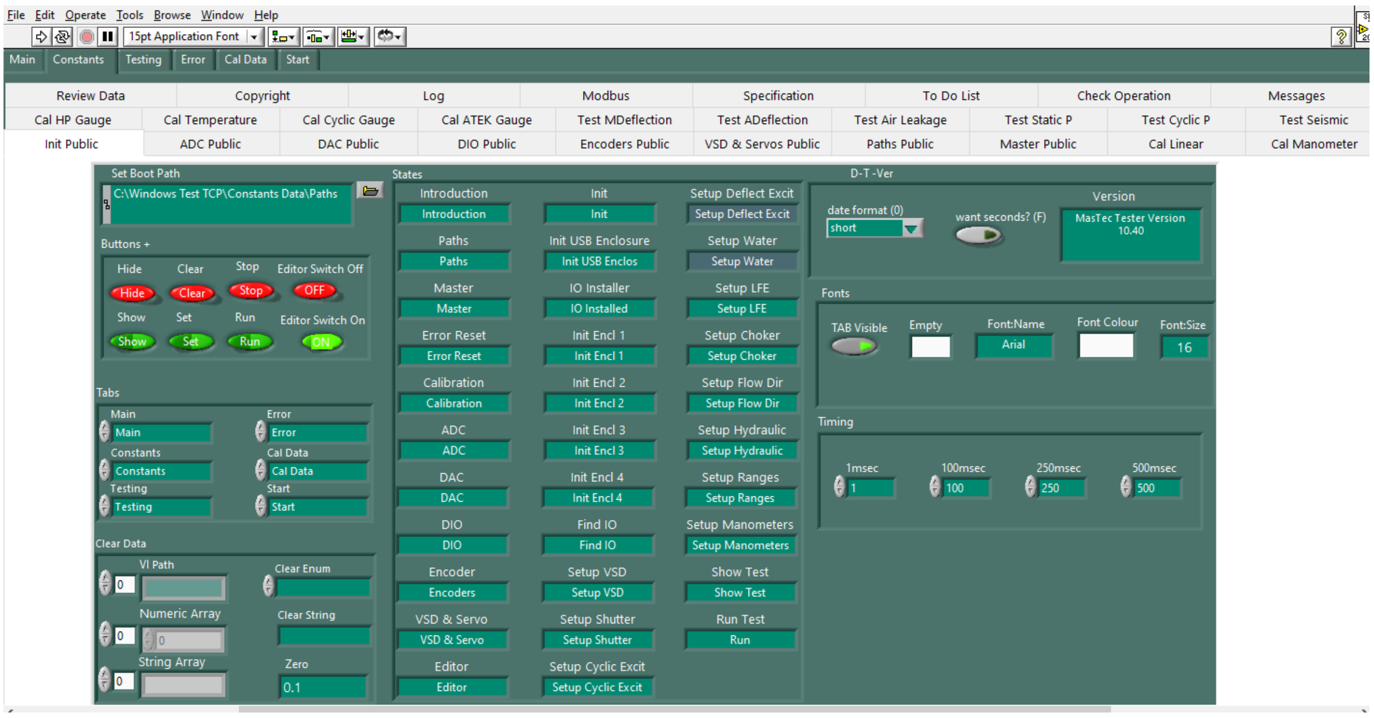
In this note I will add a new version of a “COC Data Editor” to manage the Declaration Data before the program runs or if the external editor is used even Edit while the program is running. This new Editor has a better component design and uses Event selection of states plus it is not hardwired and the Editor can be adjusted inside the Editor itself.

Over the next few pages, I will show these declaration idea more completely for you. I have also expanded the COCs, with a new “Init COC” to help in Initialisation all sections of the program correctly and managing a whole heap of house keeping and initialisation. Hence the COC name “Init”

The first section we will look at is the Main VI’s several Front Panels embedded in the Main TABs.

This first Pict is the Constants COC display, showing all the Public and Private COCs.

Notice the new “Init” COC is shown and there is also an Encoder” COCs.


File Edit Operate Tools Browse Window Help
15pt Application Font
Main Constants Testing Error Cal Data Start
Review Data Copyright Log Modbus Specification To Do List Check Operation Messages
Cal HP Gauge Cal Temperature Cal Cyclic Gauge Cal ATEK Gauge Test MDeflection Test ADeflection Test Air Leakage Test Static P Test Cyclic P Test Seismic
Init Public ADC Public DAC Public DIO Public Encoders Public VSD & Servos Public Paths Public Master Public Cal Linear Cal Manometer
Buttons +
Hide Clear Stop Editor Switch Off
Hide Clear Stop OFF
Show Set Run Editor Switch On
Show Set Run ON
Clear Data
VI Path
Clear Enum
Numeric Array
Clear String
String Array
States
D-T -Ver
Fonts
TAB Visible
Empty
Font Colour
Timing

Set Boot Path: C:\Windows Test TCP\Constants Data\Paths

Tabs
Main: Main
Error: Error
Constants: Constants
Cal Data: Cal Data
Testing: Testing
Start: Start

Clear Data: VI Path 0; Numeric Array 0, 0; String Array 0; Zero 0.1

States		
Introduction: Introduction	Init: Init	Setup Deflect Excit: Setup Deflect Excit
Paths: Paths	Init USB Enclosure: Init USB Enclos	Setup Water: Setup Water
Master: Master	IO Installer: IO Installed	Setup LFE: Setup LFE
Error Reset: Error Reset	Init Encl 1: Init Encl 1	Setup Choker: Setup Choker
Calibration: Calibration	Init Encl 2: Init Encl 2	Setup Flow Dir: Setup Flow Dir
ADC: ADC	Init Encl 3: Init Encl 3	Setup Hydraulic: Setup Hydraulic
DAC: DAC	Init Encl 4: Init Encl 4	Setup Ranges: Setup Ranges
DIO: DIO	Find IO: Find IO	Setup Manometers: Setup Manometers
Encoder: Encoders	Setup VSD: Setup VSD	Show Test: Show Test
VSD & Servo: VSD & Servo	Setup Shutter: Setup Shutter	Run Test: Run
Editor: Editor	Setup Cyclic Excit: Setup Cyclic Excit	

date format (0): short
want seconds? (F)
Version: MasTec Tester Version 10.40

Font:Name: Arial
Font:Size: 16

Timing: 1msec 1; 100msec 100; 250msec 250; 500msec 500

1/ The six Front Panels sit inside each of the the Main VI TABs.

The TABs are “Main, Constants, Testing, Error, Cal Data, Start”

You can have as many FP as you like, if you use Tabs to hide them away like this.

We will start by look inside the “Constants” Tab’s FP first

2/ The first Pict above shows the FP “Constants” and it is displaying the new “Init” COC.

This Init COC's clusters have a whole heap of useful initialisation data.

3/ The Init COC top left, has a control "Set Boot Path" which is a File Path Control that points to the apps "Paths" COC files. This is the Boot Path to start the program.

This starts the process of loading all the Hidden Data COCs back into the program and bringing up the App for use.

4/ The Init COC cluster "Button +" has useful set Boolean constants clear, set, hide, show, run, stop buttons + two buttons to allow the editor to run first if required.

Two Booleans for each state Show/Hide, Clear/Set, Stop/Run/, Editor On/OFF

5/ The Init COC cluster "Tabs" has all the Main VI Front Panel Tab names (Main, Constants, Testing, Error, Cal Data, Start). This will allow the program to select one of these tabs to show a new FP. The selection is via Tab properties control.

6/ The Init COC cluster "Clear Data" has some empty Controls to allow presetting. This can be extended as needed. Handy to have these to use in all Component initialisations.

7/ The Init COC cluster of Enum "States" has all the Main VI's States and these are all set to their particular named State.

You need one ENUM for each State.

8/ The Init COC cluster "D-T-Ver" sets date and time and version display.

9/ The Init COC Cluster "Fonts" allows setting the font type used and Font size.

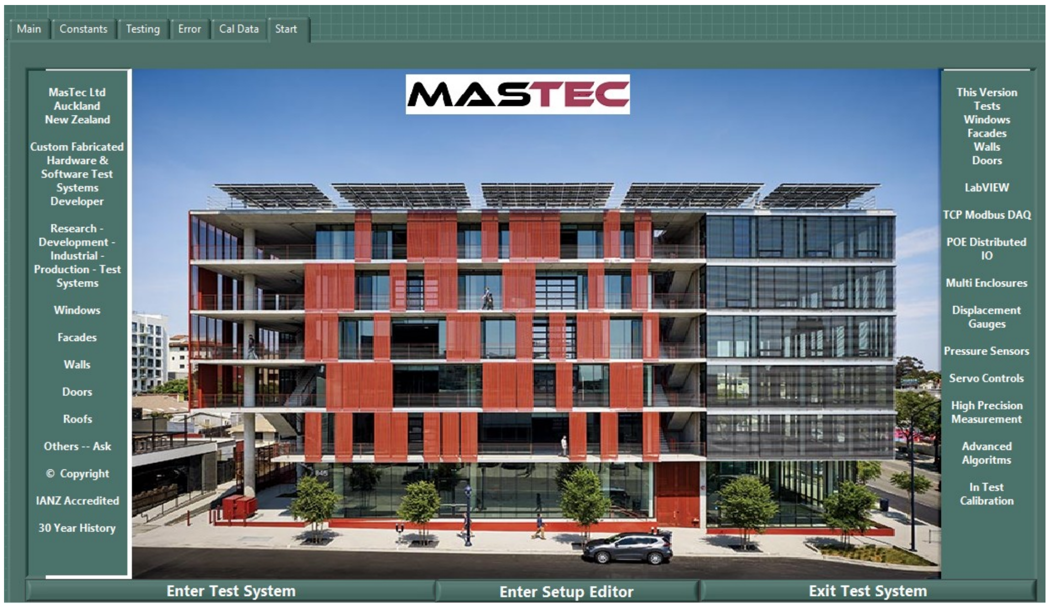
Again you can have many for different parts of the program etc.

10/ The Init COC cluster "Timing" is for loop control globally and very handy.

The Names in these numerics auto doc where they are used.

You decide what is important for your program by adding and subtracting from these Init clusters and also for the Main VI Tabs.

Next series of picts shows shows the other FPs hidden in the Main VI TAB. First the Start Screen



Main | Constants | Testing | Error | Cal Data | Start

MASTEC

MasTec Ltd
Auckland
New Zealand

Custom Fabricated Hardware & Software Test Systems Developer

Research - Development - Industrial - Production - Test Systems

Windows

Facades

Walls

Doors

Roofs

Others -- Ask

© Copyright

IANZ Accredited

30 Year History

This Version Tests
Windows
Facades
Walls
Doors

LabVIEW

TCP Modbus DAQ

POE Distributed IO

Multi Enclosures

Displacement Gauges

Pressure Sensors

Servo Controls

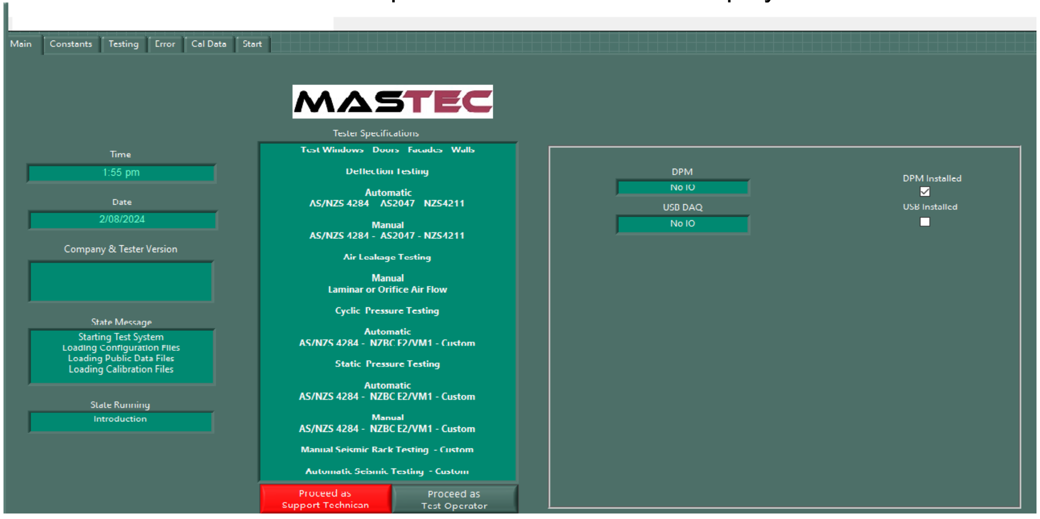
High Precision Measurement

Advanced Algoritms

In Test Calibration

Enter Test System | Enter Setup Editor | Exit Test System

The Main TAB Commission or Operator Select Screen:- Displays Site Hardware and IO available



Main | Constants | Testing | Error | Cal Data | Start

MASTEC

Time
1:55 pm

Date
2/08/2024

Company & Tester Version

State Message
Starting Test System
Loading Configuration Files
Loading Public Data Files
Loading Calibration Files

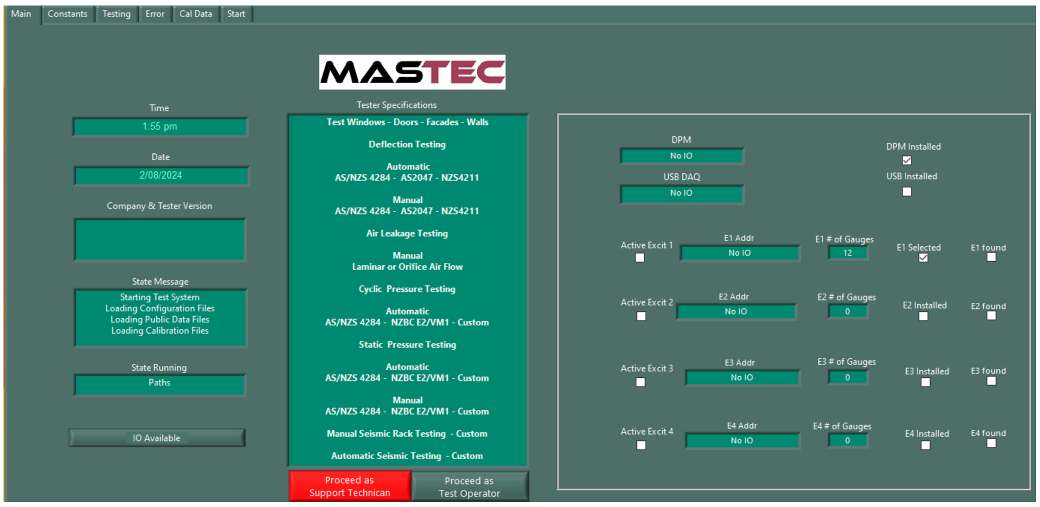
State Running
Introduction

Tester Specifications
Test Windows - Doors - Facades - Walls
Deflection Testing
Automatic
AS/NZS 4284 - AS2047 - NZS4211
Manual
AS/NZS 4284 - AS2047 - NZS4211
Air Leakage Testing
Manual
Laminar or Orifice Air Flow
Cyclic Pressure Testing
Automatic
AS/NZS 4284 - NZBC E2/VM1 - Custom
Static Pressure Testing
Automatic
AS/NZS 4284 - NZBC E2/VM1 - Custom
Manual
AS/NZS 4284 - NZBC E2/VM1 - Custom
Manual Seismic Rack Testing - Custom
Automatic Seismic Testing - Custom

Proceed as Support Technican | Proceed as Test Operator

DPM
No IO
USD DAQ
No IO

DPM Installed ✓
USB Installed ☐



Main | Constants | Testing | Error | Cal Data | Start

MASTEC

Time
1:55 pm

Date
2/08/2024

Company & Tester Version

State Message
Starting Test System
Loading Configuration Files
Loading Public Data Files
Loading Calibration Files

State Running
Paths

IO Available

Tester Specifications
Test Windows - Doors - Facades - Walls
Deflection Testing
Automatic
AS/NZS 4284 - AS2047 - NZS4211
Manual
AS/NZS 4284 - AS2047 - NZS4211
Air Leakage Testing
Manual
Laminar or Orifice Air Flow
Cyclic Pressure Testing
Automatic
AS/NZS 4284 - NZBC E2/VM1 - Custom
Static Pressure Testing
Automatic
AS/NZS 4284 - NZBC E2/VM1 - Custom
Manual
AS/NZS 4284 - NZBC E2/VM1 - Custom
Manual Seismic Rack Testing - Custom
Automatic Seismic Testing - Custom

