Mexml

ME XML Analyzer
Users Manual

Joalah Designs LLC
January 2015

Table of Contents

Introduction

Overview

The ME XML Analyzer (`Mexml`) is a stand-alone program used to analyze Machine Edition projects with the aim of identifying and documenting:

- Potential programmer mistakes
- Program Block software interfaces
- Hardware and EGD interfaces
- Program call-trees
- Program file dependency order
- Usage and definition of all variables

It does this by processing various files exported from a Machine Edition project and building up an internal model of variable and program file usage. By generating selected reports from this internal model you can highlight different aspects of the Machine Edition project. By generating the same reports from different projects or even different versions of the same project, you can identify changing patterns of usage.

The types of potential programmer mistakes that `Mexml` can help identify include:

- Variables that have been read, but not written to.
- I/O that has been defined but not used.
- Writing/Reading data beyond a defined variable or address.
- Function Block variables that are global to all instances of a Function Block.
- Float and DINT variables that have not been spaced 2 registers apart.
- Aliased variables that are missing their parent variable.
- Variables that have been mapped on top of each other (which when intentional, can be used to document such things as bit mappings)

By processing only a selected group of program blocks, `Mexml` can be used to identify variables that are external to those blocks. This allows documenting the data that flows into or out of a group and hence identifies the software interface of those program blocks.

When `Mexml` processes a Machine Edition project's Hardware or EGD configuration files it can list how the program variables are mapped onto those resources. For the hardware it will list the rack, slot, card, I/O point address, along with variable data type, name and description. While for the EGD configurations it will list all of the produced and consumed exchanges (with IP addresses) along with the variable data type, name and description.

During the processing of each program block, `Mexml` will automatically keep track of the dependency order as well as whether the block has been called or not. This allows it to:

- Display the full dependency order list of the program blocks. This information is necessary to know when manually importing program blocks into another project.
- List the full call tree showing what program blocks are called and in what order, as well as what program blocks are not called anywhere in the program.

Note that currently `Mexml` can only process Ladder Logic and Structured Text program blocks.

Project Analysis

After analyzing a project, `Mexml` can produce two different types of reports: The Program Structure report and the Variables Usage report. The former identifies aspects related to the structure of program blocks and their instructions. The later identifies how variables are utilized by those program blocks and instructions.

Program Structure

The Program Structure report extracts information related to the internal structure of the Machine Edition project. This report includes items such as:

- List program blocks that are used/not used.
- List the order of program block dependency and calls.
- Identify variables that are mapped to the same address.
- Identify variables with overlapping addresses.
- Identify variables in Function Blocks (FB) that are global to that FB, and will be shared by all instances of that FB.
- Identify program instructions that access addresses outside of a particular variables size.
- List chains of variables that are aliased from one to another.
- Display a text representation of Ladder Logic and Structured Text instructions.

Variable Usage Report

The Variable Usage Report list all the variables referenced in project that meet any combination of 6 well defined criteria. These criteria are:

Access	Actions used to access a variable, such as Read, Write, being in an Input Scan or being in an Output scan, etc.
Memory	In what section of the PLC's memory that the variable is defined, such as %I, %Q, %M, %R, Symbolic, etc.
Type	The type variable such as Local, Global, Function Block Input/Output, etc.
Name	The name of the variable as defined by a regular expression.
Address	The address of the variable as defined by a regular expression.
Description	The description of the variable as defined by a regular expression.

These criteria enable the selection of the variables included in the report to be very specific. For example, the report could only include all variables that:

- Are a part of an Input scan,
- (and) That have not been Read by any instruction,
- (and) That have a %M Address,
- (and) Are defined as a Local variable of a program block,
- (and) Have a Name that starts with “Gantry”,
- (and) With an Address in the range of %M1000 to %M1999,
- (and) Have the text “Fred” in the middle of the Description.

Report Examples

The following are two reports that were generated from the sample project included with the Mexml program. Both of these reports were generated by the included VarAnalyisis command. The first report details any structural issues with the target project, while the second specifically lists variables that have not been referenced — including Inputs not read, Outputs not written to, and other variables that have been read but never initialized.

Program Structure Report

The VarAnalysis Program Structure report looks at the structure of the project. In this particular example, the options selected for the report included displaying warning and error messages related to the program code itself, and also displaying the call order of the program blocks.

This report discovered two definite errors, as well as two items that could potentially be errors depending on the intention of the programmer. The two errors are:

- A data transfer in POU AABlock will exceed the size of the variable SmallArray
- Variables Number2 and Number1 overlap in their addresses.

The two potential errors are:

- The variable TEMP2 is not local to the Bfunction function block, and is a global variable.
- Program block EEBlock is not called anywhere in the program.

None of these issues are detected by Machine Edition.

The text of the report is:

```
Mexml version 2.0.48.14195
Started processing at 12/30/2014 2:39:30 PM
8 out of a total of 8 POUs will be processed
** ERR  In POU AABlock, rung 7, Data transfer will exceed size of Variable SmallArray↩
        by 4 INT
++ Warn In POU BFunction Interface, Local Variable TEMP2 is not a member of the↩
        Function Block and will be shared with all BFunction instances
** ERR  In Propagating mapped address variables, Number2, DINT, %R00022 overlaps↩
        Number1, DINT, %R00021

##################
  POU Call order
##################

  ============================
    Blocks that are not called
  ============================
    EEBlock
```

```
  ==========================
    Blocks that are called
  ==========================
    _MAIN
      Setup
      CCBlock
        AABlock
        BFUNCTION
      DDBlock
        AABlock
        BFUNCTION
      BBBlock

Finished processing all of the reports

Finished processing at 12/30/2014 2:39:31 PM
```

Variable Usage Report

The `VarAnalysis Variables` report describes how variables within the project were defined and accessed. EG Defined as `Inputs`, `Outputs` or `Internal`, and if they were `Read` or `Written`. The `Variables` report does not explicitly identify problems with particular variables, instead the report can be used to filter for particular patterns in how variables were defined and accessed, and hence highlight areas within a program could be problematical. In this particular example, the report has been defined to only list out variables that are either:

- Not part of the `Input Scan`, and have been `Read`, but not `Written`. Such variables are potentially uninitialized.
- A part of the `Input Scan`, but have not been `Read`. Such variables are `Input I/O` that have not been used.
- A part of the `Output Scan`, but have not been `Written`. Such variables are `Output I/O` that have not been used.

In the following example:

- `BoolDataIn[2]` is an `Input I/O` variable (in this case an `EGD` signal) that has not been `Read`.
- `BoolDataOut[3]` is an `Output I/O` variable (in this case an `EGD` signal) that has not been `Written`.
- `Data_5` is an internal variable that has been `Read`, but has not been `Written` to by any part of the project.

This can be inferred from the entries in the `Access` column, where an alphabetic entry in a particular numbered column indicates a characteristic, while a dash indicates lack of that same characteristic.

- `I` in column 1 indicates an `Input` definition (either physical or EGD).

- R in column 2 indicates that the variable was Read.
- W in column 3 indicates that the variable was Written.
- Q in column 7 indicates an Output definition (either physical or EGD).

The text of the report is (with about 160 variables removed for clarity):

```
#######################################################
  Variables Report, created at 12/30/2014 2:39:31 PM
#######################################################

  ====================
    Report Parameters
  ====================

    ~~~~~~~~~~~~~~~~~
      Source Files
    ~~~~~~~~~~~~~~~~~
      Program:         J:\Mexml\Example\Mexml\MexmlSample\ExportedFiles\Code\*.xml
      Hardware Configuration:  ↩
                       J:\Mexml\Example\Mexml\MexmlSample\ExportedFiles\Config\*.hwc
      EGD Configuration:  ↩
                       J:\Mexml\Example\Mexml\MexmlSample\ExportedFiles\Config\*.egd
      Block Interface: J:\Mexml\Example\Mexml\_Mexml\LibraryDefs\*.txt
      Preload list:    J:\Mexml\Example\Mexml\MexmlSample\ExportedFiles\Variables\*.xml
      Variable Force:  J:\Mexml\Example\Mexml\MexmlSample\LocalFiles\Forces\*.txt

    ~~~~~~~~~~~~~~~~~~~~~
      Filter Parameters
    ~~~~~~~~~~~~~~~~~~~~~
      Name Regex:       -fRN ^[^\#]
      Access Criteria:  -fA iwR/Ir/Qw
                            iRw [Not InputScan] [Read] [Not Write]
                         or Ir [InputScan] [Not Read]
                         or Qw [OutputScan] [Not Write]

    ==============
      Sort Order
    ==============
      (Default to By Name)

    =========
      Other
    =========
      Selected Columns: -rCS NDACE   ([Name] [Data Type] [Address] [Access Criteria]↩
                                      [Description])


#######################################################
                     Report Data
#######################################################

                                      Access
Name                 Type     Address 1234567 Description
-------------------- -------- ------- ------- ---------------------------------
BoolDataIn           BOOL[8]         I------ Data in from EGD exchange
BoolDataIn[2]        BOOL            I------ EGD data in bit 2
BoolDataIn[4]        BOOL            I------ EGD data in bit 4
BoolDataIn2          BOOL            -R----- EGD data in bit 2
```

```
BoolDataIn4          BOOL             -R----- EGD data in bit 4
BoolDataOut          BOOL[8]          ------Q Data sent to remote EGD exchange
BoolDataOut[3]       BOOL             ------Q EGD data out bit 3
BoolDataOut[6]       BOOL             ------Q EGD data out bit 6
Data_5               INT      %R00154 -R----- Data number 5
Data_6               INT      %R00155 -R----- Data number 6
DDBlock.CLR_Fancy    BOOL             -R----- Clear the function
EGDConsExchgStatus   WORD             I------ EGD consumed exchange 1 status
EGDProdExchgStatus   WORD             ------Q EGD produced exchange 1 status
...
IntDataIn            INT[2]           I------ Data in from EGD exchange
IntDataIn[1]         INT              I------
IntDataOut           INT[2]           ------Q Data sent to remote EGD exchange
IntDataOut[1]        INT              ------Q
LostAtSea            BOOL     %Q00002 ------Q Operator has been lost at sea
Number1              DINT     %R00021 -R----- This is number 1
Number2              DINT     %R00022 -R----- This is number 2
Q00008               BOOL     %Q00008 ------Q

183 variables matched the selection criteria
```

Licensing

The Mexml program requires a valid license in order to run, however the method used to license it is straight forward and non-intrusive, and only involves copying the license file to the Mexml installation directory, and does not require any further or ongoing external Internet connection.

There are several types of licenses available for Mexml:

Trial Enables a subset of Mexml functionality, and is will be in compliance for all versions of Mexml and for all time. The standard Mexml installation comes with a trial license by default.

Standard Enables the full set of Mexml functionality, but is limited to the version of Mexml that it was created for, plus two major versions. However the license will never expire. This type of license can be purchased directly from Joalah Designs LLC

Special Enables features of Mexml on a custom basis of time and version. EG Enabling all features for only a 2 week period. This type of license can be obtained from Joalah Designs LLC on a case by case basis.