Proceed as Support Technican | Proceed as Test Operator

DPM
No IO
USB DAQ
No IO

DPM Installed ✓
USB Installed ☐

Active Excit 1 ☐	E1 Addr No IO	E1 # of Gauges 12	E1 Selected ✓	E1 found ☐
Active Excit 2 ☐	E2 Addr No IO	E2 # of Gauges 0	E2 Installed ☐	E2 found ☐
Active Excit 3 ☐	E3 Addr No IO	E3 # of Gauges 0	E3 Installed ☐	E3 found ☐
Active Excit 4 ☐	E4 Addr No IO	E4 # of Gauges 0	E4 Installed ☐	E4 found ☐

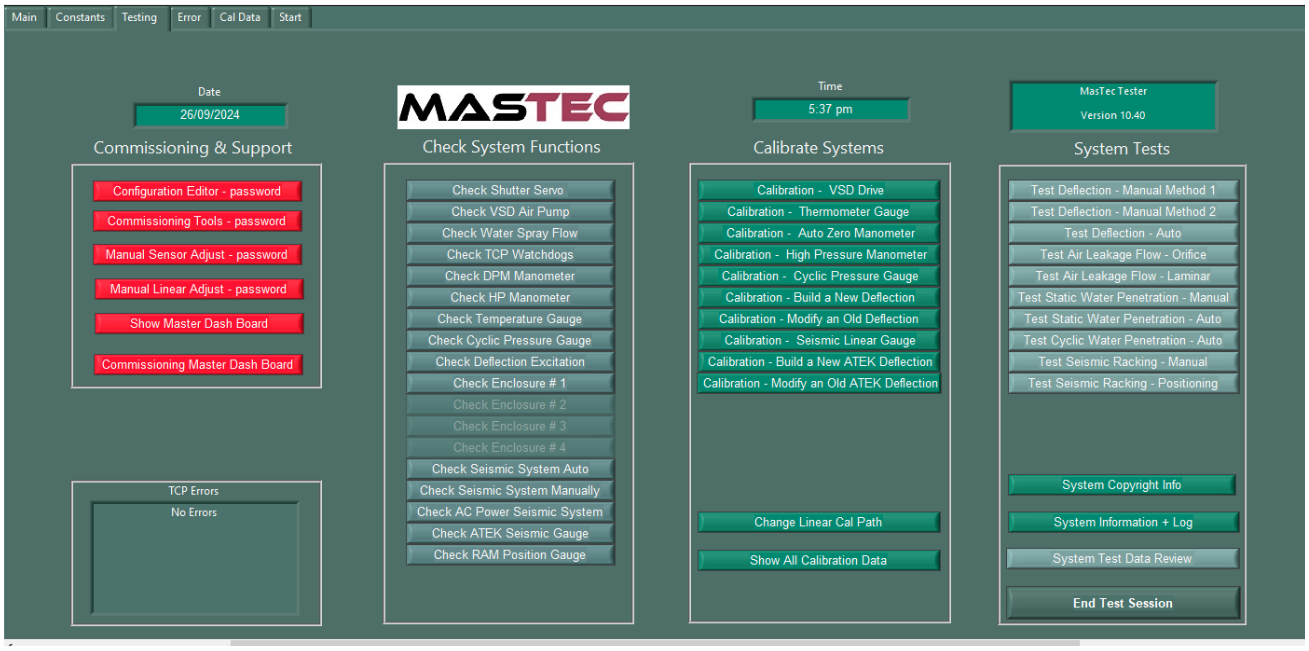
The Testing TAB’s FP is the display where the Operator and Commissioner can drive all the systems functions and options. (Commissioning, Checking, Calibrating, Testing)

(Access to the commissioning & support section are password protected).

Notice the button “Commissioning Tools”.

It has a two channel Scope and DSA, a 48 Channel DVM, and Analog Channel Noise analysers.

Plus an engineering unit converter section for the LFE Calibration polynomial coefficient calculations. An LFE is a Laminar Flow Element that measures air flow and has a linear output almost hence the poly coefficients.


Main
Constants
Testing
Error
Cal Data
Start
MASTEC
Commissioning & Support
Configuration Editor - password
Commissioning Tools - password
Manual Sensor Adjust - password
Manual Linear Adjust - password
Show Master Dash Board
Commissioning Master Dash Board
Check System Functions
Check Shutter Servo
Check VSD Air Pump
Check Water Spray Flow
Check TCP Watchdogs
Check DPM Manometer
Check HP Manometer
Check Temperature Gauge
Check Cyclic Pressure Gauge
Check Deflection Excitation
Check Enclosure # 1
Check Enclosure # 2
Check Enclosure # 3
Check Enclosure # 4
Check Seismic System Auto
Check Seismic System Manually
Check AC Power Seismic System
Check ATEK Seismic Gauge
Check RAM Position Gauge
Calibrate Systems
Calibration - VSD Drive
Calibration - Thermometer Gauge
Calibration - Auto Zero Manometer
Calibration - High Pressure Manometer
Calibration - Cyclic Pressure Gauge
Calibration - Build a New Deflection
Calibration - Modify an Old Deflection
Calibration - Seismic Linear Gauge
Calibration - Build a New ATEK Deflection
Calibration - Modify an Old ATEK Deflection
Change Linear Cal Path
Show All Calibration Data
System Tests
Test Deflection - Manual Method 1
Test Deflection - Manual Method 2
Test Deflection - Auto
Test Air Leakage Flow - Orifice
Test Air Leakage Flow - Laminar
Test Static Water Penetration - Manual
Test Static Water Penetration - Auto
Test Cyclic Water Penetration - Auto
Test Seismic Racking - Manual
Test Seismic Racking - Positioning
System Copyright Info
System Information + Log
System Test Data Review
End Test Session

Date 26/09/2024

Time 5:37 pm

MasTec Tester
Version 10.40

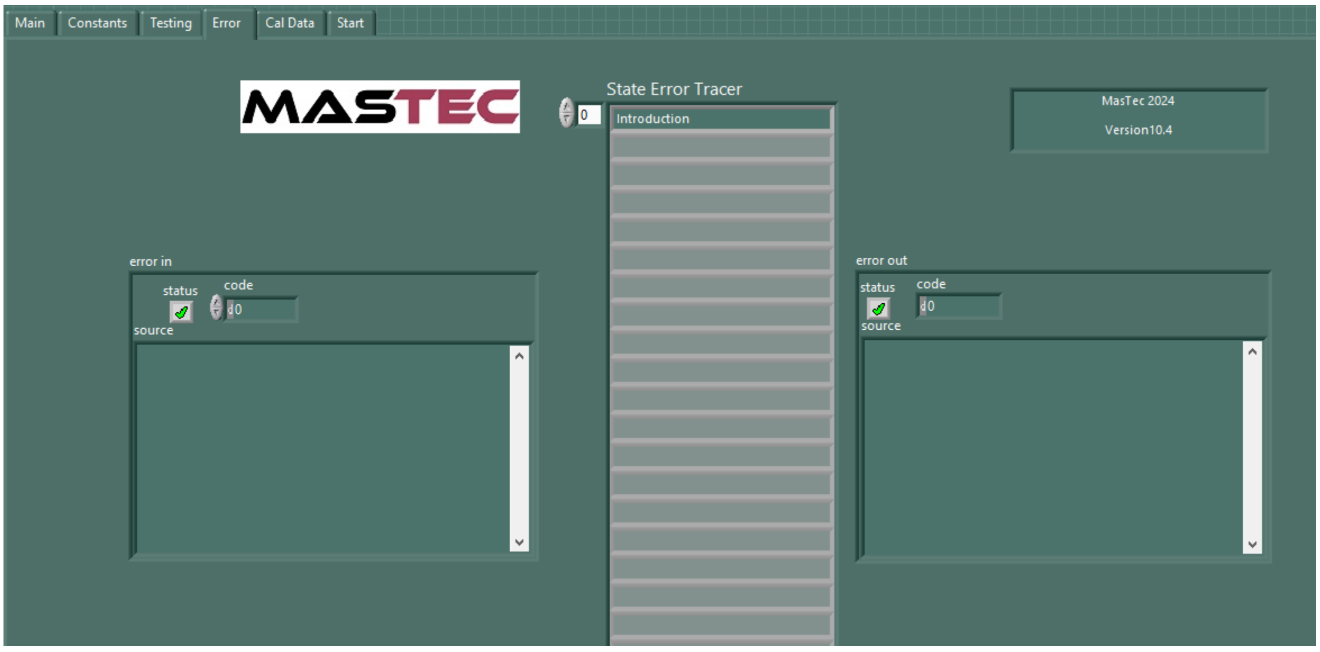
TCP Errors
No Errors

The Error TAB’s FP shows where in the system any error occurs and what it was.

(This error capture systems shows where and in what State and in what VI Level the error occurred)

This is the only part of the program that uses one Global Variable.

Very handy for IO errors in the bowels of the program.



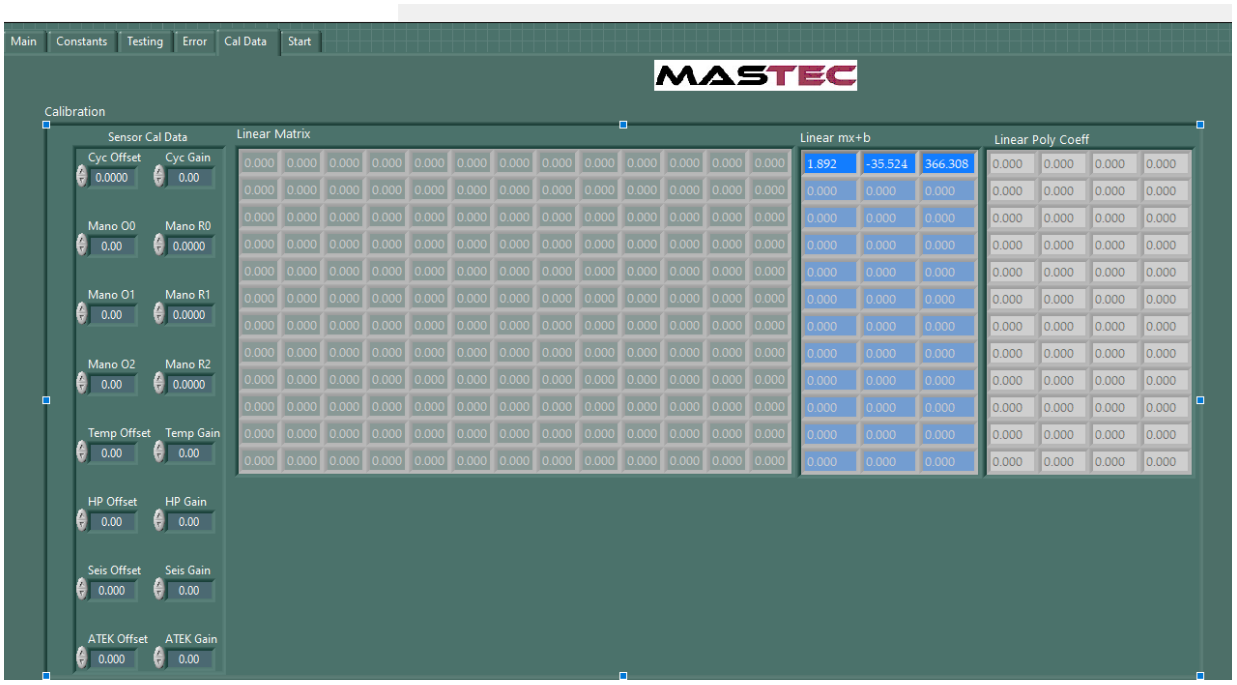
0 Introduction

MasTec 2024
Version10.4

error in
status ✓ code 0
source

error out
status ✓ code 0
source

The Calibration TAB’s FP displays the loaded current Calibration Data for each Linear Gauge (up to 48), plus all the DPM Manometer, other pressure gauges plus the Seismic Test Linear Gauge cal.



Main | Constants | Testing | Error | Cal Data | Start

MASTEC

Calibration

Sensor Cal Data

Cyc Offset 0.0000	Cyc Gain 0.00
Mano O0 0.00	Mano R0 0.0000
Mano O1 0.00	Mano R1 0.0000
Mano O2 0.00	Mano R2 0.0000
Temp Offset 0.00	Temp Gain 0.00
HP Offset 0.00	HP Gain 0.00
Seis Offset 0.000	Seis Gain 0.00
ATEK Offset 0.000	ATEK Gain 0.00

Linear Matrix

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Linear mx+b

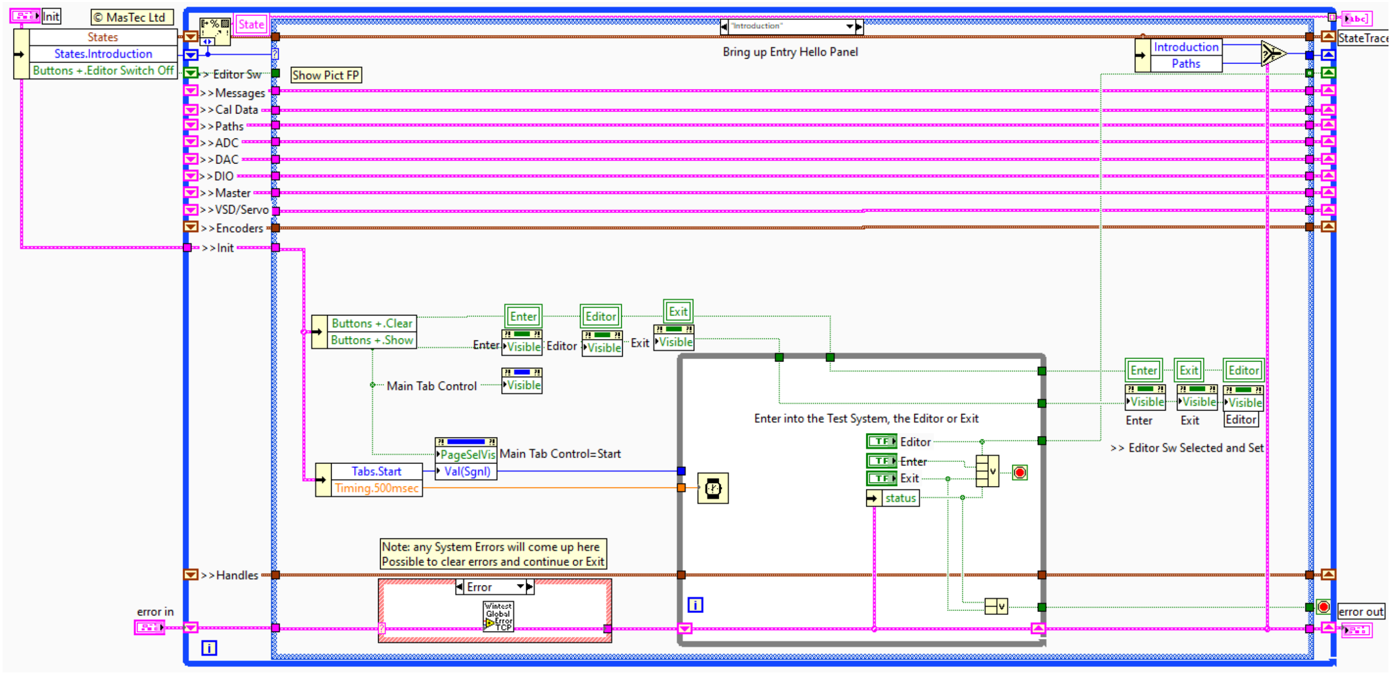
1.892	-35.524	366.308
0.000	0.000	0.000
0.000	0.000	0.000
0.000	0.000	0.000
0.000	0.000	0.000
0.000	0.000	0.000
0.000	0.000	0.000
0.000	0.000	0.000
0.000	0.000	0.000
0.000	0.000	0.000
0.000	0.000	0.000
0.000	0.000	0.000

Linear Poly Coeff

0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000

Now we will look inside the Main VI’s diagram with some explanation notes on how to use COCs in any expanded way..

First State “Introduction”



Action:- Top Left, the Init COC is unbundled to set up three SRs.

The top unbundle contains all the Main VI’s States as a Cluster of Enums.

The middle unbundle sets the first Enum state to run = “Introduction”.