To expand on the Standard license compliance with an example. If a license for v3.1 of Mexml is purchased, then that license remains valid for versions v3.2, v3.3, .. , v4.0, v4.1, .. , v4.9, v5.0, v5.1 of Mexml. The license will not be valid for v5.2 onwards. The license will however remain valid for all versions of Mexml less than v5.2, and will never expire. Finally, there is a significant discount for renewing an Mexml license.

When copying a new license to the Mexml installation directory, you do not have to delete or remove any previous licenses. Mexml will scan all the available licenses and automatically use the most permissive license.

The following tables list the functionality available with the Trial and Standard licenses:

Source Files

These items specify what data Mexml will process:

Source File Items	Trial	Standard
Program Block files	X	X
EGD Definition files		X
Hardware Definition files		X
Additional pre-loaded lists of variables		X
Lists of variables forced to particular Access Modes		X
Function Block Interface definitions		X

Source File Processing

These items specify how the source data will be processed:

Source Processing Items	Trial	Standard
Skip analysis of Program Blocks		X
Skip processing Function Blocks		X
Give auto-generated variables a blank description		X

Source File Messages

These items specify the messages that will be generated in the Program Structure report:

Source Message Items	Trial	Standard
Generate all messages		X
Suppress Warning messages		X
Suppress all messages		X
Filter messages by Program Block name		X
Ladder Logic basic messages	X	X
Ladder Logic Derived and UDT data type messages		X
Ladder Logic annotate instructions with their ID number		X
Ladder Logic Suppress rung numbers		X
Structured Text basic messages	X	X
Structured Text Derived and UDT data type messages		X
Structured Text annotate code with Read/Write markers		X
List order that Mexml processes source files		X
EGD Definition file messages		X
Hardware Definition file messages		X
Pre-loaded Variable file messages		X
Variable Force file messages		X
Function Block Interface definition file messages		X
Linked variable messages		X
List the generated Variable Reports	X	X
Specify output file for source messages and reports		X

Program Structure Reports

These items specify additional information included in the Program Structure report:

Program Structure Items	Trial	Standard
List Program Block call order		X
List Mapped Address variables		X

Variable Report Generation

These items specify how the Variables Report is generated:

Variable Report Items	Trial	Standard
Generate Variable Access reports	X	X
Specify the name of the Variable Access report	X	X
Specify the directory of the Variable Access reports		X
Define the Variable report parameters in an external file		X
Filter Variables by Access (Input, Output, Read, Write etc)		X
Filter Variables by Memory (Input, Output, Memory etc)	X	X
Filter Variables by Name		X
Filter Variables by Address		X
Filter Variables by Description		X
Filter Variables by Type (Local, Global etc)	X	X
Choose sort order of the Variables in the report		X
Select the columns displayed in the report	X	X
Select the width of the columns in the report	X	X
Generate the report as a CSV file		X
Explain the columns in the report	X	X

Other

These items specify displaying `Mexml` version and licensing information:

Other Items	Trial	Standard
Display the Mexml version	X	X

Other Items	Trial	Standard
Display the Mexml license currently being used	X	X
Display available Mexml licenses	X	X
Display Mexml help	X	X

Validation

In addition to using `Mexml` to determine the validity of a license, the `Mexml` companion application `LicScan` can be used to directly validate `Mexml` licenses. This application inspects a particular license and lists all of the information contained in it. It also allows for the compliance of the license to be tested against an arbitrary version number.

The `LicScan` program is available separately as Windows `.msi` installation file.

Installation

Program

Overview

`Mexml` is distributed as a standard Windows Installer file (`.msi`), so that installation (and removal) of the program is very easy. However this does assume that you have the requisite Windows permissions to install software on the target computer.

The installer file will be typically named like:

```
MexmlApplication.2.1.0.7.msi
```

Where the numbers after the `MexmlApplication` name represent the version of `Mexml` that will be installed. These numbers are encoded as:

```
MajorVersion.MinorVersion.Release.BuildNumber
```

So that using the example above, the details are:

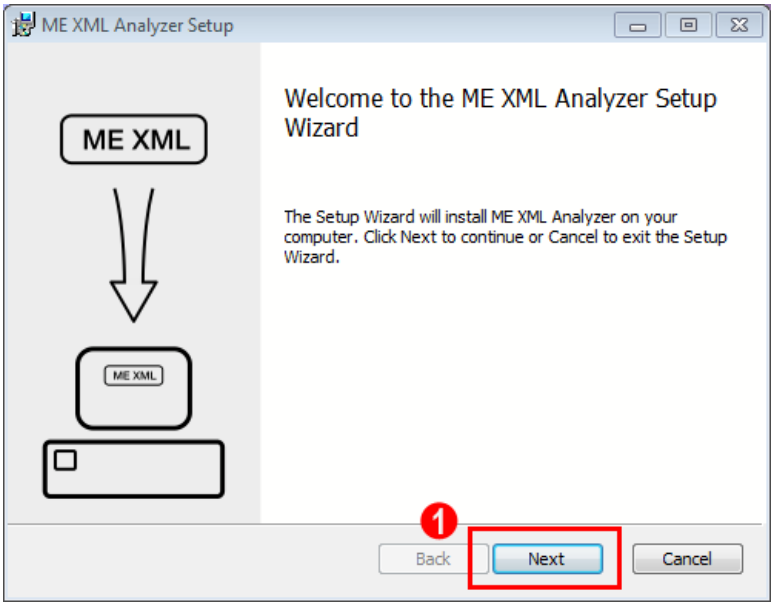
```
MajorVersion    2
MinorVersion    1
Release         0
BuildNumber     7
```

The `MajorVersion/MinorVersion` values are used to determine license conformance. The `Release` and `BuildNumber` are not used for with the licensing and are internal designations of a particular `Mexml` version.

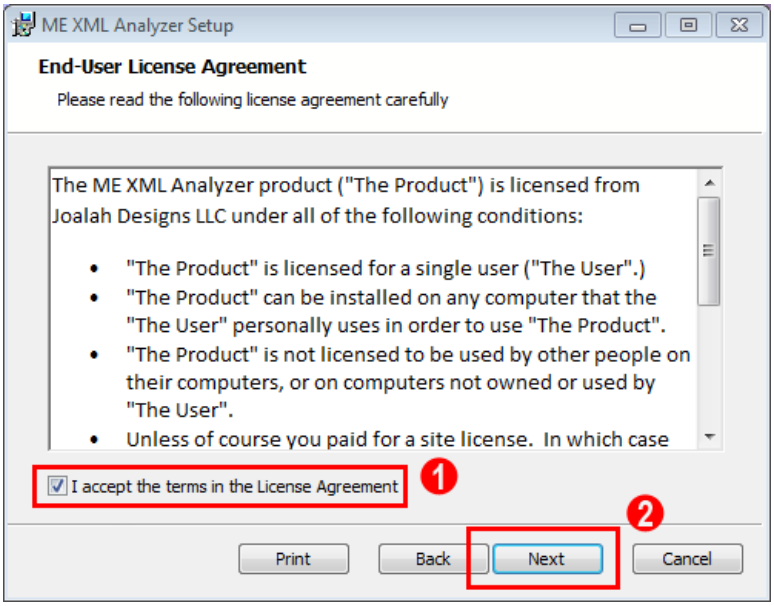
Finally the distributed `.msi` file (as well as the actual `Mexml` application) is digitally signed by Joalah Designs LLC in order to assert the provenance of the file.

Process

Opening up the `Mexml .msi` file will start the installation process, and will present the first dialog box:

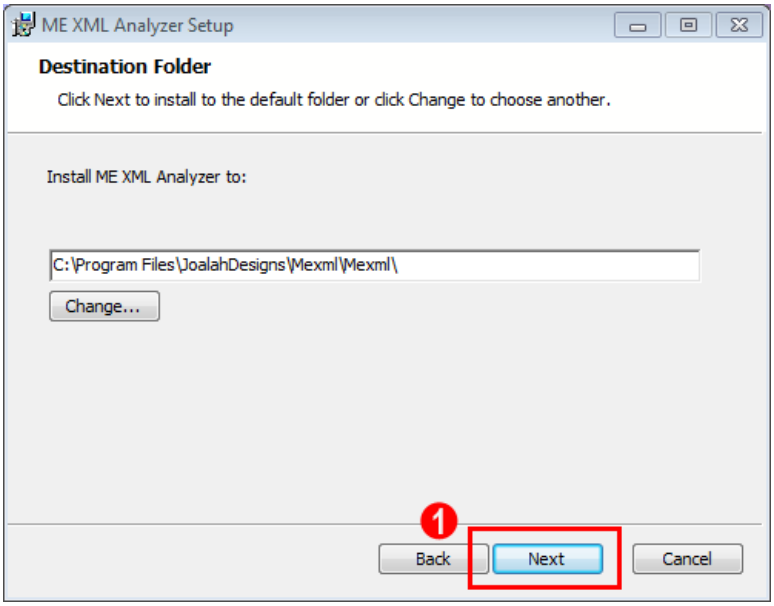


By clicking on '`Next`', the `Mexml` license agreement is presented. Installation cannot continue unless this license is agreed to (by checking the '`Accept terms`' checkbox):

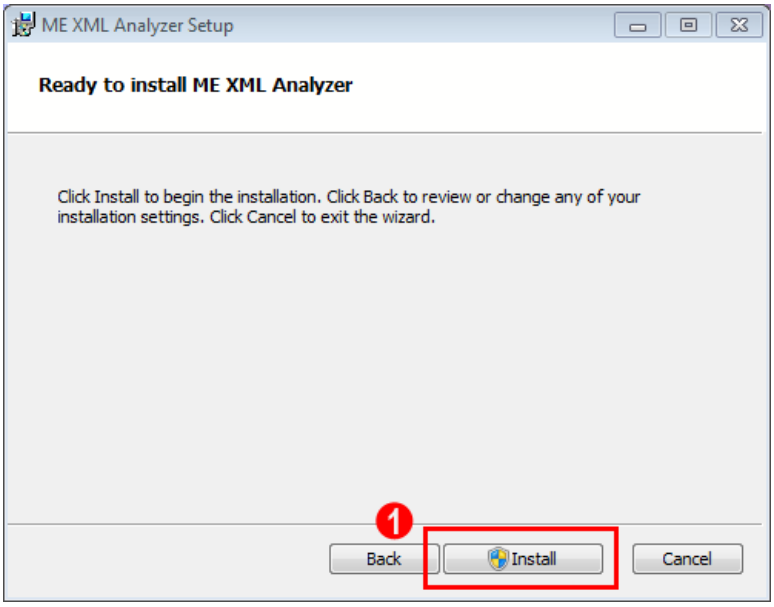


After agreeing to the license terms, and then clicking on 'Next', the location to where Mexml will be installed to, is displayed. It is recommended that this default location should be utilized. Note that the location shown in the example is where Mexml will be installed on a 32-bit Windows system. On a 64-bit system, it will be installed to:

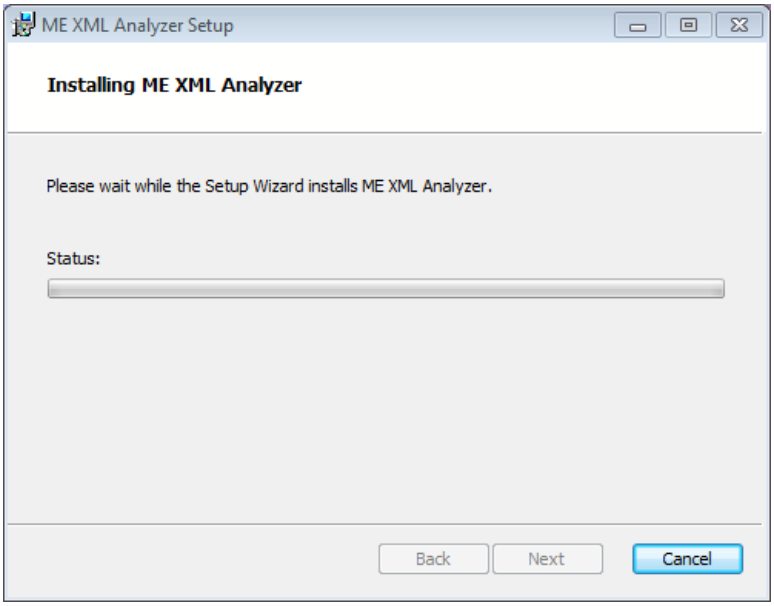
```
C:\Program Files (x86)\JoalahDesigns\Mexml\Mexml
```



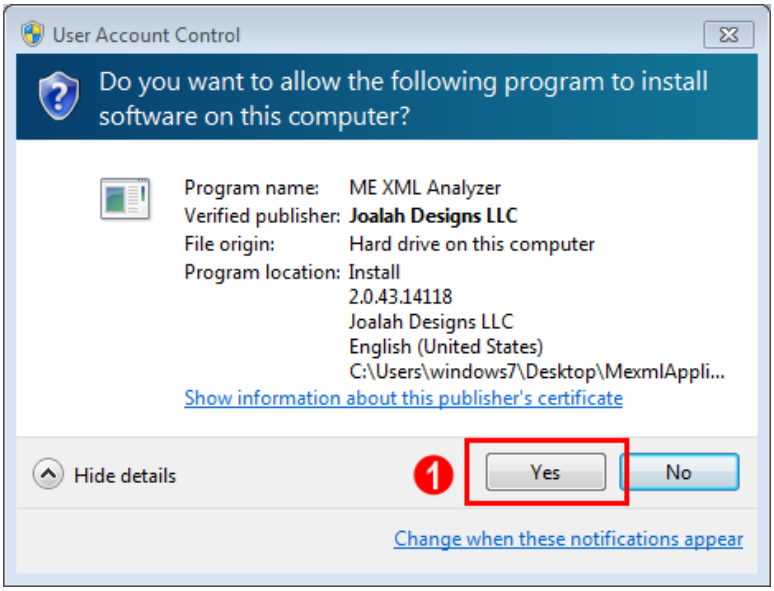
After the installation location has been selected, and 'Next' clicked on, Mexml is now ready to install:



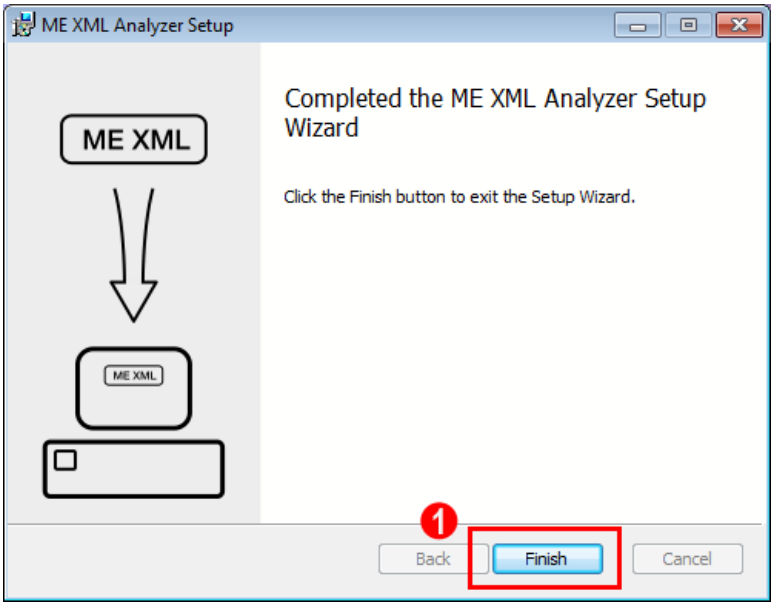
When '`Install`' is clicked, then actual process of installing the `Mexml` files will begin:



However, depending on the permissions and the local policies that are in effect, Windows may ask the for confirmation of the installation prior to actually installing the `Mexml` files:



After the installation process has completed, the final confirmation dialog is displayed. At this point `Mexml` is ready to be used, (however additional licenses may need to be installed before the full set of `Mexml` features can be used):



Installed Files

The Mexml installation process installs the following typical files. Note that it also adjusts the Windows 'PATH' environment variable to include the Mexml execution path, which allows Mexml to be called directly from a Windows command line.

```
DotNetZip_License.rtf
Ionic.Zip.dll
Joalah_PublicKey.xml
log4net.dll
Log4Net_License.txt
Mexml.exe
Mexml.exe.config
Mexml_License.rtf
Mexml_License_Trial_JoalahDesigns_Never.xml
Rhino.Licensing.dll
Rhino_License.txt
```

These files include:

- The code required for Mexml's functionality.
- Joalah Design's Public Key file (required for operation of the licensing).
- Text of all agreed to licenses, including those of 3rd the party modules used by Mexml (DotNetZip, Ionic.zip, Log4net and Rhino.Licensing), as required by their usage terms.
- A non-expiring Mexml trial license.

Licenses

Trial

Mexml is supplied with a trial license that only enables a subset of the program's full functionality. This license is will never expire and will be valid for any version of Mexml released in the future. It is installed during the Mexml installation process and named:

```
Mexml_License_Trial_JoalahDesigns_Never.xml
```

Standard

In order to use the full functionality of Mexml a separate, standard license will need to be purchased, and then copied to the same location as the trial license (which is typically the folder in which the Mexml executable resides).

Currently, licenses can only be purchased by directly contacting Joalah Designs LLC by sending an email to info@JoalahDesigns.com

Once a license is purchased, it will be delivered as a zipped up XML file, with a name such as:

```
Mexml_License_Standard_[Owners Name]_[Time].zip
```

Where '[Owners Name]' represents the name of the person or organization who purchased the license. The '[Time]' represents when the license will expire, such as 'Never', or if it is time limited, the date at which it will expire. Note that typically all Mexml licenses will never expire in time, but will be limited in version only. Time limited licenses will only be generated under special circumstances.

In order to install the license, simply unzip the file to extract the actual XML license file:

```
Mexml_License_Standard_[Owners Name]_[Time].xml
```

Then copy this license file to the folder where Mexml was installed. Once this is done, Mexml will automatically utilize the new license.

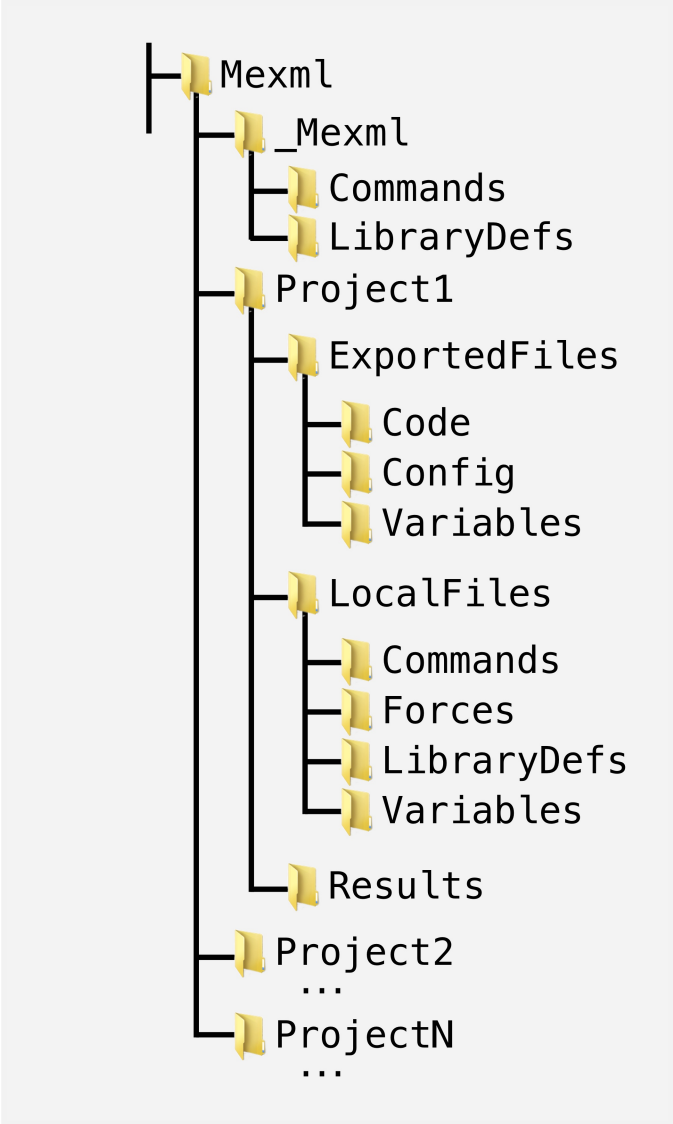
Note that when a new license is copied to the Mexml installation directory there is no need to delete any previous licenses. Mexml will automatically scan all licenses that it finds, and then use the license that is the least restrictive.

Preparation For Use

Directory setup

When using the `Mexml` program, the files that are to be processed are specified on the command line. As such these files can be stored in any folder that the program can access, which while a powerful capability, does open up the possibility of complex and hard to understand configurations. In order to reduce complexity it is recommended that a 'regular' directory structure be adopted to hold the ME projects that `Mexml` will process. Using a structured approach not only reduces complexity, it also facilitates the development of batch files that can be used to invoke `Mexml` to analyze many different projects using the same `Mexml` parameters.

The following sections describe a recommended directory structure that has proven useful for analyzing multiple projects:



The above directory structure can be split into two basic parts: The `_Mexml` section and the `Project` sections. The `_Mexml` section is used to contain items that will be common to analyzing all ME projects, while the `Project` sections contain items that are common to that specific project.

_Mexml

The `_Mexml` directory is intended to hold items that are common to analyzing each project.

Commands

The commands directory holds command (batch) files that invoke `Mexml` in order to analyze a project.

Through judicious use of command file arguments and other options it is possible to construct a command file that can analyze specific projects when invoked, and to save the results back to that project. The `Analysis Command File` section of this document will discuss in detail such a command file.