The bottom unbundle is a boolean switch to allow the Editor Button Selection on the Main Front Panel.

Looking inside the diagram the “Init” and “Encoder” public data COCs were not shown in the book.

The Action in the Introduction State/Case

1/ The Init COC data is not placed on an SR as it is fixed public data.

2/ The Init COC cluster Buttons + shows the use of Clear, Show and Run Booleans.

This use allows the three FP Control Buttons “Enter, Exit and Editor” Properties to be controlled

3/ This case also manages all the global error information and reporting. Maybe look at that later.

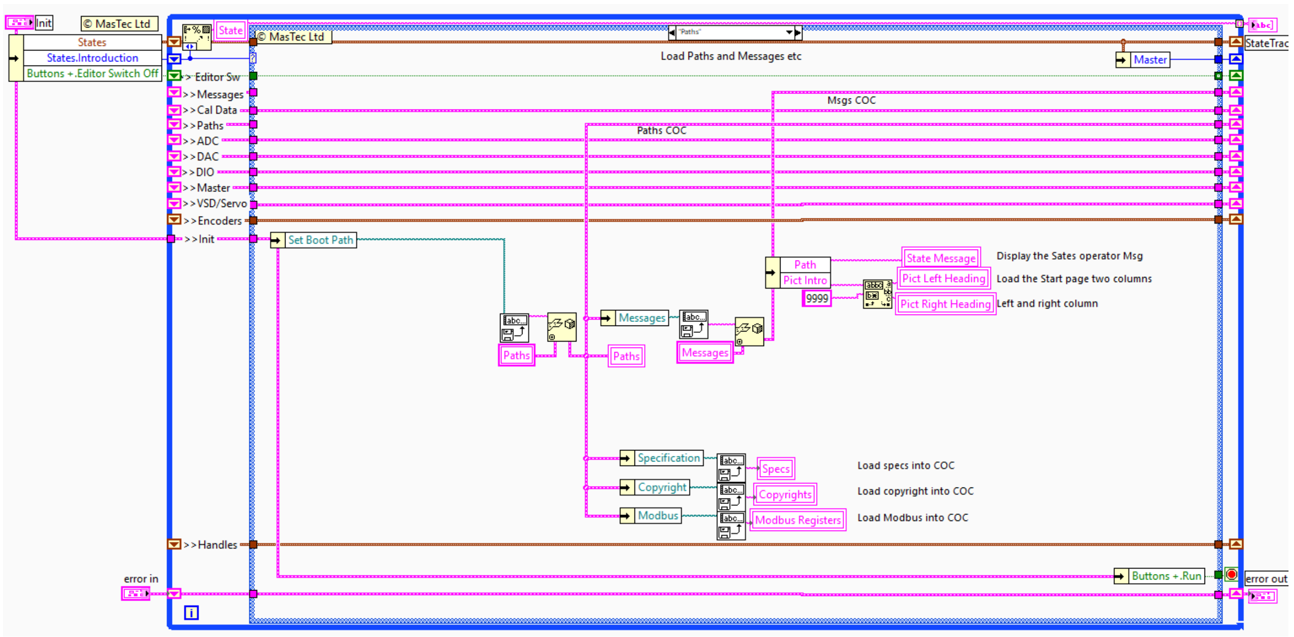
4/ Press any of the three buttons and action is taken. Enter sets up the next State to run “Paths”, Exit stops the program, and Editor sets up the boolean flag to just make the Editor to run.

5/ Any Errors not dealt with are trapped and Introduction runs again.

6/ Notice how the Main TAB is selected.

Most of this activity is directed by the Init COC.

Next State is “Paths”



Actions

1/ Paths State uses the Init COC file path control to access the file with the Paths COC.

This loads all the data into the 4 Paths clusters > Public Calibration, Private Test Data, Public Config Data and Diagram Constants Data.

2/ Plus it loads and reads various data into other local variables “Copyright”, “Modbus Registers”, “Specifications”

Also loads into the front panel in the “Start” TAB some info to show in the two columns (left and right).

Simple and clean.

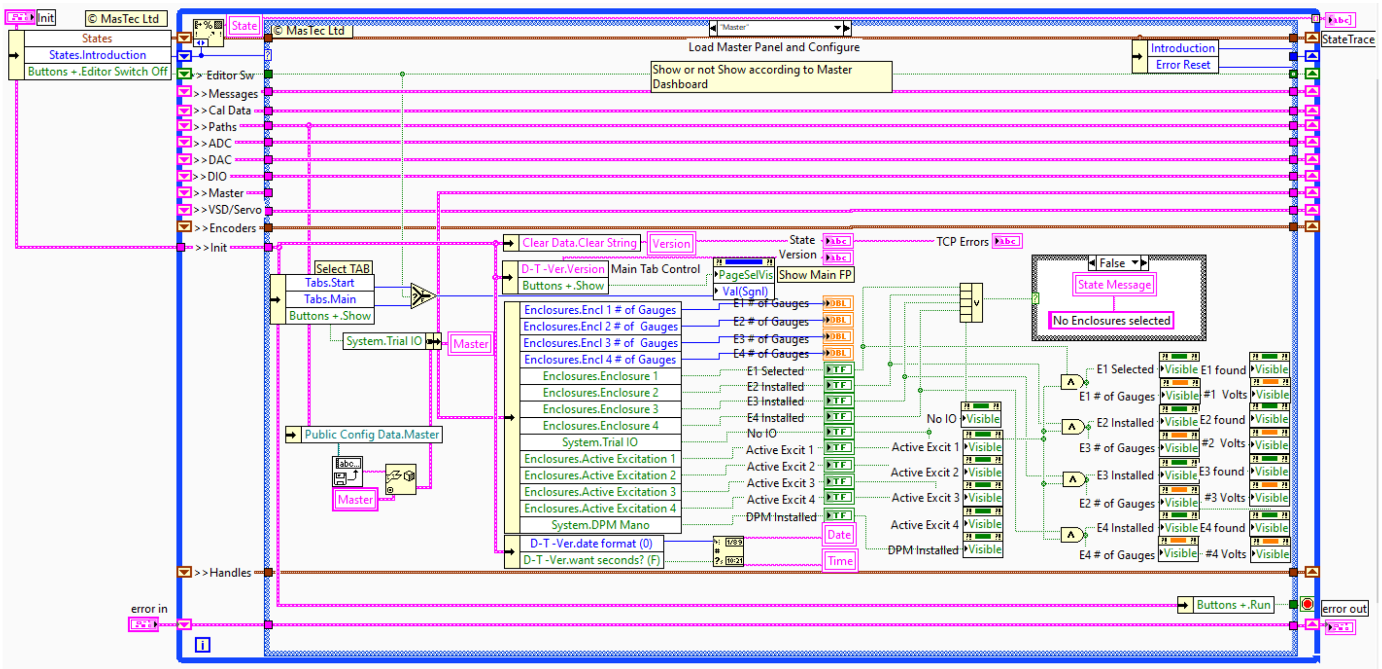
3/ Notice:- The Auto adjusting loading COCs use Local Variables for the loading of the shape of the loaded data to match the COCs.

If as you are working away and you add or subtract info in a COC, this load method automatically adjusts the unflattening from string shape to obtain the new COC design.

It is quite freaky the first time you do this. Add a whole new section to a COC save it and on the next load the additions magically appear correctly, with no program adjustments.

4/ Notice also the Init COC unbundling of “States” and selecting the next state to run as “Master” Top right.

Next State Master is very very busy Unbundling and Displaying



Action >

1/ The Main FP displays the Master COC settings for the IO.

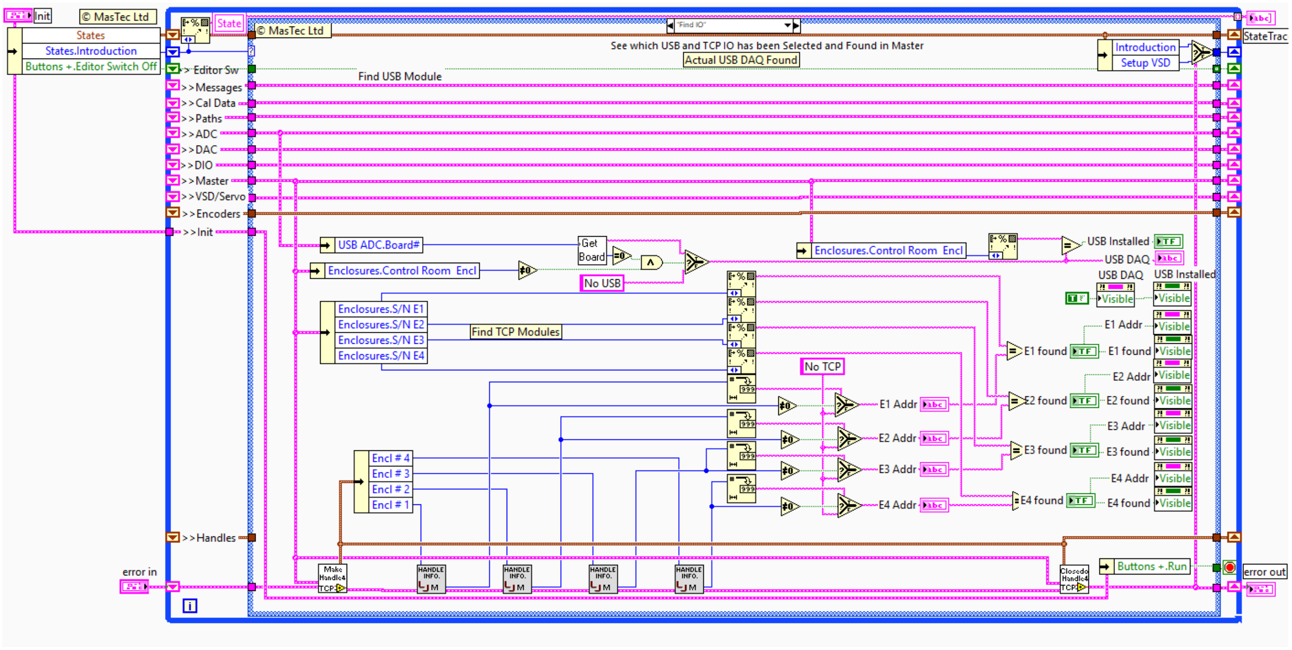
2/ This Master COC is usually setup by the Commission Engineer and never adjusted by the Operators.

3/ The next State Find IO actually checks to see if all the IO selected is actually present.

It is late in the order of loading and configuring.

Next State to look at is “Find IO”, the one that verifies that the Master COC settings are correct.

It looks at the TCP DAQ and the USB DAQ to verify they are present and have the right Serial numbers etc.

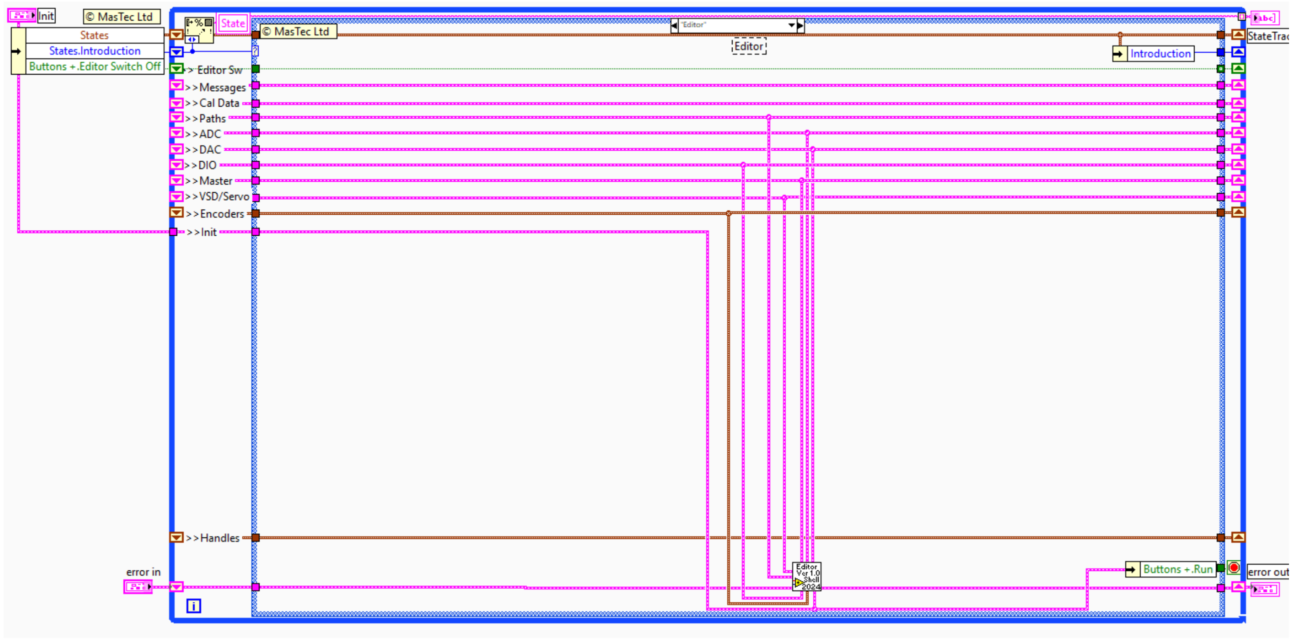


This “Find IO” State checks for the USB DAQ unit plus runs up the TCP Modbus handles for each of the TCP DAQs (up to 4 units) to see if they are present. It does the TCP test by reading the TCP DAQ Units Serial numbers to see if they match the Serial numbers stored in the Master COC. It then displays the TCP Serial numbers of each TCP DAQ unit and marks them absent or present.

If the site installs a new TCP DAQ unit this will get flagged and the Commissioner will have to load in the new Serial number of the new TCP DAQ module, before it can be used.

I have had to personally do this several times using Teamviewer access to a site in South East Asia. This method allows the commissioner to know each site’s Hardware and which TCP DAQs are working and which have failed etc.

Here is the “Editor” State



The “Editor” pops its FP when it runs and on exiting closes the FP and goes right back to the “Introduction” State

The new “Init” COC allows the Editing the settings of the Main States, plus it allows adjusting the state engine order of running easily.

For example you don’t have the IO running yet but you want to be able to continue to develop just make the states miss out all the IO stuff.

This is what happens when you click the Enter Setup Editor button on the Start FP. The method I use, all the currently saved Public COC Data (ADC, DAC, DIO, Encoder, VSD & Servo) are loaded from disk and then the program goes to the Editor. None of the IO states run.

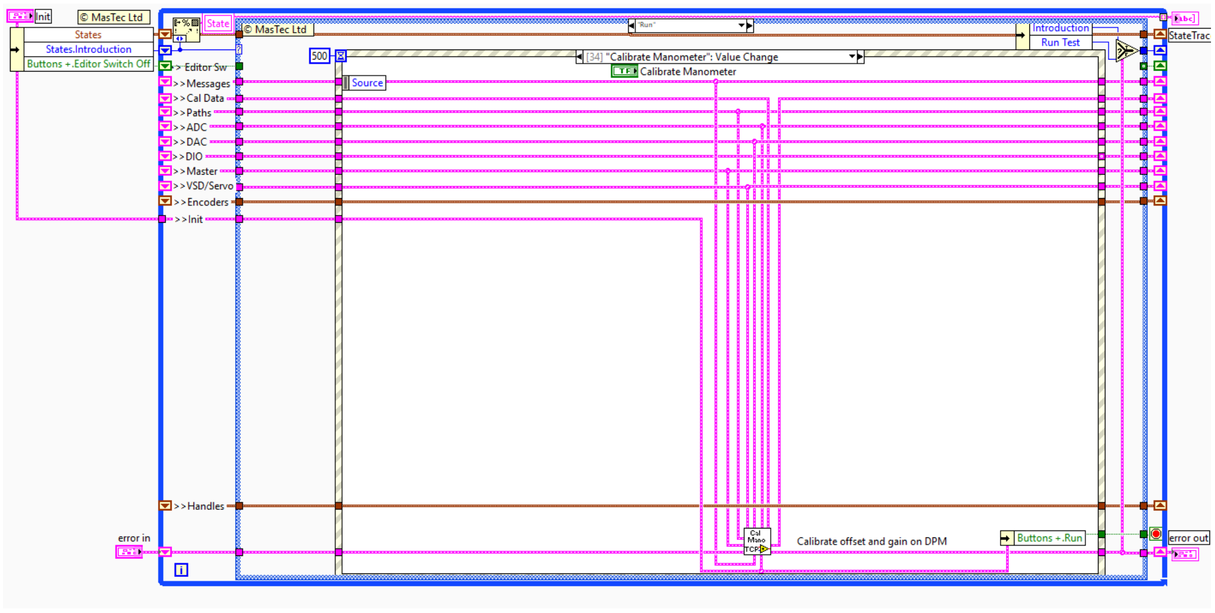
When exiting the Editor program you go back to the “Introduction” State and display the Start FP again.

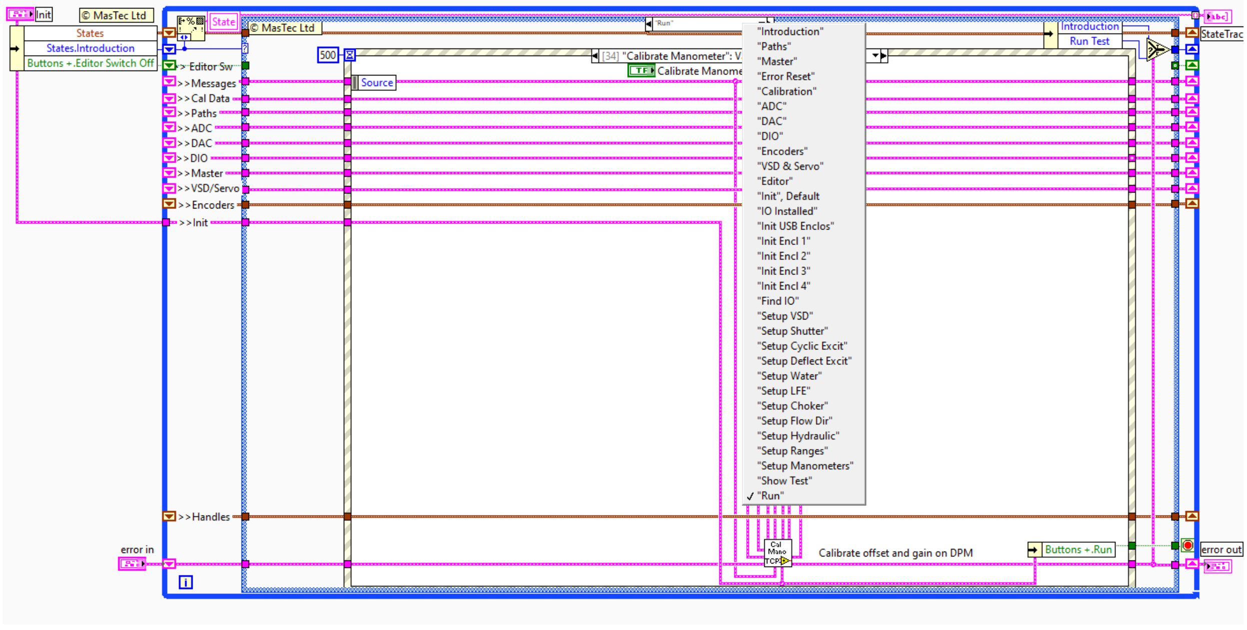
Note :- To access the Editor you have enter a password and only the Commissioner and Site Test Manager can access this.

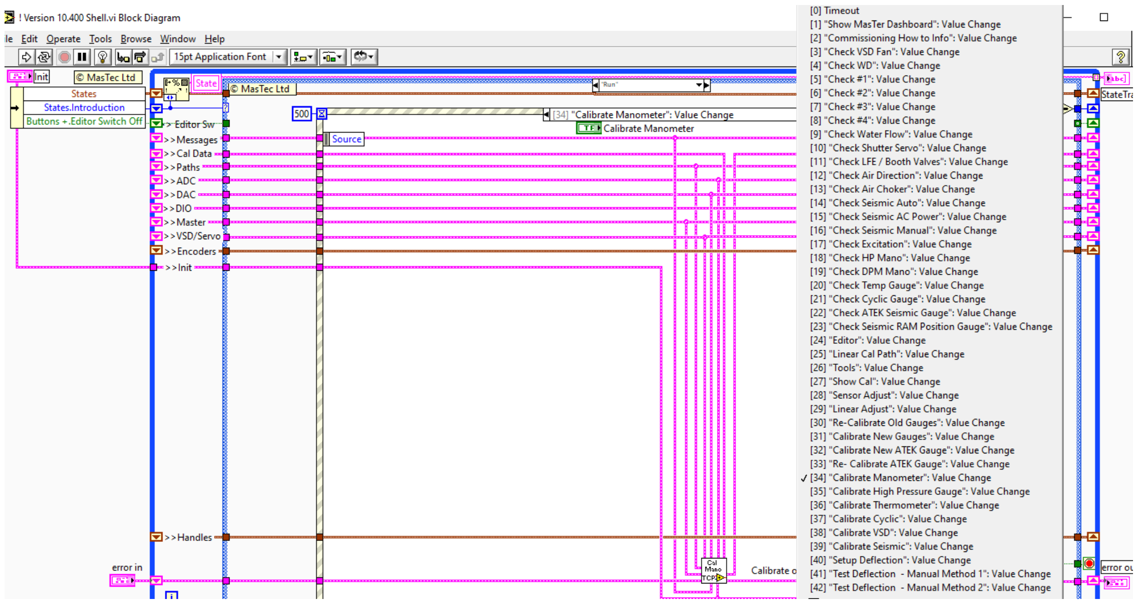
Note :- I have also built a new editor that is an improvement also.
I will show the addition to the Editor that allows creating a Backup copy of all the COCs files and folders for safety.

Next we can look at a Test Component (A State Engine) and see how the COCs have become much more than just “Hidden Data” orientated but more like a declaration data block that is editable.

This Component is “Calibration of the DPM Manometer” as a typical example.





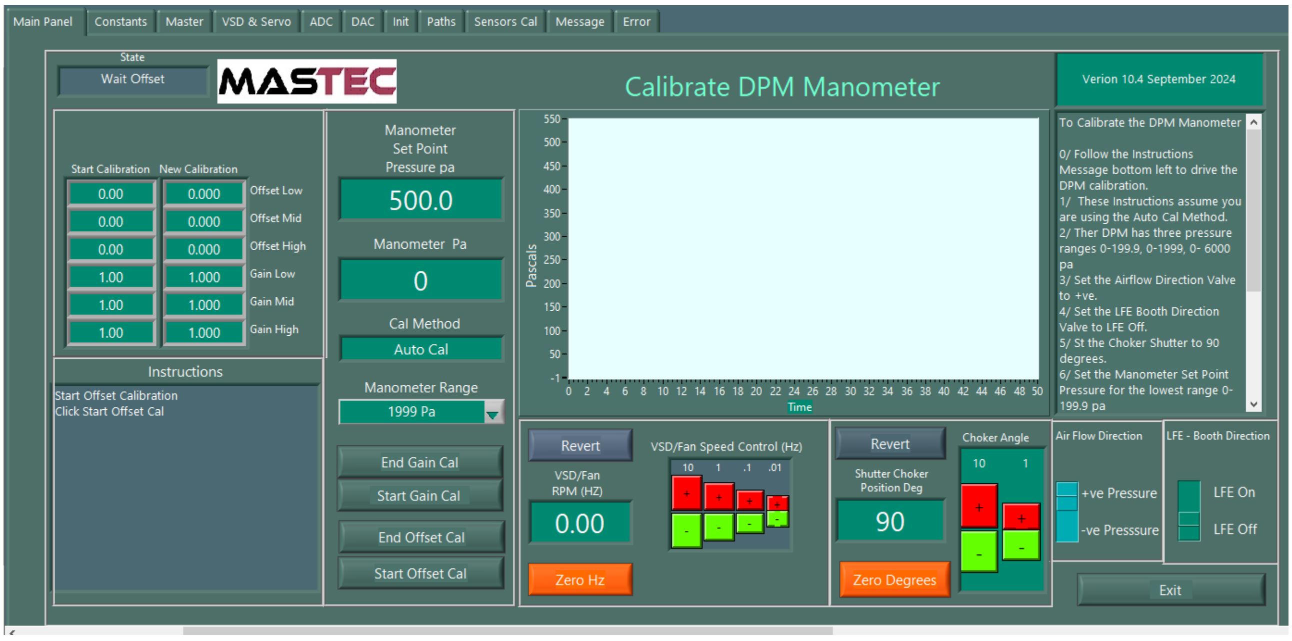


```
[0] Timeout
[1] "Show MasTer Dashboard": Value Change
[2] "Commissioning How to Info": Value Change
[3] "Check VSD Fan": Value Change
[4] "Check WD": Value Change
[5] "Check #1": Value Change
[6] "Check #2": Value Change
[7] "Check #3": Value Change
[8] "Check #4": Value Change
[9] "Check Water Flow": Value Change
[10] "Check Shutter Servo": Value Change
[11] "Check LFE / Booth Valves": Value Change
[12] "Check Air Direction": Value Change
[13] "Check Air Choker": Value Change
[14] "Check Seismic Auto": Value Change
[15] "Check Seismic AC Power": Value Change
[16] "Check Seismic Manual": Value Change
[17] "Check Excitation": Value Change
[18] "Check HP Mano": Value Change
[19] "Check DPM Mano": Value Change
[20] "Check Temp Gauge": Value Change
[21] "Check Cyclic Gauge": Value Change
[22] "Check ATEK Seismic Gauge": Value Change
[23] "Check Seismic RAM Position Gauge": Value Change
[24] "Editor": Value Change
[25] "Linear Cal Path": Value Change
[26] "Tools": Value Change
[27] "Show Cal": Value Change
[28] "Sensor Adjust": Value Change
[29] "Linear Adjust": Value Change
[30] "Re-Calibrate Old Gauges": Value Change
[31] "Calibrate New Gauges": Value Change
[32] "Calibrate New ATEK Gauge": Value Change
[33] "Re- Calibrate ATEK Gauge": Value Change
✓ [34] "Calibrate Manometer": Value Change
[35] "Calibrate High Pressure Gauge": Value Change
[36] "Calibrate Thermometer": Value Change
[37] "Calibrate Cyclic": Value Change
[38] "Calibrate VSD": Value Change
[39] "Calibrate Seismic": Value Change
[40] "Setup Deflection": Value Change
[41] "Test Deflection  - Manual Method 1": Value Change
[42] "Test Deflection  - Manual Method 2": Value Change
```

I put up three picts as you can see with one showing off all the Main VI States and another one showing the Test Events. There are more events than the 42 but it gives an idea of what the program does.

First pict is the Event engine where the Manometer Calibration Component sits in. The Component has the all needed Public COC Data connected to it, even the new Init COC.

The "Calibrate DPM Manometer" Front Panel.



Start Calibration	New Calibration	
0.00	0.000	Offset Low
0.00	0.000	Offset Mid
0.00	0.000	Offset High
1.00	1.000	Gain Low
1.00	1.000	Gain Mid
1.00	1.000	Gain High

Instructions

Start Offset Calibration
Click Start Offset Cal

To Calibrate the DPM Manometer

0/ Follow the Instructions Message bottom left to drive the DPM calibration.
1/ These Instructions assume you are using the Auto Cal Method.
2/ Ther DPM has three pressure ranges 0-199.9, 0-1999, 0- 6000 pa
3/ Set the Airflow Direction Valve to +ve.
4/ Set the LFE Booth Direction Valve to LFE Off.
5/ St the Choker Shutter to 90 degrees.
6/ Set the Manometer Set Point Pressure for the lowest range 0-199.9 pa

Just a note here on the calibration method used.
We are not actually calibrating the manometer per se but just the analog output voltages of the meter.
The reason for this, is that the DPM is the sites secondary standards calibration pressure device and the readings off the digital display are accurate but the voltages until calibrated out are not known.
For example: Zero pressure may not read zero volts and on the low range 199.9 pa may not read 199.9 mv.
By Calibrating with known pressures we obtains these voltages,
Offsets reading from a zero pressure and a Gains reading off an applied Pressure.
Note :- There are three ranges 0-199.9 pa, 0-1999 pa and 0-6000 pa.
So we are doing y=mx+b stuff. b= offset and m = gain or the slope.

This VI has two ways to calibrate the DPM meter, either using a Manual method or an Auto method. In most cases the Auto is best and faster.

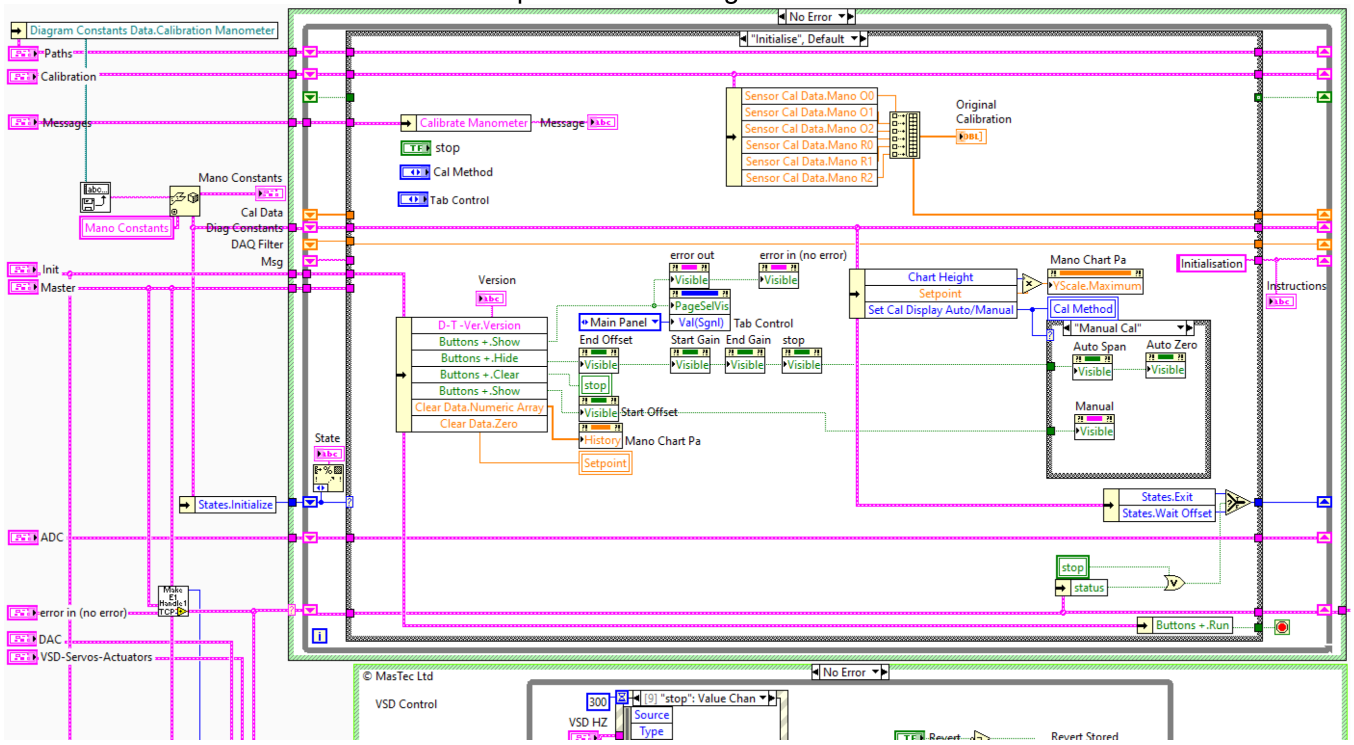
Notice the operator can also see the previous cal data. As they calibrate, they can see the new data and compare to the old values. If there is a large difference this will more than likely indicate the manometer is starting to fail in some way and should be sent in for service. My long experience with the DPM meters has shown their calibration voltages are relatively stable for up to 20 years.

These DPM meters are bi directional + differential pressure + auto zero meters. The auto zero feature means they keep their baseline (offset readings) true all day even if there are temperature changes etc. The newer models are “orientation unaffected” also.

When calibrating the method used, is offset cal first and then the gain cal. This Cal Manometer component does not leave this to chance and guides the operator to do the cal in those steps.

So don’t assume an operator will know this. If for example they do Gain cal first using the Auto cal method the gain for that pressure range will almost go to infinity. I use to get calls on this issue before I forced the correct order method to cal in software. You can put it in a manual and assume the operator will read the method but not always and sometimes misread.