LibraryDefs

The `LibraryDefs` directory holds the text files that would be passed into `Mexml` via the `-I <InterfaceFile(s)>` command line option. Files located here would be available for use in analyzing any of the exported Machine Edition projects.

Project

The `Project` directory contains a series of three sub-directories that hold the data and results related to a single Machine Edition project analysis project.

Note that this directory is only named Project here as an example. In real life the directory should be named after the Machine Edition project that it is related to.

Using the suggested Analysis Command File and directory structure, there is no limit to the number of project directories that can be set up, as long as they are all uniquely named.

ExportedFiles

The `ExportedFiles` directory contains a series of three sub-directories that hold the raw files that have been exported from a Machine Edition project, with different types of files being exported to their own directory.

These sub-directories and their contents are:

Code	Program files (`*.xml`) that contain the project's logic and would be passed into `Mexml` via the `-X <ProgramFile(s)>` command line argument.
Config	EGD (`*.egd`) and Hardware (`*.hwc`) configuration files that define the external connectivity of the PLC, and that would be passed into `Mexml` via the `–E <EGDFile>` and `-H <HardwareFile>` command line options.
Variables	Files of variables (`*.xml`) that have been exported from the project, and would be passed into `Mexml` via the `–P <PreloadFile(s)>` command line option.

LocalFiles

The LocalFiles directory contains a series of four sub-directories that hold analysis information that has been explicitly created by the user in order to aid analyzing this particular Machine Edition project.

Commands	Similar to the Commands directory in the `_Mexml` section, this directory holds command files that invoke `Mexml` in order to analyze a project. However these files are expected to perform project specific analysis.
Forces	The Forces directory holds the text files that would be passed into `Mexml` via the `–V <VarForceFile(s)>` command line option. Files located here would be specific to this particular project.
LibraryDefs	Similar to the Commands directory in the `_Mexml` section, this directory holds text files that would be passed into `Mexml` via the `–I <InterfaceFile(s)>` command line option. Files located here would be specific to this particular project.
Variables	Similar to the Variables directory in the `ExportedFiles` section, this holds variable files (`*.xml`) that would be passed into `Mexml` via the `–P <PreloadFile(s)>` command line option.

Results

The Results directory is the location where analysis data generated by `Mexml` will be saved to. The files that will be written here depend on what actions are performed by the analysis commands. Typically these files will include a program report (which reports on the actions of the analysis and the structure of the analyzed project) and a variable report (which reports on the read/write state of selected variables) for each command.

Exporting Files From Machine Edition

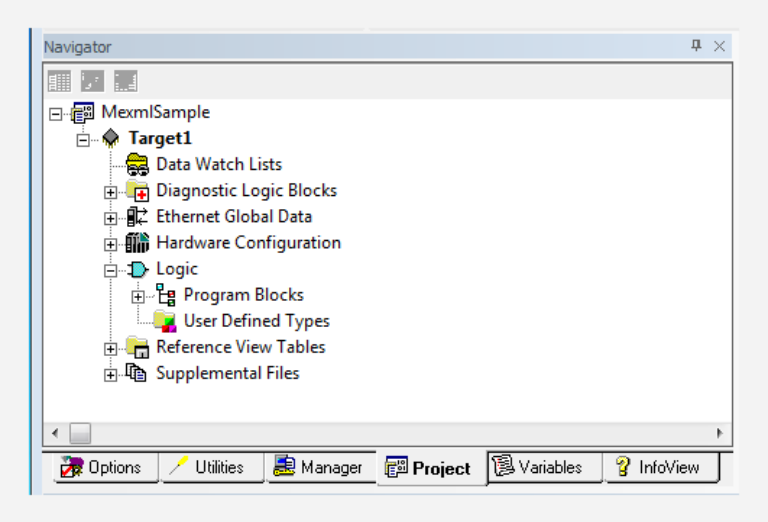
In order to utilize all of the functionality of `Mexml` when analyzing a Machine Edition project, several different types of data have to be exported from that project. The following sections show what data should be exported and where this is done within Machine Edition.

The examples shown will use the sample project `MexmlSample` to illustrate the steps needed to export each type of data. In addition, the exported data will be stored in the previously recommended data structure in order to better re-use analysis commands.

The `MexmlSample` project is described in detail in a later section, but for the purpose of exporting the data required by `Mexml` all that is needed to be known about it is that it:

- Is a simple Rx3i PLC project, with a single rack populated with a PSU, CPU, 2 I/O cards and an Ethernet card.
- Includes 8 program blocks (both Ladder Logic and Structured Text), including one configured as a Function Block.
- Defines 2 EGD exchanges that produce and consume data.
- Defines over 300 variables that are both local and global to the program blocks.

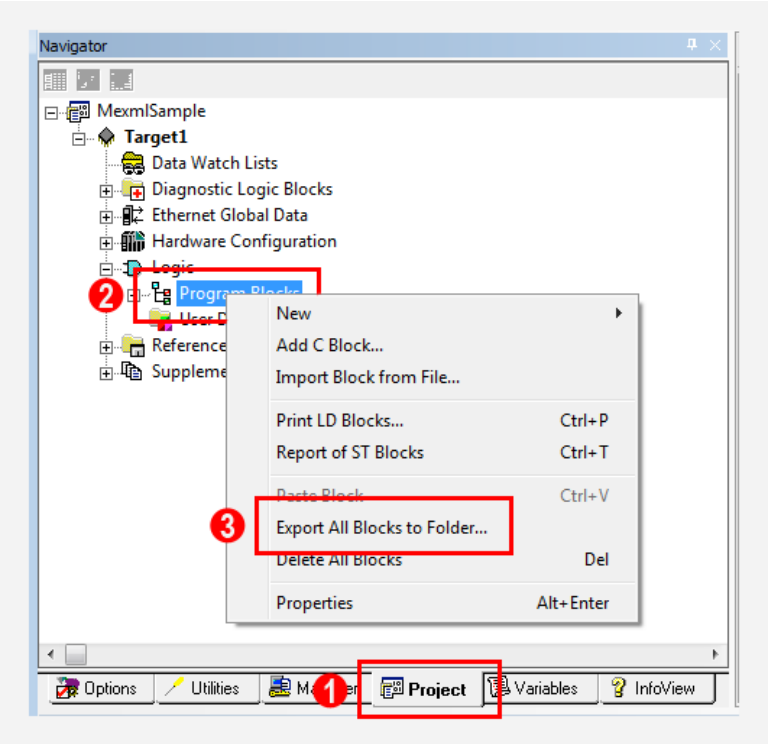
This represents the minimal project that can illustrate all of Mexml's features. When viewing this project in Machine Edition's navigator it will appear as:



Program Blocks

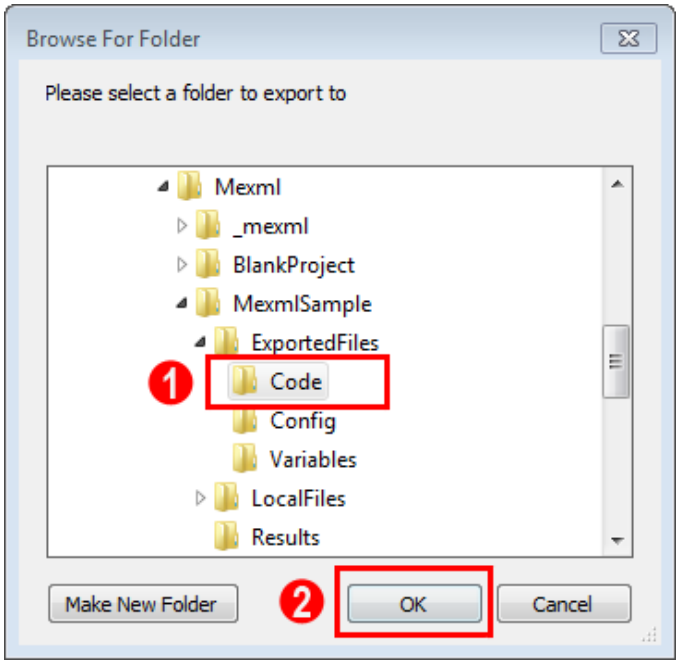
The main use of `Mexml` is to analyze the Program Blocks of a Machine Edition project. The actions required to export these blocks start with (after opening the project in Machine Edition):

1. Select the "`Project`" tab within the "`Navigator`" window
2. Expand the "`Logic`" section and `Right Click` on the "`Program Blocks`" entry.
3. Then click on the "`Export All Blocks to Folder ..`" item.



Clicking on the "`Export All Blocks to Folder ..`" item will open up a dialog box that is used to select the location to where the exported files will be saved. In keeping in line with the recommended directory structure for `Mexml` projects, use this dialog box to:

1. Select the "`Code`" directory underneath the "`ExportedFiles`" directory of the `Mexml` project.
2. Click on/select the "`OK`" button.



When “OK” is selected, Machine Edition will export all of the Program Blocks to the selected directory.

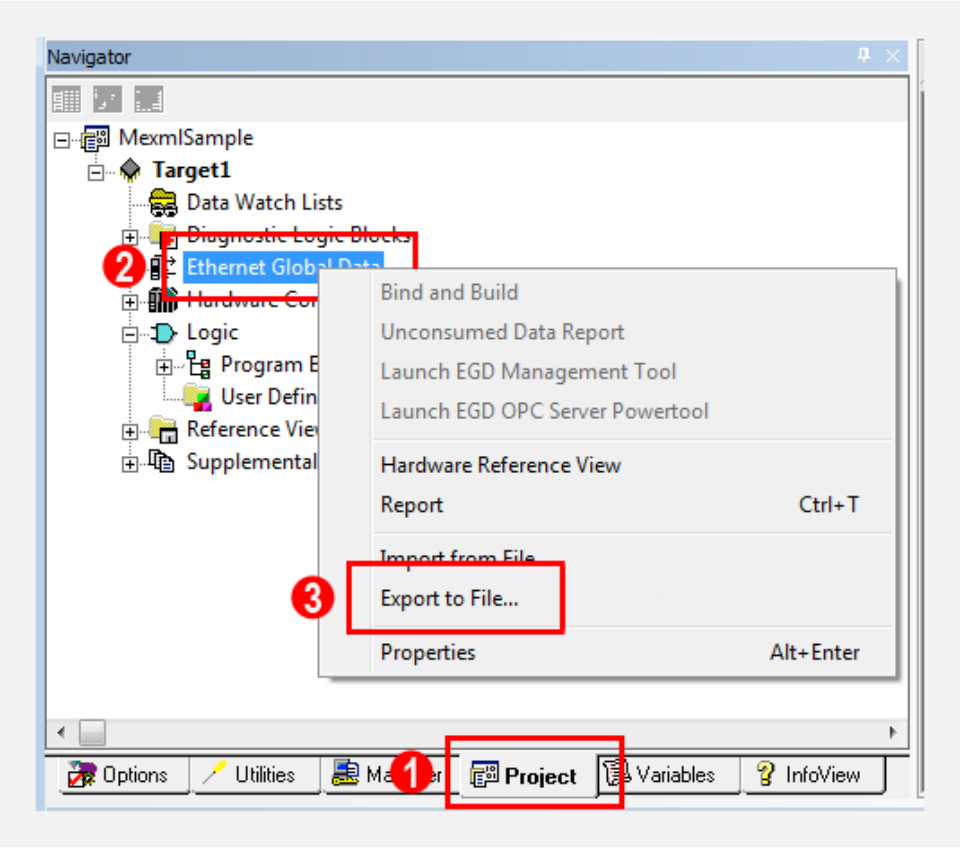
EGD Configuration

If a project uses EGD to transfer data, the information contained in the EGD configuration can optionally be used by `Mexml` in two different ways:

1. Mark the variables referenced in each EGD exchange as either being included in an Input or Output I/O scan, depending on if that exchange is consuming or producing data. This assists in determining whether a variable has been read or written by the project's logic.
2. List out the configuration of each EGD exchange in a manner not provided by Machine Edition. This aids the user in documenting the flow of data into and out of a project.

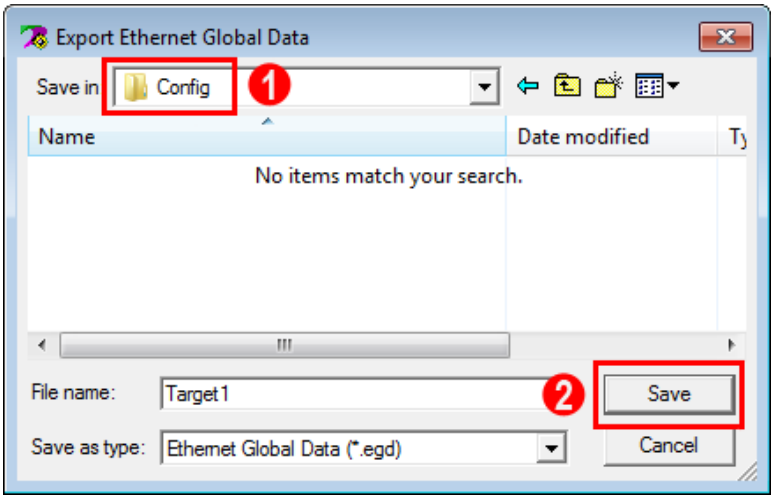
The method used to export the EGD configuration data from a project is:

1. Select the "`Project`" tab within the "`Navigator`" window
2. Right Click on the "`Ethernet Global Data`" item
3. Select the "`Export to File..`" item.



Selecting the "Export to File .." item will open up a dialog box that is used to select the location to where the exported files will be saved. In keeping in line with the recommended directory structure for Mexml projects, use this dialog box to:

1. Select the "Config" directory underneath the "ExportedFiles" directory of the Mexml project.
2. Click on/select the "Save" button.



When "Save" is selected, Machine Edition will export the EGD configuration file to the selected directory.

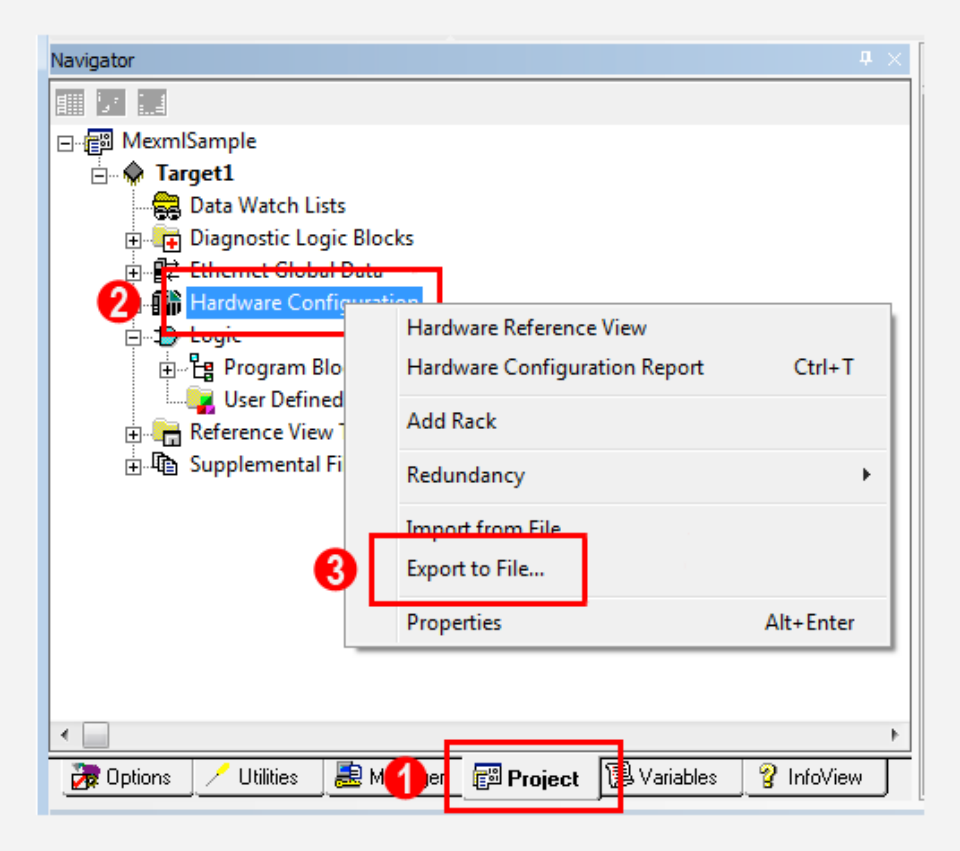
Hardware Configuration

The information contained in a project's Hardware Configuration can optionally be used by `Mexml` in two different ways:

1. Identify the full span of potential variables defined for each I/O card, rather than just those referenced with the program.
2. List out the configuration of each I/O card in a manner not provided by Machine Edition. This aids the user in documenting the flow of data into and out of a project.

The method used to export the Hardware Configuration data from a project is:

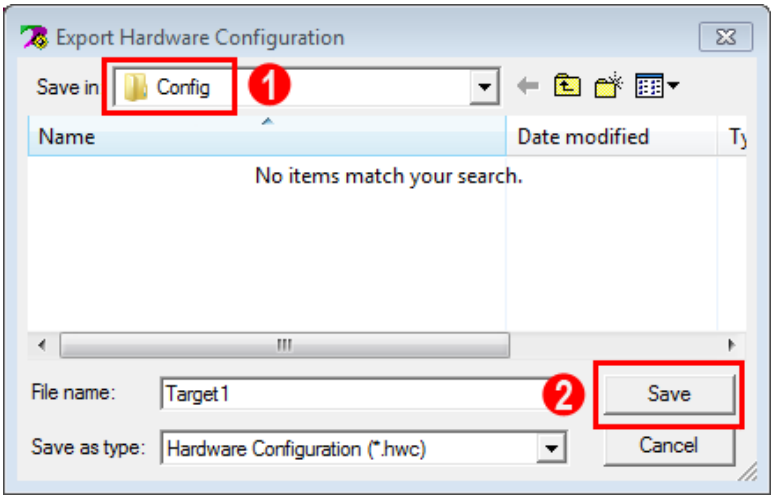
1. Select the "`Project`" tab within the "`Navigator`" window
2. Right Click on the "`Hardware Configuration`" item
3. Select the "`Export to File..`" item.



Selecting the "`Export to File ..`" item will open up a dialog box that is used to select the location to where the exported files will be saved. In keeping in line with the

recommended directory structure for Mexml projects, use this dialog box to:

1. Select the "Config" directory underneath the "ExportedFiles" directory of the Mexml project.
2. Click on/select the "Save" button.



When "Save" is selected, Machine Edition will export the Hardware configuration file to the selected directory.

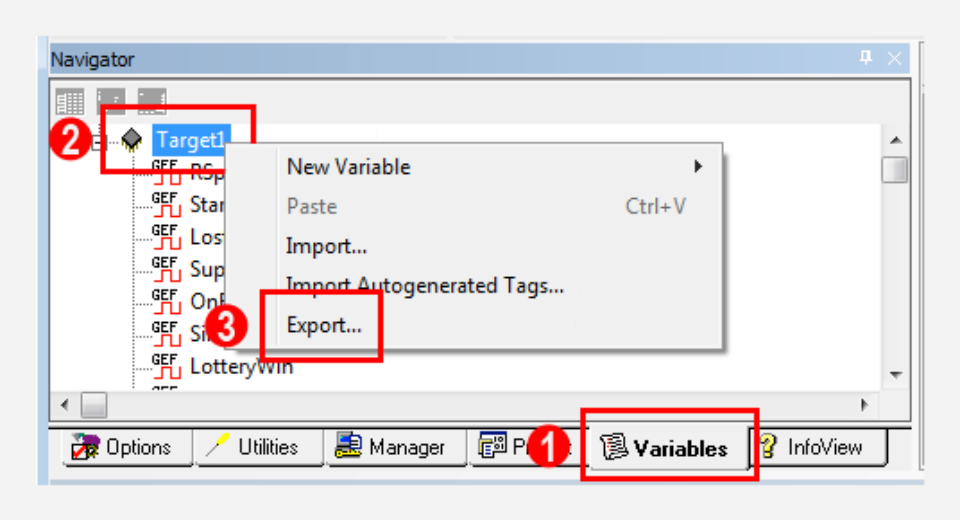
Variables

When analyzing a project's program blocks, `Mexml` will build up a list of all the variables that are referenced within the code. However this list will not include variables that have been defined but not used in any code. The only way to include non-referenced variables in the `Mexml` analysis is to export the complete list of all variables from the project itself. `Mexml` can then (optionally) include this list when processing the Program Blocks.

The method used to export the Hardware configuration data from a project is:

1. Select the "`Variables`" tab within the "`Navigator`" window
2. Right Click on the "`Target`" item of the desired project.
3. Select the "`Export ..`" item.

Note that you can Right Click on any part of the `Variables` list in order to get to the `Export` item. Doing so on the `Target` is suggested in order to reinforce the fact that variables are exported from this particular `Target`.