Now we look inside to see how he component has changed from the book.

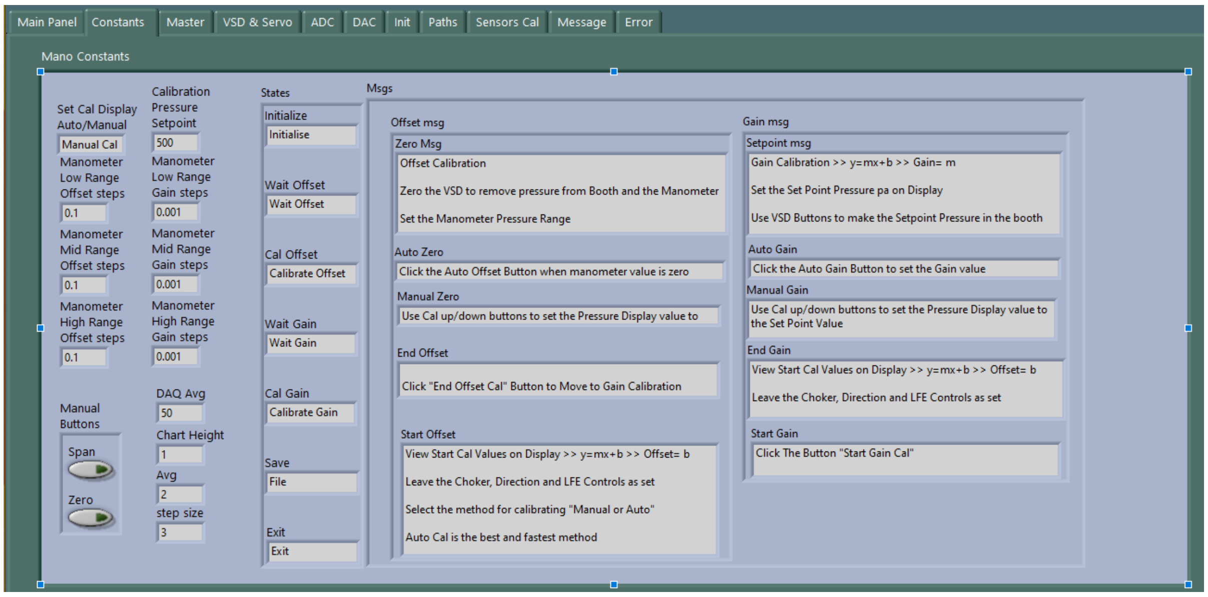


First point to note, is that this Component has multiple tasks running. You can just see the top of the second task VSD Control showing at the bottom.

There are four more tasks that run in asynchronous timing to do the Main task shown and these sit below the VSD Task. They all need access to the COCs also.

This is why the Private COC is loaded outside the all the Tasks to allow easy sharing for all

This pict is the Private COC for this Calibrate DPM component.



Mano Constants

Set Cal Display Auto/Manual: Manual Cal
Manometer Low Range Offset steps: 0.1
Manometer Mid Range Offset steps: 0.1
Manometer High Range Offset steps: 0.1
Manual Buttons: Span, Zero

Calibration Pressure Setpoint: 500
Manometer Low Range Gain steps: 0.001
Manometer Mid Range Gain steps: 0.001
Manometer High Range Gain steps: 0.001
DAQ Avg: 50
Chart Height: 1
Avg: 2
step size: 3

States
Initialize: Initialise
Wait Offset: Wait Offset
Cal Offset: Calibrate Offset
Wait Gain: Wait Gain
Cal Gain: Calibrate Gain
Save: File
Exit: Exit

Msgs

Offset msg

Zero Msg:
Offset Calibration
Zero the VSD to remove pressure from Booth and the Manometer
Set the Manometer Pressure Range

Auto Zero: Click the Auto Offset Button when manometer value is zero

Manual Zero: Use Cal up/down buttons to set the Pressure Display value to

End Offset: Click "End Offset Cal" Button to Move to Gain Calibration

Start Offset:
View Start Cal Values on Display >> y=mx+b >> Offset= b
Leave the Choker, Direction and LFE Controls as set
Select the method for calibrating "Manual or Auto"
Auto Cal is the best and fastest method

Gain msg

Setpoint msg:
Gain Calibration >> y=mx+b >> Gain= m
Set the Set Point Pressure pa on Display
Use VSD Buttons to make the Setpoint Pressure in the booth

Auto Gain: Click the Auto Gain Button to set the Gain value

Manual Gain: Use Cal up/down buttons to set the Pressure Display value to the Set Point Value

End Gain:
View Start Cal Values on Display >> y=mx+b >> Offset= b
Leave the Choker, Direction and LFE Controls as set

Start Gain: Click The Button "Start Gain Cal"

This COC is expanded now, to have two new clusters “States” and “Msgs”.

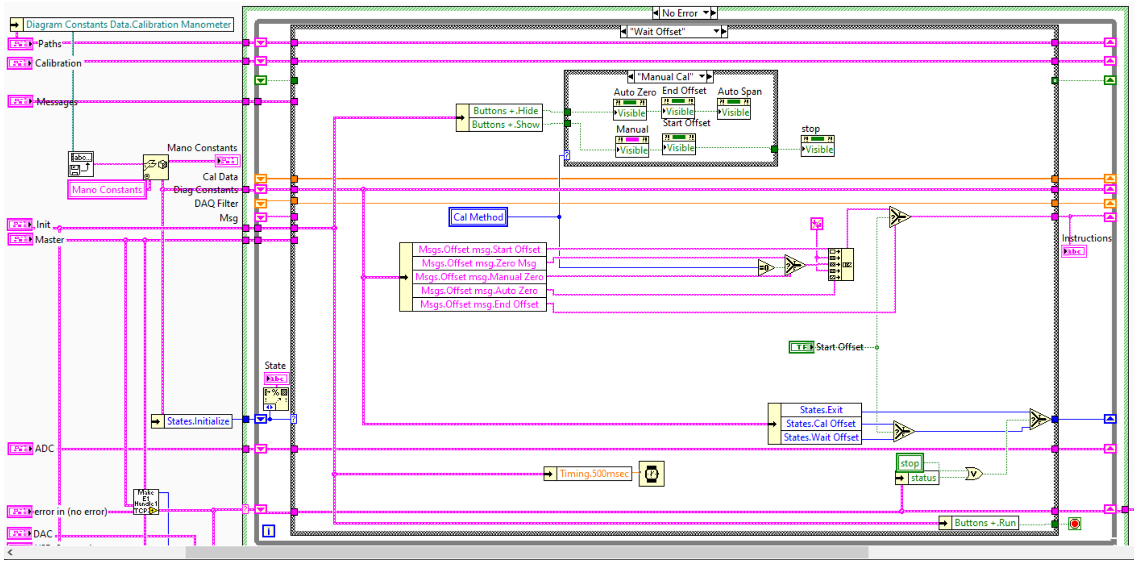
The cluster “States” is the same as the Init COC cluster “States” in that all the component states enums are present and named and are used to drive the component states, messages, initialisation with unbundling.

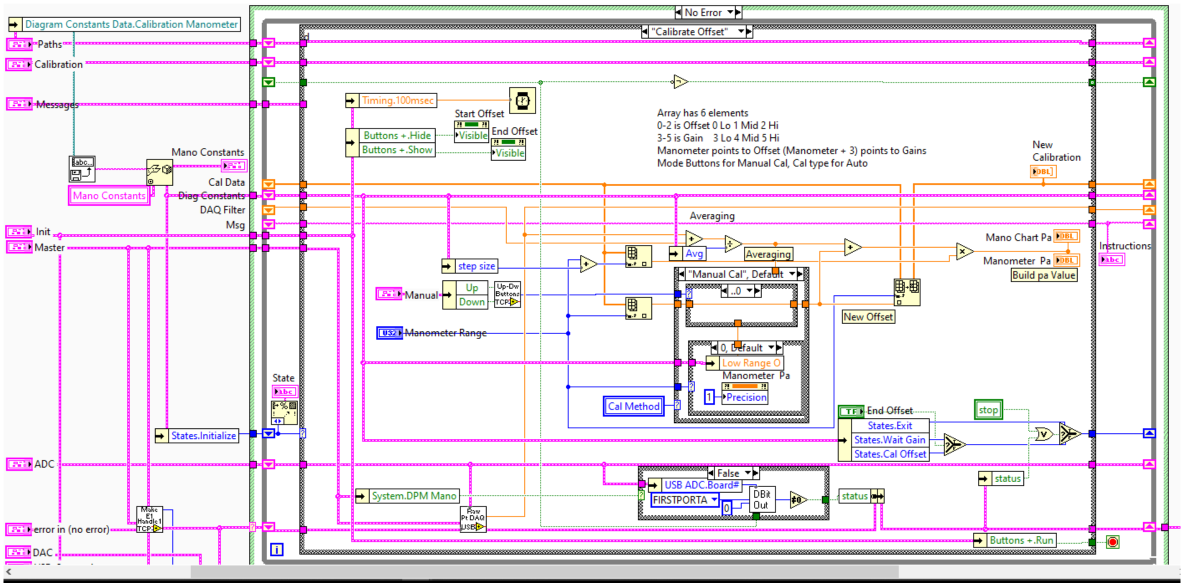
The “Msgs” Cluster gives specific instructions to the operator as they calibrate.

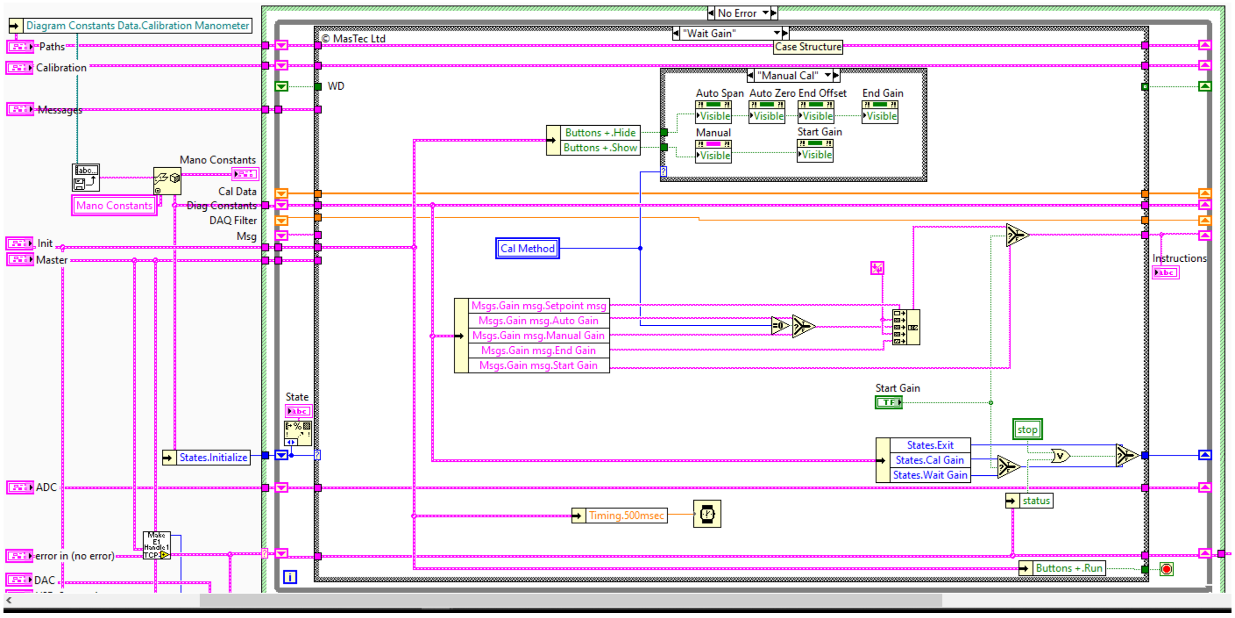
This could be in the Message COC but I feel it is private data for the Component.

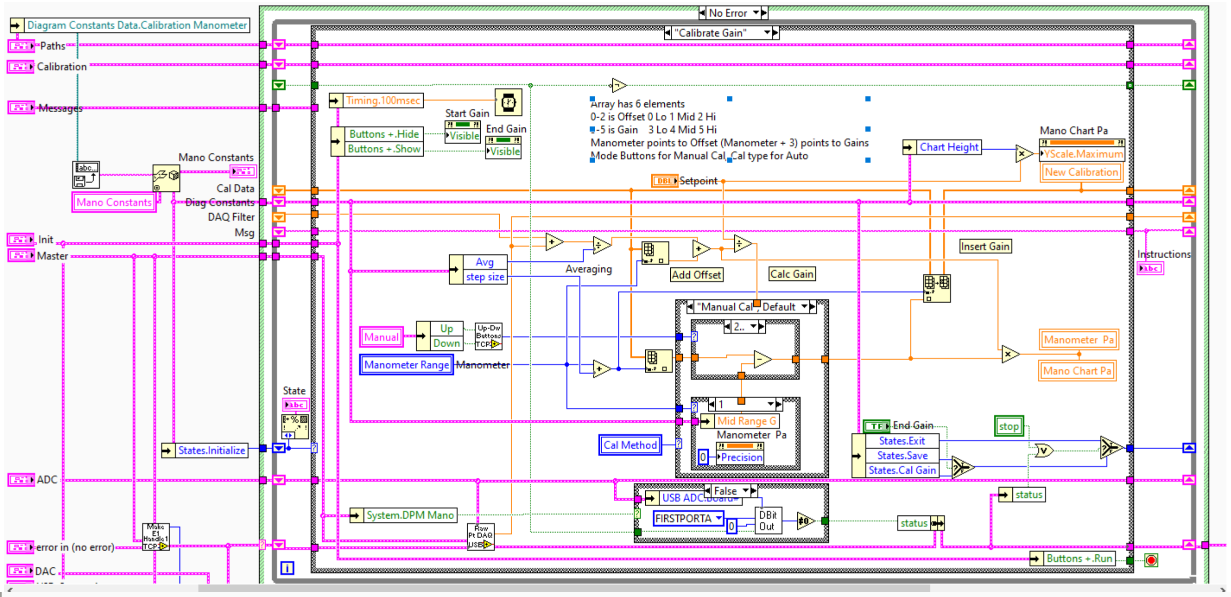
The next few diagrams will show how useful the Init COC’s “Buttons +” cluster is in initialising display and value properties etc.

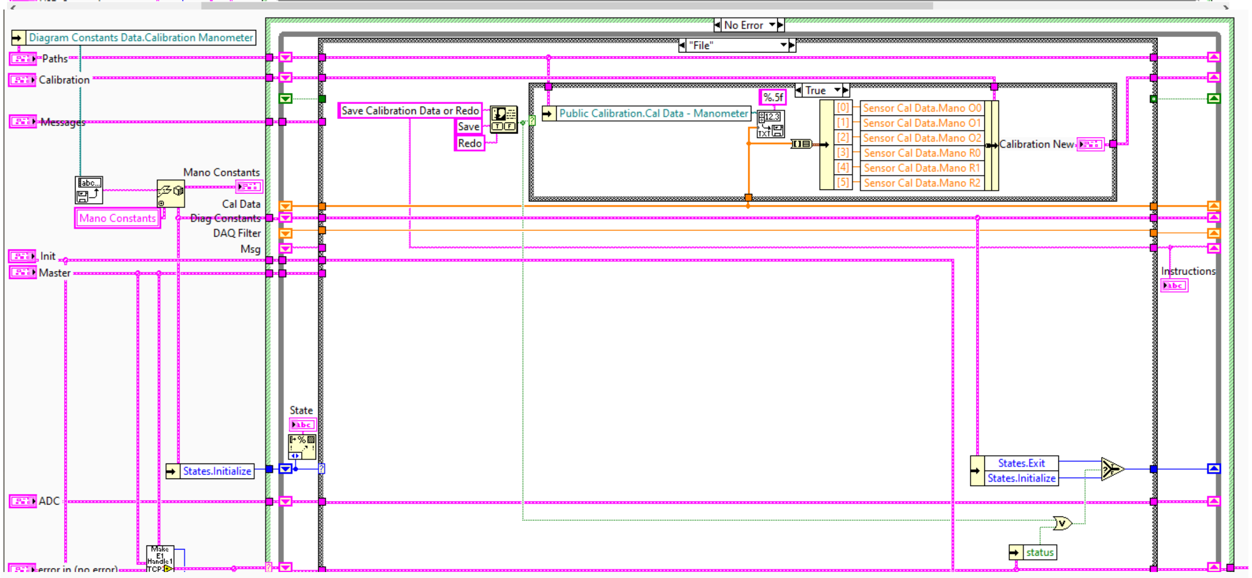
Now all the additional states without explantation.
> states “wait offset, calibrate offset, wait gain, gain calibration, file, and exit”

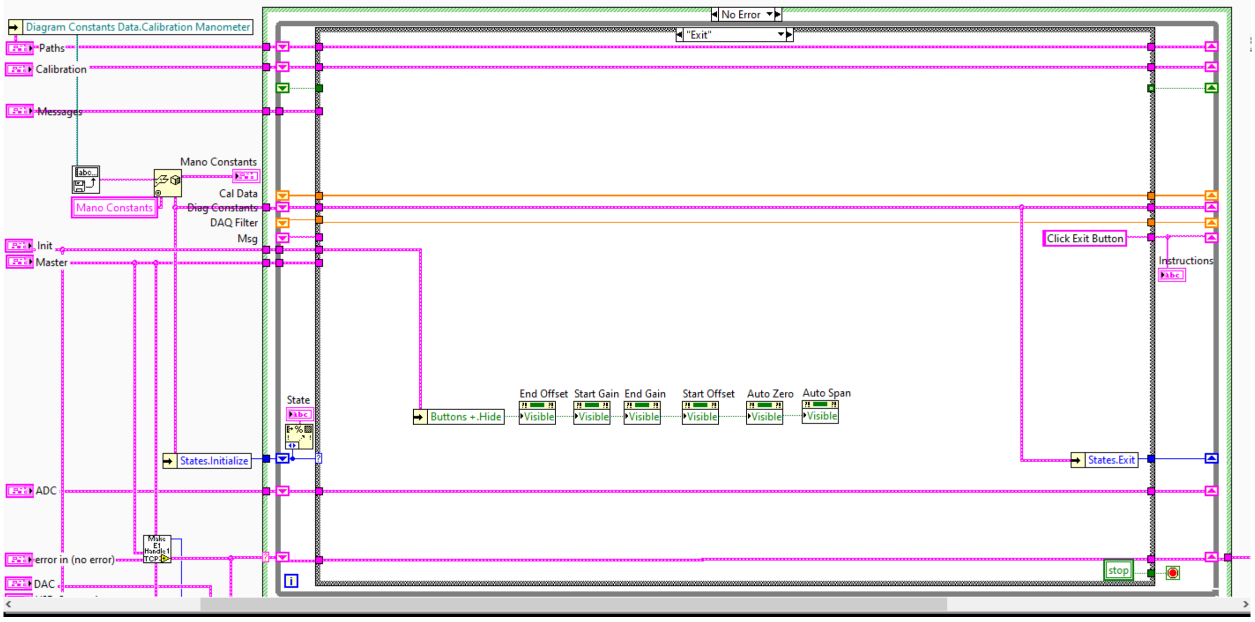






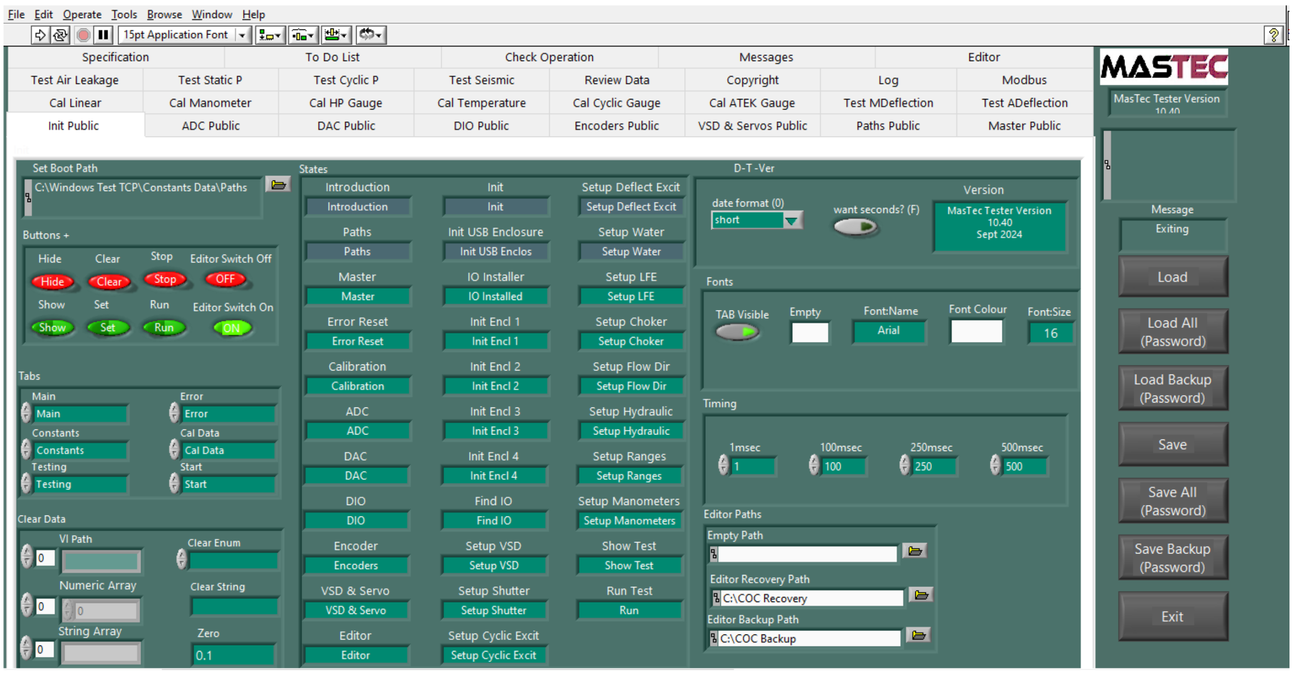






The new Editor is a single case state engine and is event driven now
plus it is completely configurable using all the COCs and that includes the new “Editor” COC
So the editor can edit its own COCs also.

First the Front panel with buttons to drive it. Notice the new Editor COC name in the TABs.



Set Boot Path: C:\Windows Test TCP\Constants Data\Paths

Tabs

Main	Main
Constants	Constants
Testing	Testing
Error	Error
Cal Data	Cal Data
Start	Start

Clear Data

VI Path	0
Numeric Array	0 / 0
String Array	0
Clear Enum	
Clear String	
Zero	0.1

States

Introduction	Introduction
Paths	Paths
Master	Master
Error Reset	Error Reset
Calibration	Calibration
ADC	ADC
DAC	DAC
DIO	DIO
Encoder	Encoders
VSD & Servo	VSD & Servo
Editor	Editor
Init	Init
Init USB Enclosure	Init USB Enclos
IO Installer	IO Installed
Init Encl 1	Init Encl 1
Init Encl 2	Init Encl 2
Init Encl 3	Init Encl 3
Init Encl 4	Init Encl 4
Find IO	Find IO
Setup VSD	Setup VSD
Setup Shutter	Setup Shutter
Setup Cyclic Excit	Setup Cyclic Excit
Setup Deflect Excit	Setup Deflect Excit
Setup Water	Setup Water
Setup LFE	Setup LFE
Setup Choker	Setup Choker
Setup Flow Dir	Setup Flow Dir
Setup Hydraulic	Setup Hydraulic
Setup Ranges	Setup Ranges
Setup Manometers	Setup Manometers
Show Test	Show Test
Run Test	Run

D-T -Ver

date format (0): short
want seconds? (F)
Version: MasTec Tester Version 10.40 Sept 2024

Fonts

TAB Visible	Empty	Font:Name	Font Colour	Font:Size
		Arial		16

Timing

1msec	100msec	250msec	500msec
1	100	250	500

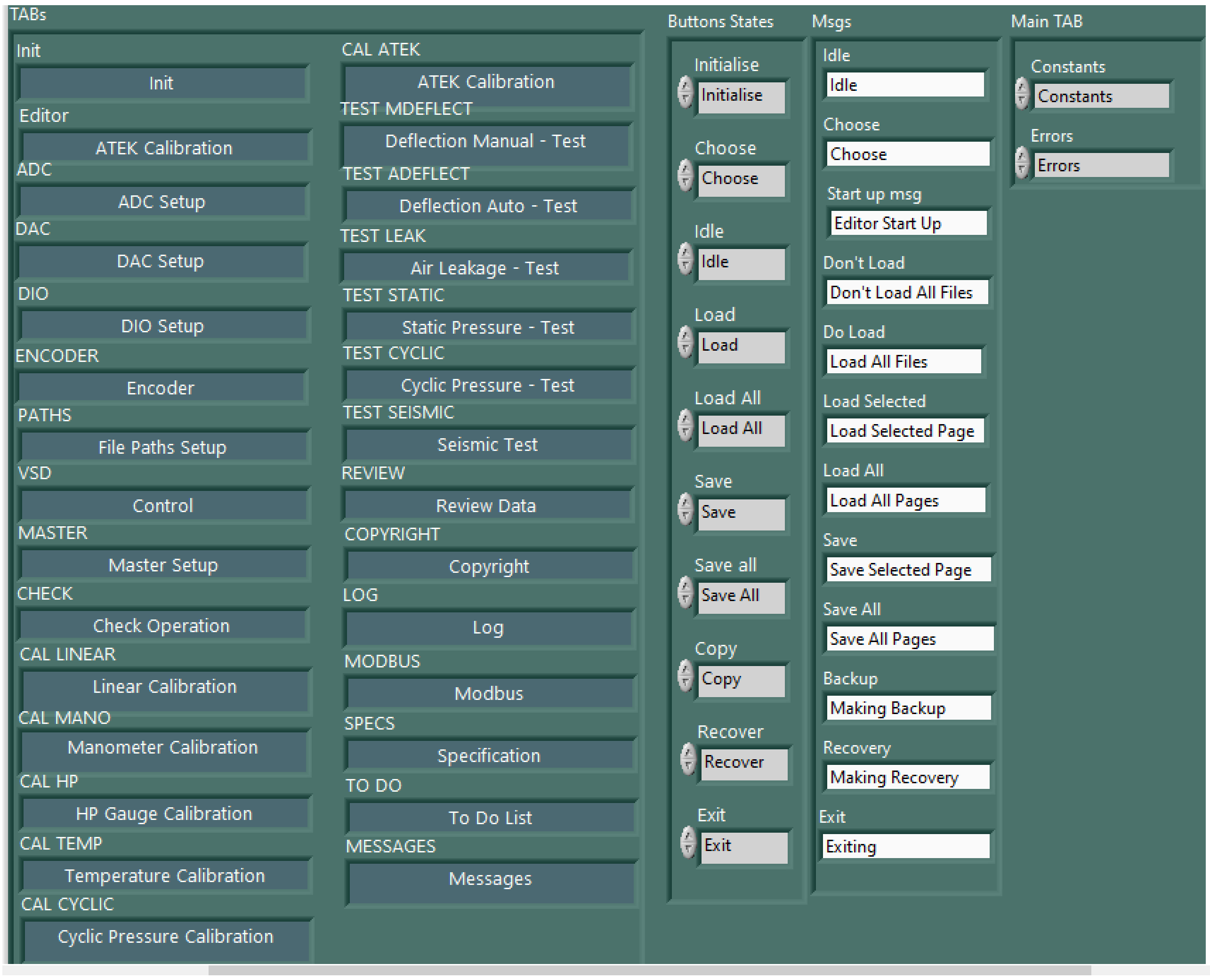
Editor Paths

Empty Path:
Editor Recovery Path: C:\COC Recovery
Editor Backup Path: C:\COC Backup

MasTec Tester Version 10.40

Message: Exiting

The New Editor COC



TABs

Init	Init
Editor	ATEK Calibration
ADC	ADC Setup
DAC	DAC Setup
DIO	DIO Setup
ENCODER	Encoder
PATHS	File Paths Setup
VSD	Control
MASTER	Master Setup
CHECK	Check Operation
CAL LINEAR	Linear Calibration
CAL MANO	Manometer Calibration
CAL HP	HP Gauge Calibration
CAL TEMP	Temperature Calibration
CAL CYCLIC	Cyclic Pressure Calibration
CAL ATEK	ATEK Calibration
TEST MDEFLECT	Deflection Manual - Test
TEST ADEFLECT	Deflection Auto - Test
TEST LEAK	Air Leakage - Test
TEST STATIC	Static Pressure - Test
TEST CYCLIC	Cyclic Pressure - Test
TEST SEISMIC	Seismic Test
REVIEW	Review Data
COPYRIGHT	Copyright
LOG	Log
MODBUS	Modbus
SPECS	Specification
TO DO	To Do List
MESSAGES	Messages

Buttons States

Initialise	Initialise
Choose	Choose
Idle	Idle
Load	Load
Load All	Load All
Save	Save
Save all	Save All
Copy	Copy
Recover	Recover
Exit	Exit

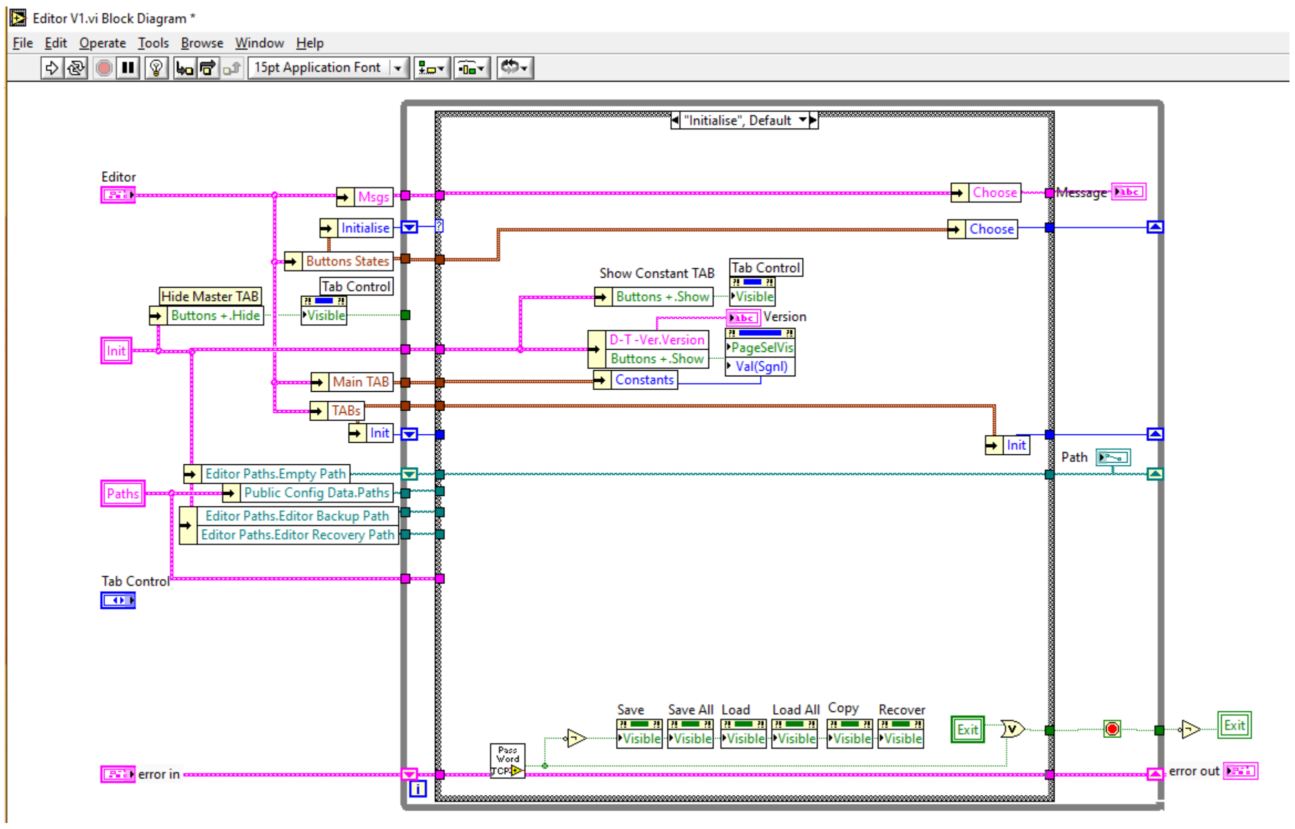
Msgs

Idle	Idle
Choose	Choose
Start up msg	Editor Start Up
Don't Load	Don't Load All Files
Do Load	Load All Files
Load Selected	Load Selected Page
Load All	Load All Pages
Save	Save Selected Page
Save All	Save All Pages
Backup	Making Backup
Recovery	Making Recovery
Exit	Exiting