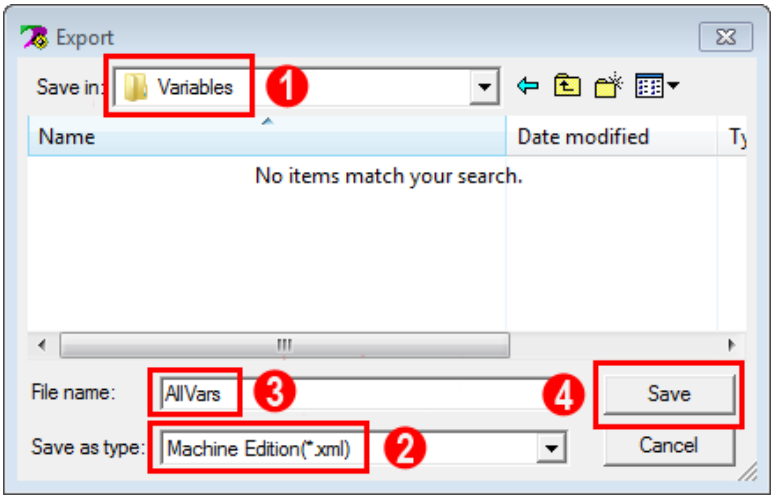


Selecting the "`Export ..`" item will open up a dialog box that is used to select the location to where the exported files will be saved, as well as the format in which the data will exported. In keeping in line with the recommended directory structure for `Mexml` projects, use this dialog box to:

1. Select the "`Variables`" directory underneath the "`ExportedFiles`" directory of the Mexml project.

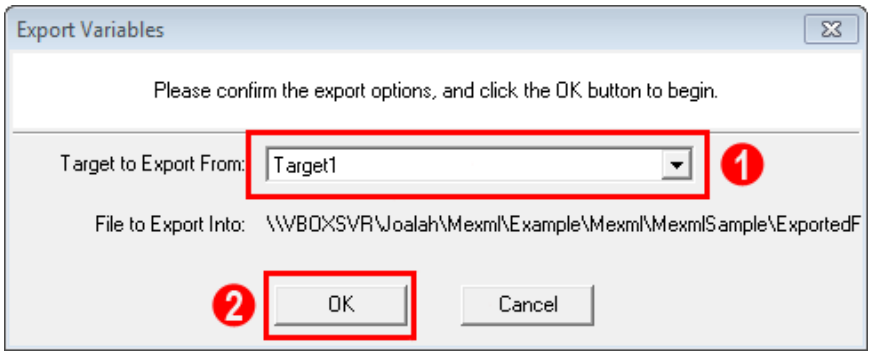
2. Ensure that the "`Save as type`" is selected as "`Machine Edition(*.xml)`"

3. Manually enter the desired name of file. Using the same name for each project that data is exported from will aid in re-using `Mexml` analysis commands. A suggested name for the file where all the variables are export to is "`AllVars`"
4. Click on/select the "`Save`" button.



When "`Save`" is selected, Machine Edition present a dialog box in order to select from which target variables are being exported. Thus:

1. Ensure that the "`Target to Export From`" is the target from which the variables should be exported.
2. Select "`OK`"



When "`OK`" is selected, Machine Edition will export the list of variables to the specified file in the selected directory.

Analysis Command File

Introduction

The most efficient way to use Mexml is through the creation and use of Windows command files that can be reused when analyzing different Machine Edition projects. This section describes such a command file and it can also be used as the basis for creating more tailored command files.

The key to developing a re-useable command file is the strict use of a well defined directory structure to contain the data files exported from the Machine Edition projects. An example of such a structure has previously been described and that will become the basis for this example command file. This file is also provided with the MexmlSample project as the Analysis.cmd file.

The complete Analysis.cmd file is simply:

```
mexml -X %~dp0..\..\%1\ExportedFiles\Code\*.xml -aBD ^
      -E %~dp0..\..\%1\ExportedFiles\Config\*.egd   ^
      -H %~dp0..\..\%1\ExportedFiles\Config\*.hwc  ^
      -P %~dp0..\..\%1\ExportedFiles\Variables\*.xml  ^
      -V %~dp0..\..\%1\LocalFiles\Forces\*.txt ^
      -I %~dp0..\..\%1\LocalFiles\LibraryDefs\*.txt ^
      -I %~dp0..\LibraryDefs\*.txt  ^
      -pv  ^
      -O %~dp0..\..\%1\Results\AnalysisData.txt ^
      -rV ^
      -rM  ^
      -rCS NDACE  ^
      -rF %~dp0..\..\%1\Results\AnalysisVars.txt
```

A variation of the Analysis.cmd file, is also supplied as VarAnalysis.cmd. This file adds the following additional lines to the Analysis.cmd file:

```
      -fA iwR/Ir/Qw ^
      -fRN "^[^\#]" ^
```

These two lines help analyze variables used in a Machine Edition project.

NOTE that the above sample files uses features of Mexml that require a fully licensed version of the program. In order to demonstrate how to use Mexml when only a Trial license is available an additional command file (TrialAnalysis.cmd) has been supplied. This file includes the most functionality that can be achieved with a Mexml Trial license.

The complete TrialAnalysis.cmd file is simply:

```
mexml -X %~dp0..\..\%1\ExportedFiles\Code\*.xml ^
      -pv  ^
      -rV ^
      -rM  ^
      -rCS NDACE  ^
      -rF %~dp0..\..\%1\Results\TrialAnalysisVars.txt ^
      > %~dp0..\..\%1\Results\TrialAnalysisData.txt
```

Deconstruction

Architecture

Both the `Analysis.cmd` and `TrialAnalysis.cmd` files rely on several aspects of both Windows Command files and the recommended directory structure in order to successfully complete its task. These aspects include:

- The `Mexml` executable has been correctly installed, and that it can be executed directly from the command line.
- The command file expects that the recommended directory layout has been adhered to 100%, with no deviations. As such it can specify directory names by relative paths, rather than absolute ones.
- The command file takes a single parameter, the name of the base directory to which the project files have been exported to. Within the command file this parameter is represented by the "`%1`" string.
- Within a Windows Command file, the directory that contains the command file can be referred to as "`%~dp0`". This allows the overall directory to be located anywhere on a computers hard drive, as long as the `mexml.exe` executable file can be located and executed by the command file.
- The "^" character represents the line-continuation symbol within a Command File, and that there are no characters to the right of any of the "^" characters. As a result Windows sees the Analysis command as a single line of text. If the "^" character is needed to be used a part of an `Mexml` option, then the data for that option will need to be wrapped within quotes in order to protect/hide it from the Windows command file processing.

Arguments

The instance of `Mexml` that is executed by the `Analysis.cmd` file is provided with 13 separate arguments, which cover the most basic operation of the program. Breaking down these arguments item by item we have:

-X Selects the `Program Block` files that were exported to the `Code` sub-directory. By using a parameter of "`*.xml`", all files in that directory will be selected.

-E Selects the `EGD Configuration` file that was exported to the `Config` sub-directory. Even though there can only be one `EGD Configuration` file, by using a parameter of "`*.egd`", the `Analysis Command` file does not have to know the name of the target from which the file was exported from. Note that if by accident two such files end up in the same directory, then only the first one will be used. [Req. lic.]

-H Selects the `Hardware Configuration` file that was exported to the `Config` sub-directory. Even though there can only be one `Hardware Configuration` file, by using a parameter of "`*.hwc`", the Analysis Command file does not have to know the name of the target from which the file was exported from. Note that if by accident two such files end up in the same directory, then only the first one will be used. [Req. lic.]

-P Selects the `Variable Preload` files that were exported to the `Variables` sub-directory. By using a parameter of "`*.xml`", the `Analysis Command` file does not have to know the names of these preload files. Note that all such files in this directory will be processed. [Req. lic.]

-V Selects the `Variable Force` files that were manually created in the `Forces` sub-directory. By using a parameter of "`*.txt`", the `Analysis Command` file does not have to know the names of these files. Note that all such files in this directory will be processed. [Req. lic.]

-I Selects the `Function Block Library Definition` files that were manually created either in the overall `LibraryDefs` sub-directory, or the projects `LibraryDefs` sub-directory. By using a parameter of "`*.txt`", the `Analysis Command` file does not have to know the names of these files. Note that all such files in this directory will be processed. [Req. lic.]

-O Defines the file that the `Program Structure` report will be written to. This is the Analysis`Data.txt` file in the `Results` sub-directory. [Req. lic.]

-rF Defines the file that the `Variables` report will be written to. This is the Analysis`Vars.txt` file in the `Results` sub-directory.

-pV Causes the `Mexml` version information to be output to the `Program Structure` report file.

-rV Causes the `Variables` report to be generated.

-rM Includes statistics about the `Variables` report in the `Program Structure` report.

-rCS Selects the columns that will be generated in the `Variables` report. The "`NDACE`" string indicates that the `Name [N]`, `Data Type [D]`, `Address [A]`,

`Access Criteria [C]` and `Description [E]` columns should all be generated.

-fA Filters the items included in the `Variables` report by `Access Criteria`, so that only items matching the specified criteria will be included. [Req. lic.]

In the example given, the specified criteria of `iwR/Ir/Qw` defines three criteria clauses that will be "`Or'ed`" together – thus an item needs to only meet one of the specified criteria in order to be included in the `Variables` report. These criteria clauses are:

- `iwR` Variables that are not a part of the Input Scan, and have been Read without being Written to. These are potentially uninitialized variables.
- `Ir` Variables that are a part of the Input Scan, but have not been Read. These are Input I/O that have potentially not been used.
- `Qw` Variables that are a part of the Output Scan, but have not been Written to. These are Output I/O that have potentially not been used.

-fRN Filters the items included in the `Variables` report to those whose name matches a Regular Expression. [Req. lic.]

In the example given, this expression is "`^[^\#]`". The expression is wrapped in quotes in order to protect the "^" characters from being processed by the Windows command file processing. The expression itself will only match variable names that do no start with a "#" character. Thus this option will eliminate all items like `#ALW_ON` from the `Variables` report.

-aBD When `Mexml` has to automatically generate a variable in order to satisfy a requirement, this option indicates that description of that variable should be left blank (as opposed to generating a description that describes where the variable was created). [Req. lic.]

-sCO Generates the Program Block call order in the `Program Structure` report. [Req. lic.]

Notes:

1. The order off these arguments within the `Analysis.cmd` file is irrelevant.
2. For the arguments that reference files, if those files do not exist then `Mexml` will simply generate a warning message and continue on with its processing.
3. Many of these arguments can only be used with a fully licensed copy of `Mexml`. Those that are not available with a Trial licensed copy of `Mexml` are marked as "[Req. lic.]".

The above selection of arguments should provide a useful starting point for analyzing many different Machine Edition projects.

Example Invocation

Assuming that the command file is located at `c:\Mexml\_mexml\Commands\Analysis.cmd`, and that a Windows Command Prompt has been opened in the c:\`Mexml` directory, then this file could be invoked to process the `MexmlSample` project by entering the following command line:

```
c:\Mexml>_mexml\Commands\Analysis.cmd MexmlSample
```

Note that entering this command line is not as cumbersome as it first seems, as using Windows Auto-Completion through the use of the `[Tab]` key means that the complete command line can be created in only 11 key-strokes.

```
c:\Mexml>_[Tab]\C[Tab]\A[Tab] M[Tab]
```

On completion of the analysis, the following files will have been created:

```
c:\Mexml\MexmlSample\Results\AnalysisData.txt
c:\Mexml\MexmlSample\Results\AnalysisVars.txt
```

The power of the Analysis Command file comes into play when analyzing multiple projects. Assuming a second Machine Edition project called "`Project2`" has been exported to the same set of sub-directories, then this project can analyzed by executing the command line:

```
c:\Mexml>_mexml\Commands\Analysis.cmd Project2
```

Thus `Mexml` will be invoked with the same set of arguments as was used to analyze the MexmlSample project, which will result in the following files being generated:

```
c:\Mexml\Project2\Results\AnalysisData.txt
c:\Mexml\Project2\Results\AnalysisVars.txt
```

MexmlSample Project

Overview

Mexml is supplied with a sample Machine Edition project that deliberately includes several errors and oversights that can easily be made when using Machine Edition. These “mistakes” have been included to highlight many of the analysis techniques that Mexml implements, and to show that even in a small project it is hard to manually verify the quality of a PLC program.