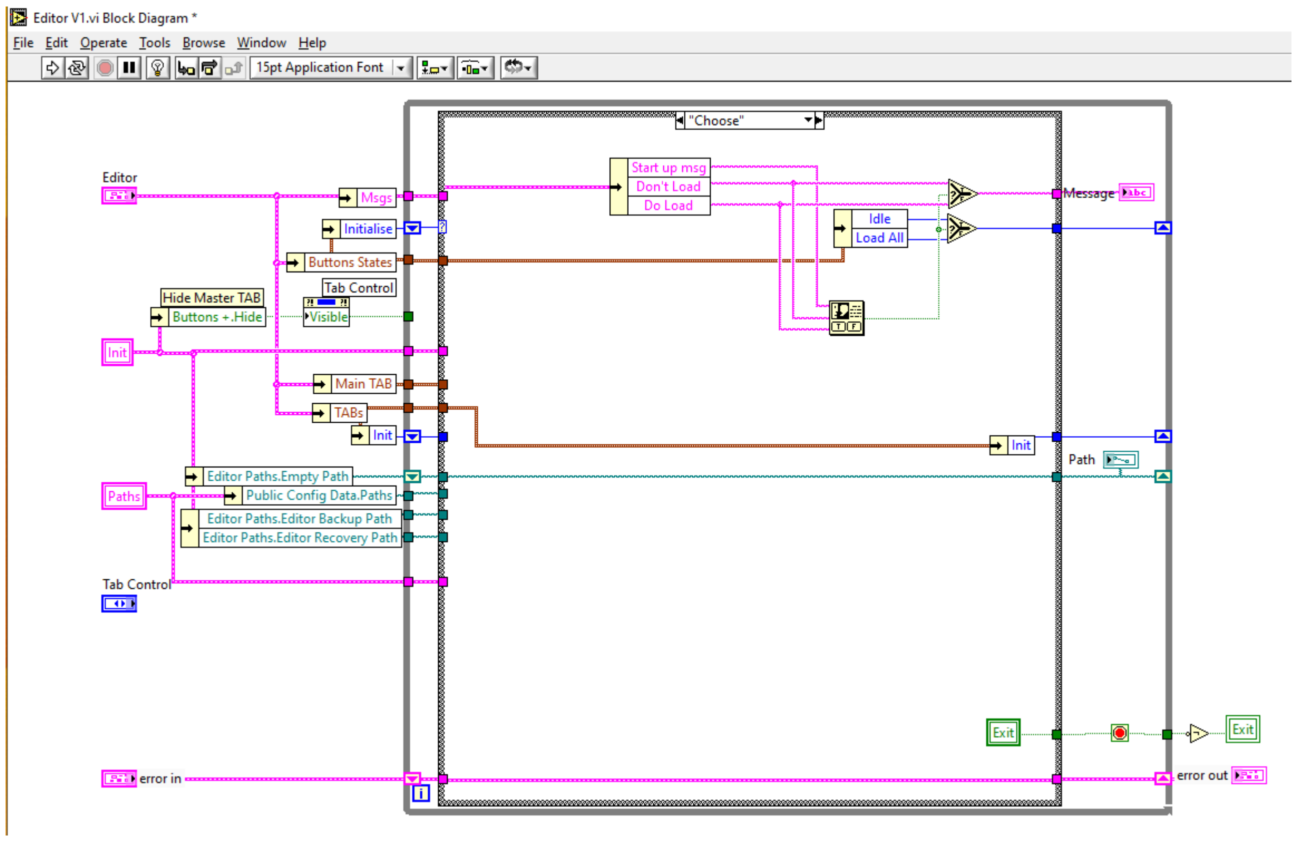
Main TAB

Constants	Constants
Errors	Errors

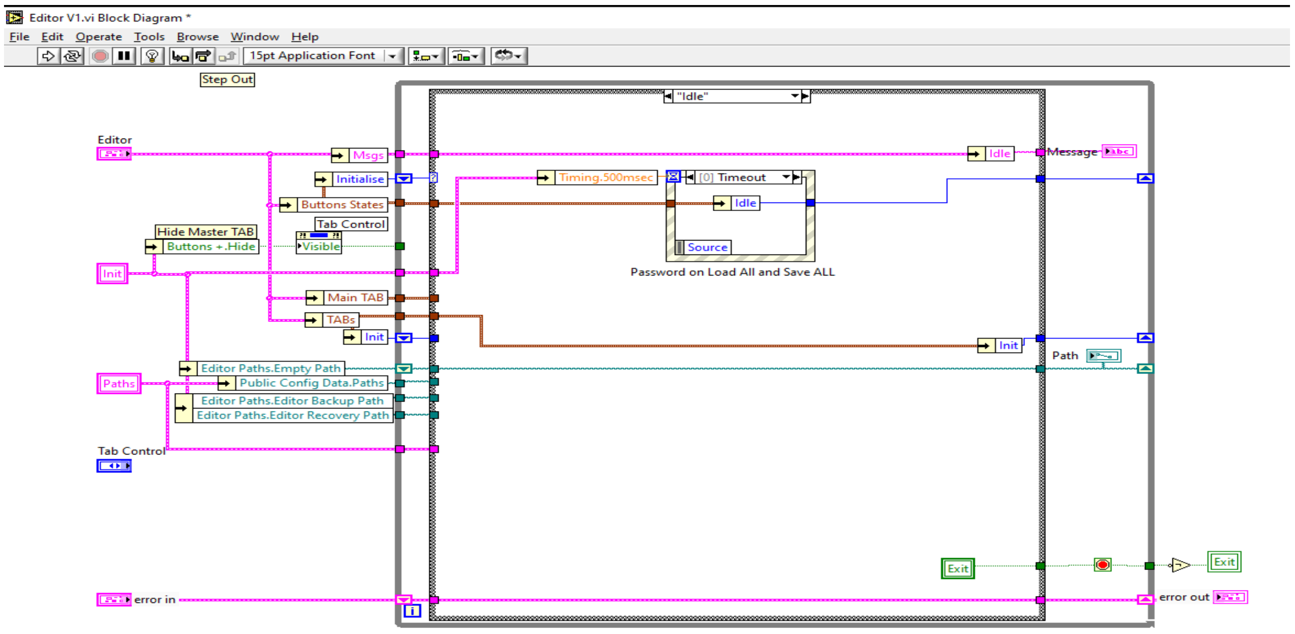
Next Diagram Initialisation.



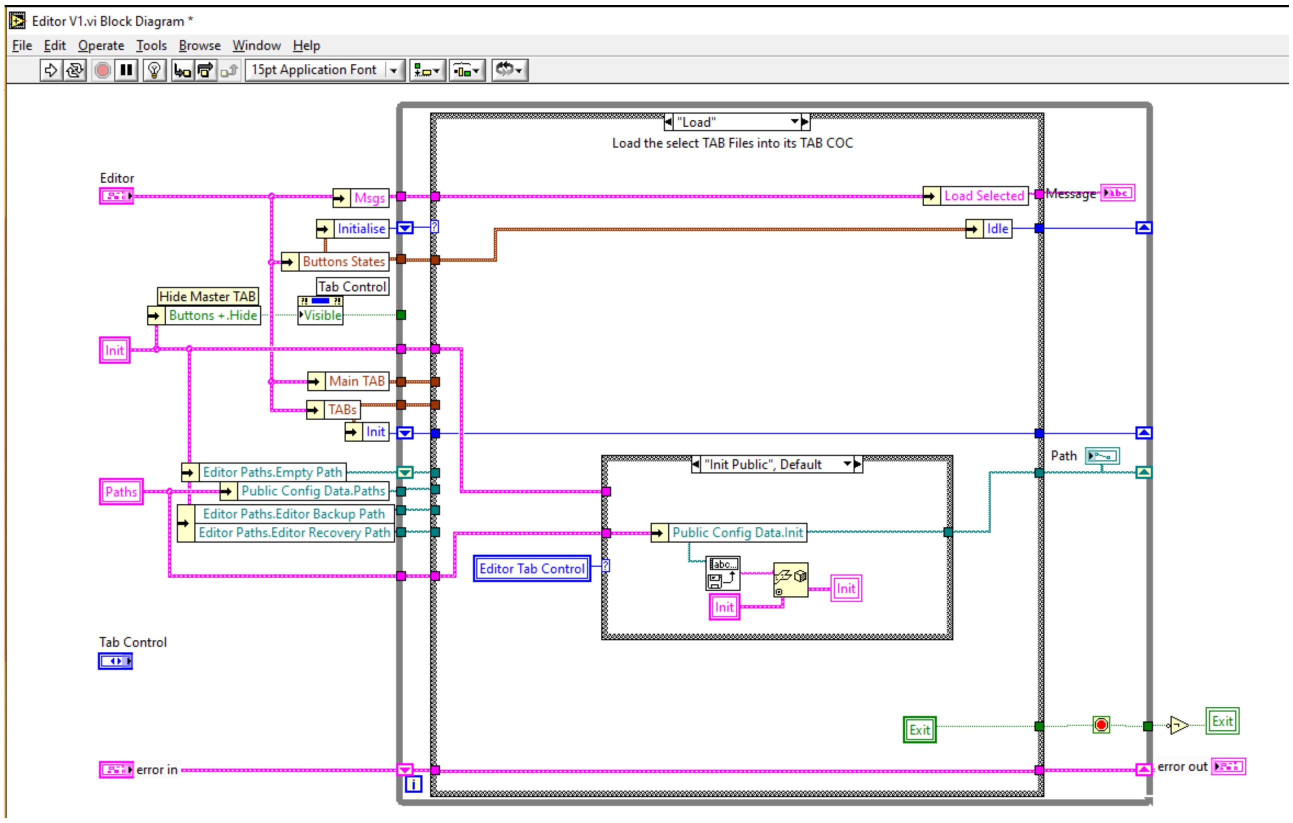
“Choose”> allows you bring up the Editor either “Not loading” the TABs with COC data or “Load them all” with COC data. You may need this if you have changed the COCs layouts.



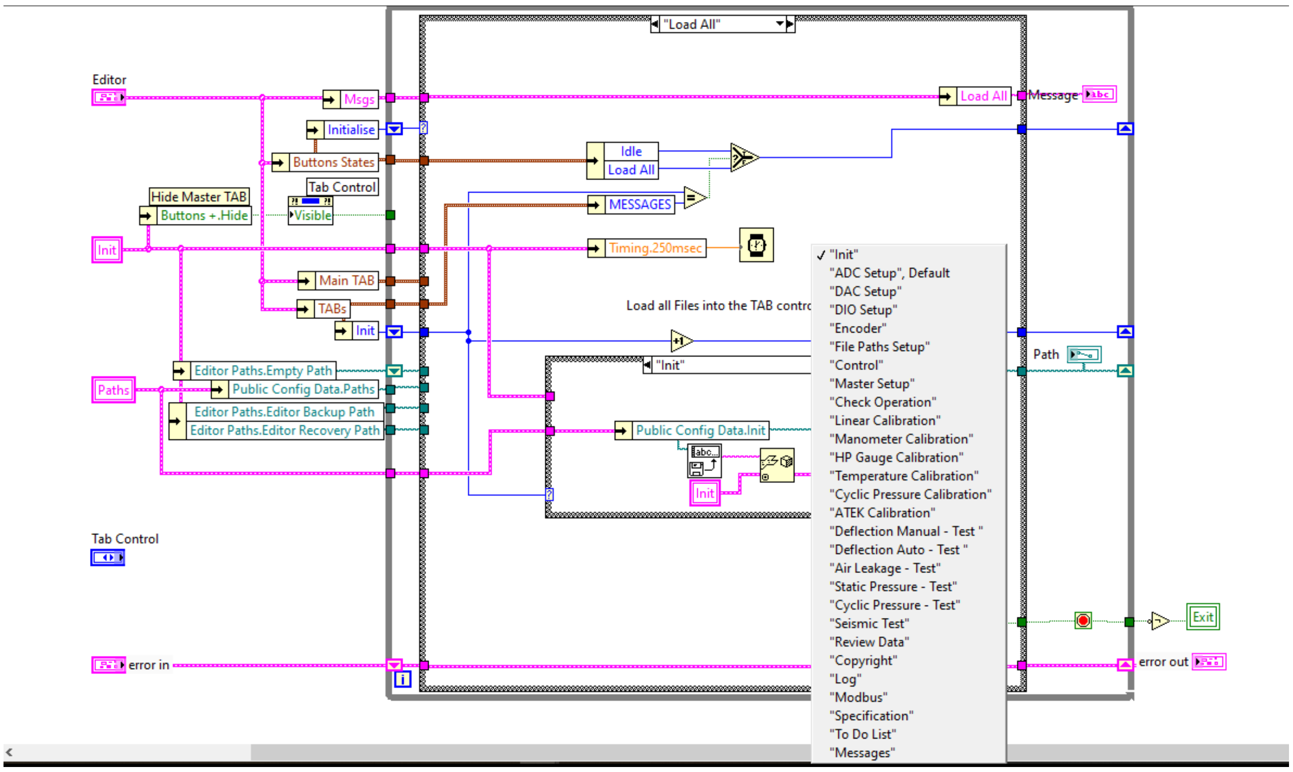
"Idle" > is where the new Event selector is and I have made it so it cannot get stuck with a timeout of 500msecs.



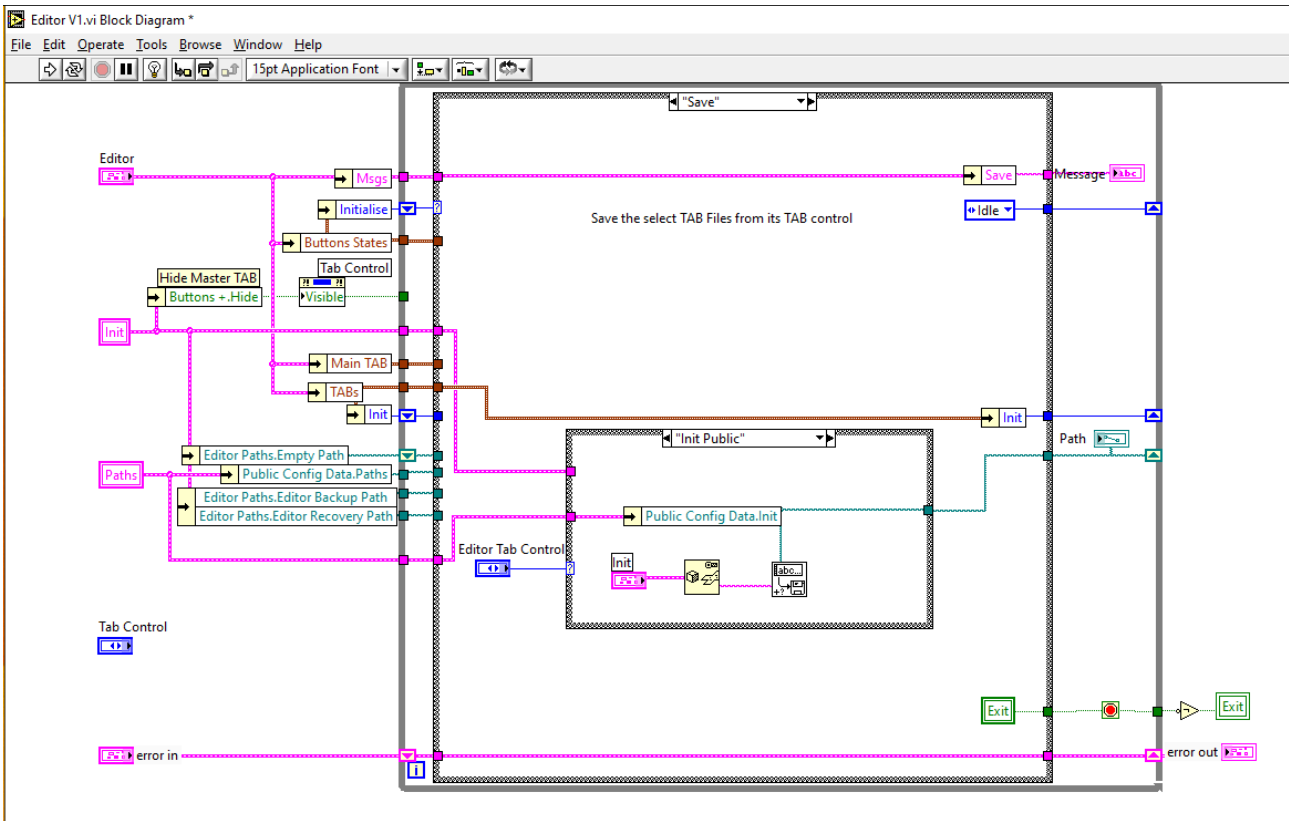
"Load"> loads only the COC data for the selected TAB that has been clicked.