The sample itself is a zipped up file (MexmlSampleProject.zip) that contains the following items:

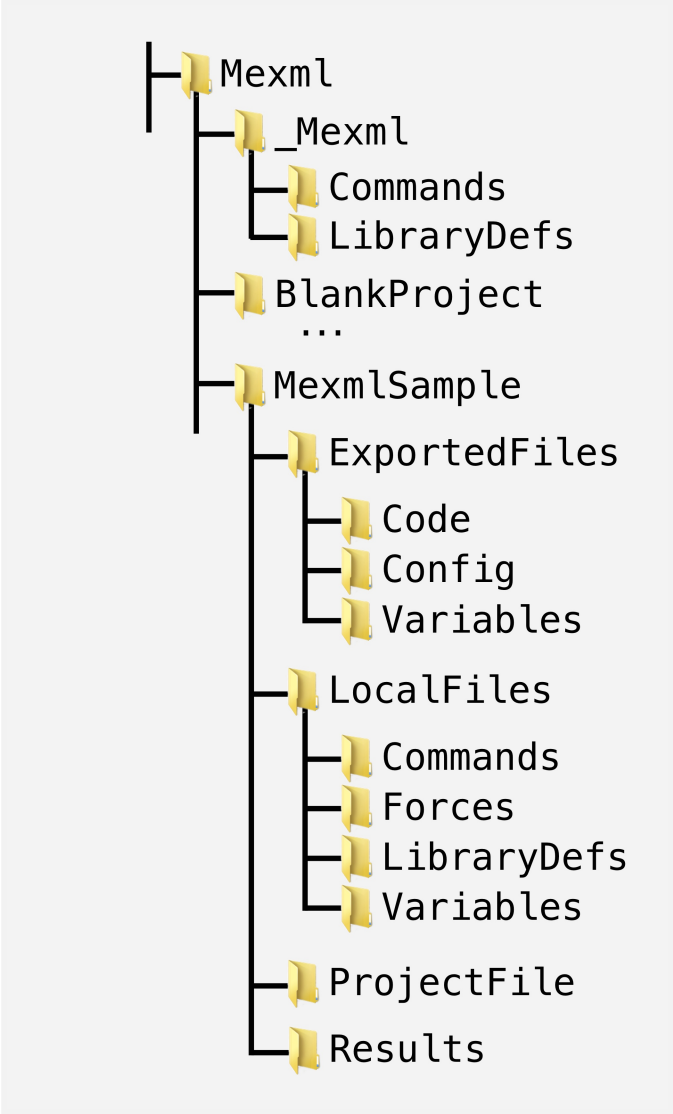
- A complete instance of the recommended directory structure.
- Examples of the Analysis.cmd, VarAnalysis.cmd and TrialAnalysis.cmd files for analyzing general Machine Edition projects.
- Examples of command files for analyzing the MexmlSample in more detail
- The MexmlSample Machine Edition project.
- The complete set of data that has been exported from the MexmlSample, and saved to the expected directories.
- Manually created files used to assist in analyzing the MexmlSample project.
- Sample results files as created from using the Analysis.cmd and TrialAnalysis.cmd files. (Note that these files are named in such a way as to not be overwritten when the actual result files are generated.)
- A set of blank project directories to aid in working with Mexml projects.

All that is required to set up this system to show the capabilities of Mexml is to unzip the MexmlSampleProject.zip file to a local directory. (And of course have installed Mexml)

The MexmlSample project itself is a simple Rx3i based PLC project that consists of a single Rack with CPU and I/O cards, and also defines two EGD exchanges. The program logic is contained in several Program Blocks and includes Ladder Logic, Structured Text and Function Block definitions. This range of functionality and techniques covers what is needed in a large number of real world projects, yet the small size also enables most features of Mexml to be demonstrated with a project that is easy to comprehend.

Contents

Unzipping the `MexmlSampleProject.zip` file results in the directory structure of:



There are 3 main directories under the `Mexml` root directory:

`_Mexml`	Contains commands and data files that can be shared across multiple Machine Edition projects.
`BlankProject`	A series of directories and empty files that form a template for analyzing a single Machine edition project.
`MexmlSample`	The complete set of files exported from the MexmlSample Machine Edition project, laid out in the "`BlankProject`" style. It also contains the original Machine Edition project as well as examples of

the results created by the analysis command files.

The contents of each the key directories are:

_Mexml\		
Commands\	Analysis.cmd	Main analysis command
	VarAnalysis.cmd	Variable analysis command
	TrialAnalysis.cmd	“Trial license” command
LibraryDefs\	GlobalLibraryDefs.txt	Available to all projects
MexmlSample\		
ProjectFile\	MexmlSample.zip	Machine Edition project
MexmlSample\ExportedFiles\		
Code\	_Main.xml	Program Blocks from project
	AABlock.xml	
	BBBlock.xml	
	BFunction.xml	
	CCBlock.xml	
	DDBlock.xml	
	EEBlock.xml	
	Setup.xml	
Config\	Target1.egd	Project's EGD definition
	Target1.hwc	Project's Hardware definition
Variables\	AllVars.xml	All variables from project
MexmlSample\LocalFiles\		
Forces\	LocalForces.txt	Forces for just this project
LibraryDefs\	LocalLibraryDefs.txt	Library Defs for just this project
Results\	_AnalysisData.txt	Sample results from analysis files
	_AnalysisVars.txt	
	_VarAnalysisData.txt	
	_VarAnalysisVars.txt	
	_TrialAnalysisData.txt	
	_TrialAnalysisVars.txt	

Program Errors

The MexmlSample program deliberately includes several errors in order to highlight the analysis strengths of Mexml. These errors are both in the structure of the program as well as in how variables are treated. The following sections document each error, and how and where it is reported.

Program Structure report

The following issues are reported in the `Program Structure` report. These issues are explicitly reported in this report, and (with the exception of the `Block not called` issue) are marked in the report as:

`** ERR`	Indicates an issue that `Mexml` believes is an error in the PLC program. Such issues are likely to cause incorrect or unexpected operation of the PLC program.
`++ Warn`	Indicates an issue that needs to be reviewed for correct operation. Such issues could arise from design decisions and as such `Mexml` can't determine whether they are in fact errors or deliberate choices.

Note that in the example Program Structure reports, all of the reported issues appear at the top of the file. This grouping only occurs due to the set of arguments provided to `Mexml`. If arguments requesting the listing of program logic were supplied, then the errors and warning would appear scattered throughout the report, as close as possible to the location were the issue was first detected.

The `Program Structure` reports generated by each of the sample analysis commands, and in which the issues can be seen are:

`_AnalysisData.txt`	Created by the `Analysis.cmd` file
`_VarAnalysisData.txt`	Created by the `VarAnalysis.cmd` file
`_TrialAnalysisData.txt`	Created by the `TrialAnalysis.cmd` file

`Block not called`	The `EEBlock` is defined in the project's logic, but is not called anywhere in the program. This is seen in the "`POU Call order`" section of the `Program Structure` reports.
`Bad Transfer Size`	In the `AABlock`, a group of contiguous registers is copied to an array of registers. However the number of registers copied is larger than the size of the array, and the copy will overflow past the end of the array. This is reported as an error ("`** ERR`") .

Shared Variable The BFunction function block uses a variable (TEMP2) that is not a "Member Variable" of the function block. This means that all instances of this function block will share the same instance (and value) of the one and only TEMP2 variable. As the intent of the programmer can't be discerned from the code, Mexml can't determine if this is actually an error or simply a design choice. Thus this condition is reported as a warning ("++ Warn") .

Overlapping Vars The DINT sized variables "Number1" and "Number2" each require 2 consecutive register locations. So that if "Number1" was defined at address %R0021, it utilizes %R0021 and %R0022, and that the next free address to use is %R0023. However "Number2" is configured to start at %R0022, and thus partially overlaps the memory of "Number1", which can cause data errors in the PLC program. This is reported as an error ("** ERR") .

Variables report

The following issues are reported in the Variables report. In general these issues are discerned from the way that a particular variable has been accessed. For example having a "Read" access but no "Write" access implies an uninitialized variable.

The Variables reports generated by each of the sample analysis commands, and in which the issues can be seen are:

_AnalysisVars.txt	Created by the Analysis.cmd file
_VarAnalysisVars.txt	Created by the VarAnalysis.cmd file
_TrialAnalysisVars.txt	Created by the TrialAnalysis.cmd file

Uninitialized Var In the DDBlock, the local BOOL variable CLR_Fancy was used in a contact, but there is no instruction setting its value. This is seen by the variable DDBlock.CLR_Fancy only having a Read "R" access. The root cause of this issue is that previously in the same program block a typo was made when creating the instruction that should have set the value of CLR_Fancy. This resulted in an additional variable unexpectedly being created with the name CLR_Fancey.

Uninitialized Vars In the EEBlock, the global BOOL variable BoolDataIn2 and BoolDataIn4 were used in contacts, but there are no instruction setting their value. This is seen by the variables BoolDataIn2 and BoolDataIn4 only having a Read "R"

access. The root cause of this issue is that when the contacts were being defined, a typo was made, and that instead of entering `BoolDataIn[2]` and `BoolData[4]`, the brackets were left off. This resulted in an additional variables unexpectedly being created with the names `BoolData2` and `BoolData2`.

Unused Input I/O The `EGD Input` variables `BoolDataIn[2]` and `BoolDataIn[4]` were defined, but the variables are not being used anywhere in the program. This is seen by these variables only having an `Input Scan "I"` access. While not using an Input I/O value may be a design choice, in this case it is an error, and the root cause can be seen in the `EEBlock`. This error is a follow-on from the typos that generated the `BoolDataIn2` and `BoolDataIn4` `Uninitialized Vars` error, where `BoolDataIn[2]` and `BoolData[4]` should have been used for the contacts.

Unused Output I/O The `EGD Output` variables BoolDataOut[3] and BoolDataOut[6] were defined, but the variables are not being used anywhere in the program. This is seen by these variables only having an `Output Scan "Q"` access. While not using an Output I/O value may be a design choice, in this case it is an error, for which the root cause can be seen in the `EEBlock`. The errors are due typos being made when defining the instructions that should have written the values of these variables. The typos left of the brackets from the variable names and unexpectedly cause variables `BoolDataOut3` and `BoolDataOut6` to be created.

Uninitialized Vars The variables `Data_5`, `Data_6`, `Number1` and `Number2` are used in instructions that read their values, but those values are not being written anywhere in the program. This is seen by these variables only having a `Read "R"` access. In these cases, typos are not to blame, and that instead the code to write the values is simply missing. Alternatively, the values were meant to be written by an external source directly into the program — for example by an HMI. In that case these variables should be entered into the requisite "`Forces`" definition file.