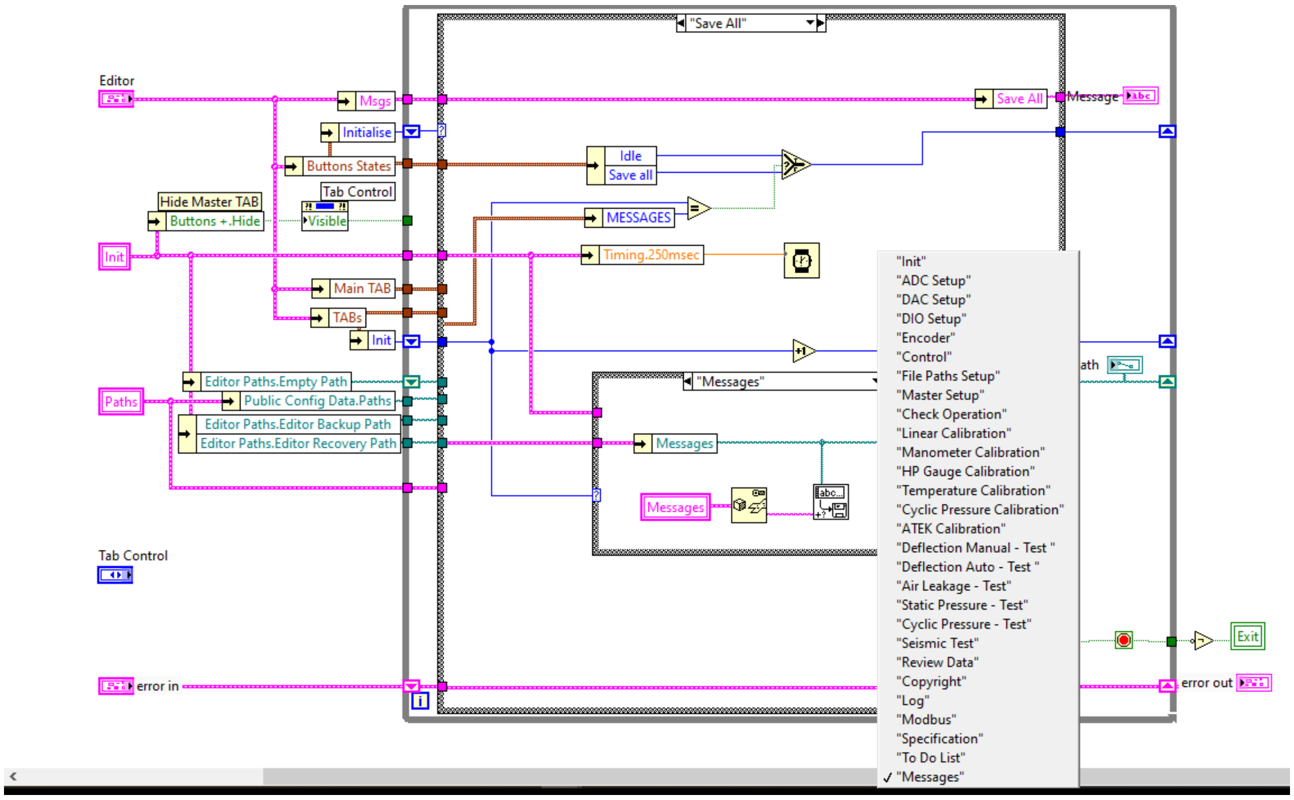
“Load All”> Loads all the COCs into the TABs.



“Save” > Like Load just saves the one COC selected from the TAB Selection.

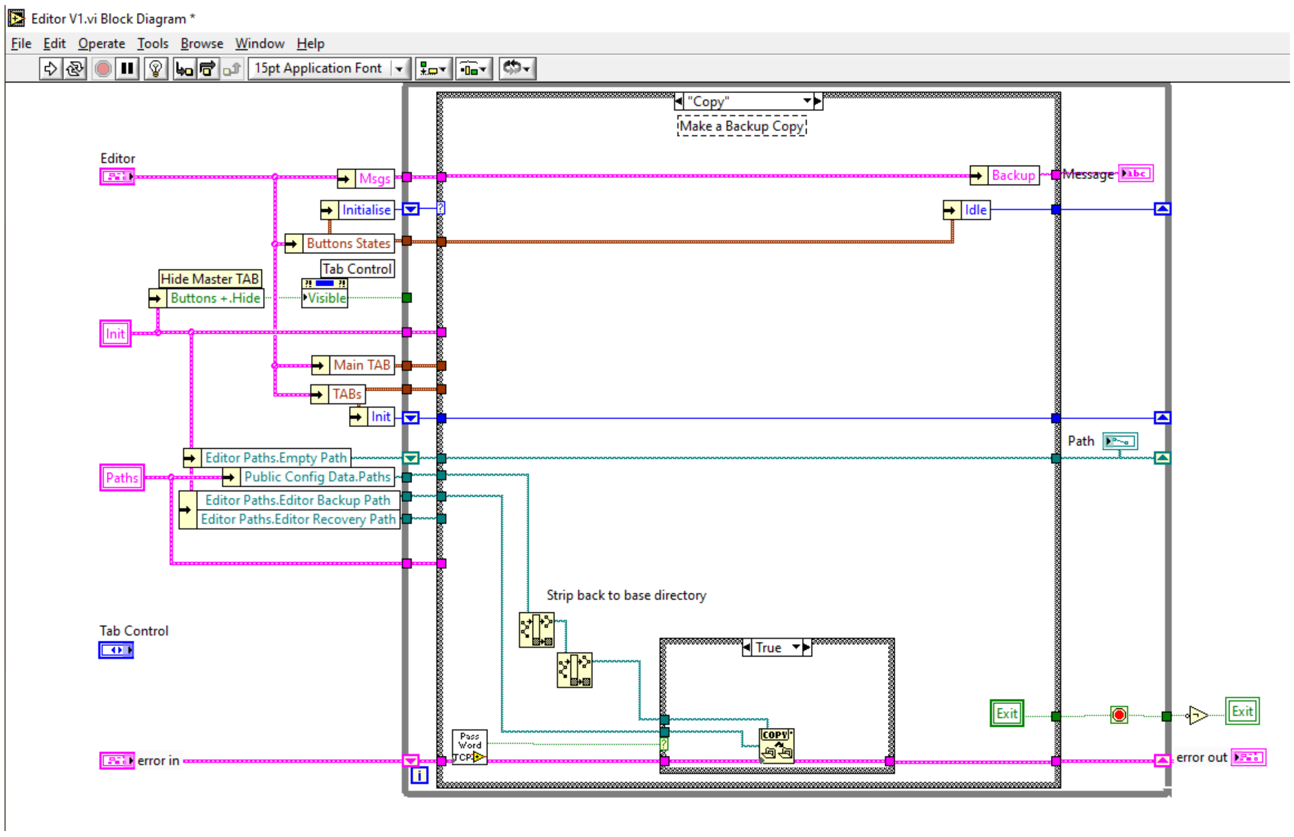


“Save All”> is like “Load All” saves all the TABS COCs.

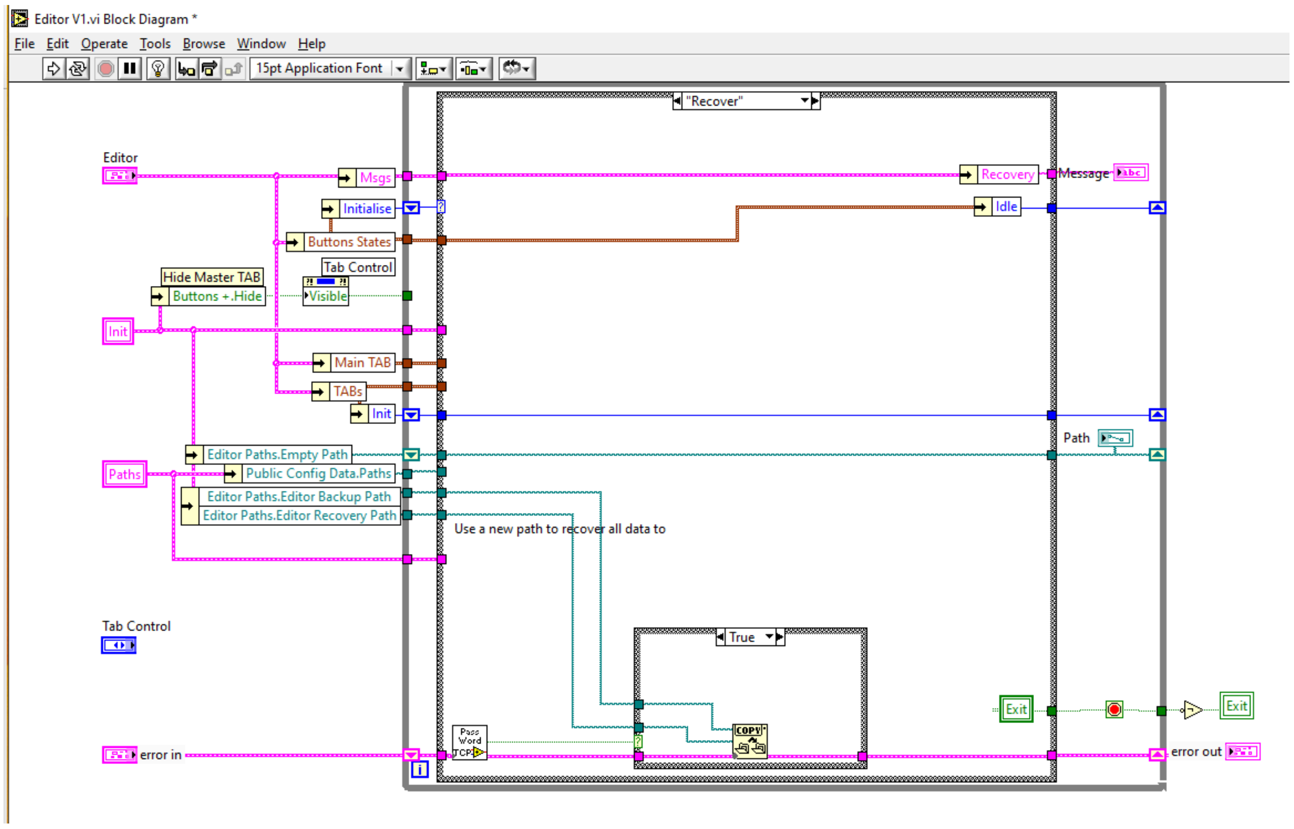


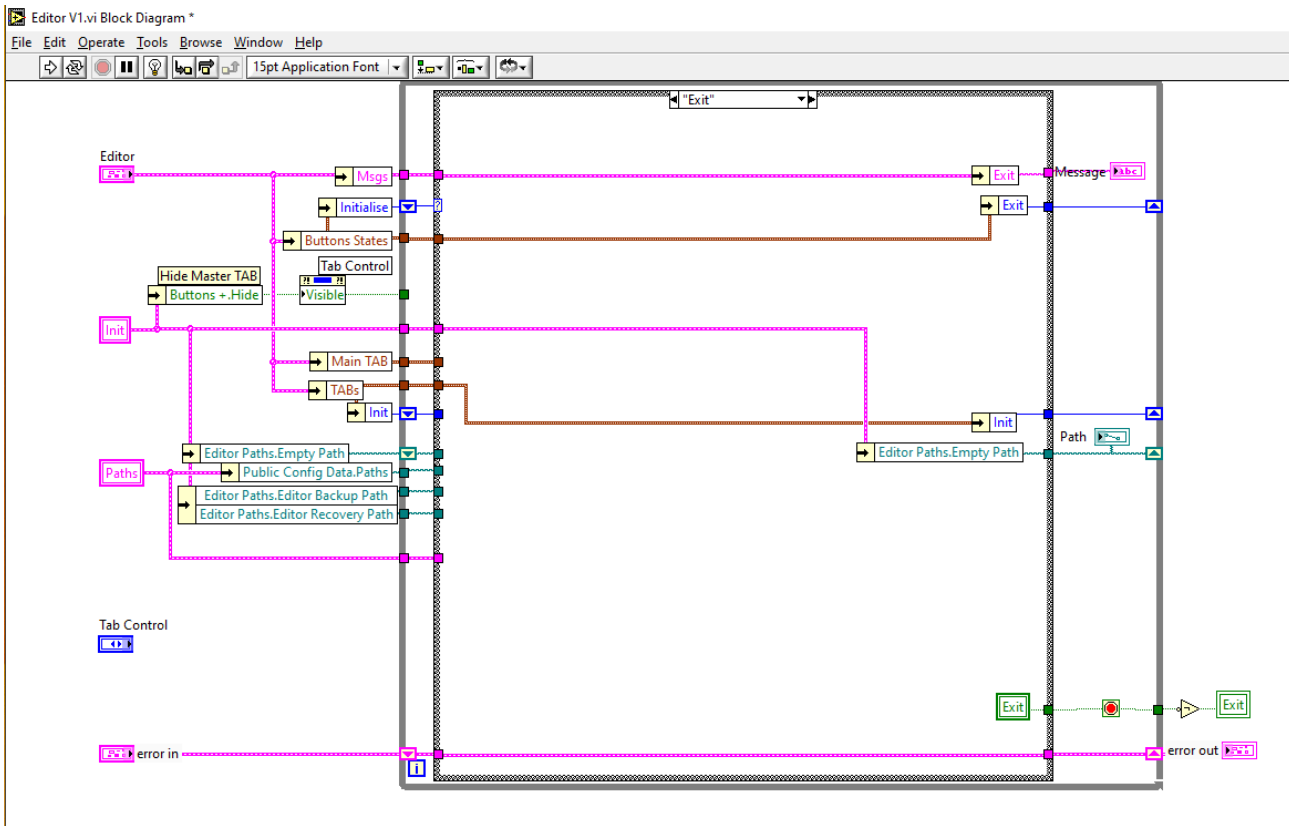
"Init"
"ADC Setup"
"DAC Setup"
"DIO Setup"
"Encoder"
"Control"
"File Paths Setup"
"Master Setup"
"Check Operation"
"Linear Calibration"
"Manometer Calibration"
"HP Gauge Calibration"
"Temperature Calibration"
"Cyclic Pressure Calibration"
"ATEK Calibration"
"Deflection Manual - Test "
"Deflection Auto - Test "
"Air Leakage - Test"
"Static Pressure - Test"
"Cyclic Pressure - Test"
"Seismic Test"
"Review Data"
"Copyright"
"Log"
"Modbus"
"Specification"
"To Do List"
✓ "Messages"

"Copy" > backs up the Apps Folder of Data including COCs, Calibration and Test data, to a new folder you choose by setting in the "Init" COC. These file paths could be in Paths but!! I figured you will only do this once so "Init" is best.


Editor V1.vi Block Diagram *

Recover" > does the reverse of "copy" except you can choose again where you put the recovery data.





So that is it. I could show more but that is enough to work with.

You will find that if you are building big programs this expanded LCOD COC methods will be very helpful.

It is house keeping stuff, just like in C Code, where you are forced to declare and define everything in detail.

I still have to complete a few things, like loading the Editor's COCs and will do this.

Now if you have any questions or thoughts do email me.

If I have more brain waves on this I will do another PDF.

All the best to you all.

Did you notice that the App expanded as I wrote this PDF up.
I was developing and expanding as I made this PDF.

Rob Maskell

www.lcodcoc.co.nz
www.mastec.co.nz
www.mastecnz.co.nz
www.jenlogix.co.nz
www.testequipment.co.nz
www.powerbackup.co.nz
www.medicaltech.co.nz
lcodcoc@gmail.com