Unused Input I/O The `EGD Input` variable `EGDConsExchgStatus`, the `Ethernet` card status variables `EthernetStatus_1` and `EthernetStatus_2` are defined but the variables are not being used anywhere in the program. This is seen by these

variables only having an Input Scan "I" access. In these cases, typos are not to blame, and that the code to use these variables is simply missing from the program.

Unused Output I/O The EGD Output variable EGDProdExchgStatus is defined but does not appear to be written anywhere in the program. This is seen by the variable only having an Output Scan "Q" access. In this case there is no error, as it is the PLC system itself that writes the value of this variable. In order to remove this artifact from being reported, the variable should be entered into the requisite "Forces" definition file.

Unused Output I/O The physical output variable LostAtSea, is defined but does not appear to be written anywhere in the program. This is seen by the variable only having and Output Scan "Q" access. In this case, typos are not to blame, and that the code to write this variables is simply missing from the program.

Unused Output I/O The physical output variable Q00008, is defined but does not appear to be written anywhere in the program. This is seen by the variable only having an Output Scan "Q" access. In this case, that the variables name matches its address (%Q00008) alludes to there being no user defined variable at this address, and in fact this location is likely to be spare I/O.

Unused Input I/O The EGD Input variable IntDataIn[1] was defined, but the variable is not being used anywhere in the program. This is seen in by the variable only having an Input Scan "I" access. In this case this is an error for which the root cause can be seen in the EEBlock. The IntDataIn[1] variable is a part of the IntDataIn array (which is 2 elements long), however in the EEBlock where the contents of the IntDataIn array are copied to another location, a transfer size of only 1 is specified. This results in IntDataIn[1] not being read.

Unused Output I/O The EGD Input variable IntDataOut[1] was defined, but the variable is not being used anywhere in the program. This is seen by the variable only having an Output Scan "Q" access. In this case this is an error for which the root cause can be seen in the EEBlock. The IntDataOut[1] variable is a part of the IntDataOut array (which is 2 elements long), however in the EEBlock where the contents of the InDataOut array are copied from another location, a transfer

size of only 1 is specified. This results in `IntDataOut[1]` not being written.

Tips And Tricks

There are many ways that Mexml can be used to analyze a Machine Edition project in order to highlight aspects of its design and usage. The followings sections present some of these ways in which Mexml can be used.

Program Blocks

In order to analyze the program blocks, first include them using the "Program files (.xml) exported from the ME project" argument (–X <ProgramFiles(s)>).

```
–X path\to\program\files\*.xml
```

The results from analyzing the program blocks will appear in the Program Structure report. By default this report will be sent to the console output. However it can be directed to a specific file by using the "Output file for all Source Messages and Structure Reports" argument (–O <OutputFile>), which can be any valid Windows file designation.

```
–O path\to\ProgramReportFile.txt
```

Program bock call order

The "Program Block Call Order" argument (–sC0) generates the list of program blocks and displays them in the order that they are processed by the PLC. Using this argument on a large project allows you to get a top down view on how the project is structured. In addition this argument also identifies any program blocks that are in the project, but not called — facilitating the location of dead or overlooked code.

The complete set of arguments can be summarized as:

```
Mexml –X path\to\program\files\*.xml ^
      –O path\to\ProgramReportFile.txt ^
      –sC0
```

Where the trailing "^" is the Windows Command file line continuation character.

File processing order

The "File Processing Order" argument (–xF0) generates the list of program blocks in their dependency order, from least dependent to most dependent. This information is needed when manually transferring a group of program blocks from one project to another. As although you can export the program blocks in any order, you have to

manually import them from the least to most dependent program block file.

In order to speed up generation of just this data, and not process the entire ME project, combine the –xFO argument with the "Skip analysis of all Program Blocks" argument (–xKA).

The complete set of arguments can be summarized as:

```
Mexml –X path\to\program\files\*.xml ^
      –O path\to\ProgramReportFile.txt ^
      –xFO ^
      –xKA
```

Where the trailing "^" is the Windows Command file line continuation character.

Documentation

Being able to automatically document aspects of a Machine Edition project ensures that all such documentation is up to date and not subject to the vagaries of manual notes or inspections.

EGD Exchanges

In order to document the EGD exchanges, first include them with the "EGD configuration file (.egd) exported from the ME project" argument (–E <EGDFile>).

```
–E path\to\egd\configurations\*.egd
```

The details of the EGD exchanges can be listed with the "EGD Configuration messages" argument (–eM), which will list out the details of each exchange, such as:

- Produced or Consumed
- IP addresses associated with the exchange
- Variables or addresses referenced by the exchange

It is recommended that this is combined with the "Give auto generated variables a blank description" argument (–aBD) in order to produce an output that is "cleaner" and hence more easily understood by a person.

The produced EGD documentation will appear in the Program Structure report.

Note that this data can be produced independently from analyzing the project's Program Blocks.

The complete set of arguments can be summarized as:

```
Mexml –E path\to\egd\configurations\*.egd ^
      –O path\to\ProgramReportFile.txt ^
      –eM ^
      –aBD
```

Where the trailing "^" is the Windows Command file line continuation character.

Hardware I/O

In order to document the Hardware I/O, first include it with the "`Hardware Configuration file (.hwc) exported from the ME project`" argument (`–H <HardwareFile>`).

```
–H path\to\hardware\configurations\*.hwc
```

The details of the I/O can be listed with the "`Hardware Configuration messages`" argument (`–hM`), which will list out the details of each I/O card in the system, such as:

- Rack, slot and model number of the card
- Variables or addresses referenced by the card
- Details of any daughter cards installed on the main card
- Details of any remote I/O (EG Genius, Profibus etc) connected to the card

It is recommended that this is combined with the "`Give auto generated variables a blank description`" argument (`–aBD`) in order to produce an output that is "cleaner" and hence more easily understood by a person.

The produced Hardware documentation will appear in the `Program Structure` report.

Note that this data can be produced independently from analyzing the project's Program Blocks.

The complete set of arguments can be summarized as:

```
Mexml –H path\to\hardware\configurations\*.hwc ^
      –O path\to\ProgramReportFile.txt ^
      –hM ^
      –aBD
```

Where the trailing "^" is the Windows Command file line continuation character.

Bit Mappings

Machine Edition allows for register type variables (EG INT, DINT, FLOAT etc) to be located in bit based address spaces (EG %M, %I, %Q etc), and for separate bit based variables (BOOL) to be mapped into the same addresses.

There are two ways from which the information of these mappings can be ascertained:

1. By processing all of the Program Files in the project through the use of the "`Program files (.xml) exported from the ME project`" argument (`–X <ProgramFiles(s)>`).

```
–X path\to\program\files\*.xml
```

2. By processing the all of the variables exported from the Machine Edition project through the use of the "`Preload Variables (.xml) exported from the ME project`" argument (`–P <PreloadFile(s)>`).

```
–P path\to\preload\files\*.xml
```

The former will analyze all mappings that are actually used in the program, while the latter will produce the results faster — although it may include mappings that are not in use.

The actual details of the mappings can be listed with the "`Mapped Address variables`" argument (`–sMA`), which will list out details of each mapping, such as:

- The base register variable, including its starting address and data type
- The list of all other variables that map into the same address as the base register

It is recommended that this is combined with the "`Give auto generated variables a blank description`" argument (`–aBD`) in order to produce an output that is "cleaner" and hence more easily understood by a person.The produced Bit Mapping documentation will appear in the `Program Structure` report.

Note that the -sMA argument will list out all variables which have the same or overlapping addresses, which includes items that are synonyms for individual addresses.

The complete set of arguments can be summarized as:

```
Mexml –X path\to\program\files\*.xml ^
      –O path\to\ProgramReportFile.txt ^
      –sMA ^
      –aBD
```

Or

```
Mexml –P path\to\preload\files\*.xml ^
      –O path\to\ProgramReportFile.txt ^
      –sMA ^
      –aBD
```

Where the trailing "^" is the Windows Command file line continuation character.

Uninitialized variables

Machine Edition does not identify uninitialized variables — ones that have been read, but not written to. `Mexml` can detect such items through a combination of processing the Program Blocks and then filtering the resulting `Variables report`.

The program blocks are included using the "`Program files (.xml) exported from the ME project`" argument (`–X <ProgramFiles(s)>`).

```
–X path\to\program\files\*.xml
```

The `Variables report` will be generated when the "`Required to report on Variables`" argument (`–rV`) is specified.

By default the `Variables report` will be sent to the console output, however it can be directed to a specific file by using the "`Report Filename to save the report data to`" argument (`–rF <ReportFile>`), which can be any valid Windows file designation. If the file was specified using a relative path, then the `–rF` argument can be combined with the "`Base directory for all reports`" argument (`–rD <ReportDir>`) which will prepend the `<ReportFile>` with the `<ReportDir>` path.

```
–rV  –rD c:\path\to\all\reports  –rF path\to\VarReportFile.txt
```

In order to better highlight the uninitialized items in the Variables report, that report should be filtered to only include the desired results. This can be done only including variables that are not a part of an input (I/O or EGD) scan, that have been read, but have not been written to. The filtering is done using the "`Filter report by variables that meet the access criteria`" argument (`–fA <AccessCriteria>`), specified as:

```
-fA iRw
```

Where:

- A lower case criteria implies "Not Set", while upper case implies "Set"
- "iRw" represents "Input Scan (Not set)" and "Read (Set)" and "Write (Not set)".

Where a lowercase criteria implies "Not Set", while upper case implies "Set", and "i" is "Input Scan (not set)", "R" is "Read (set)" and "W" is "Write (set)".

This filtering will not exclude Machine Edition system variables such as #ALW_ON and #ALW_OFF. The best way to eliminate these from the Variables report is to only match items whose names do not start with a "#" character, by using the "Filter report by name Regex" argument (-fRN <NameRegex>). A Regular Expression (Regex) is a specialized grammar used to describe strings of characters. It is outside the scope of this document to describe the grammar itself in detail. In addition there are many online resources devoted to explaining Regexes and how to use them.

The Regex that will eliminate the system variables from the Variables report is:

```
-fRN "^[^\#]"
```

The Regex itself is enclosed in quotes in order to stop the Windows command line from interpreting some of the characters as relevant to the command line. This Regex can be interpreted as:

1. Match from the start of the name (the first "^")
2. Any characters in the set defined within "[" and "]"
3. The set excludes the following character (the second "^")
4. The literal "#" character defined by "\#"

Thus it excludes all variables whose name starts with "#".

Finally, the above reporting cannot take into account any external sources that write directly into the PLCs memory — such as an HMI system. These items can only be excluded from the Variables report by using the "Force State of Variables as defined in user created text file(s)" argument (-V <VarForceFile(s)>) to mark items that are known to be initialized. This would be done by first creating a VarForceFile with entries like:

```
// Forces for program

// HMI interface
// The folowing varables are written by the HMI system
WI Variable1
WI Variable2
WI Variable3
WI Variable4
...
```

Which would force the `Write` and `InputScan` Access Criteria of `Variable1` to be set. This file is included for processing by:

```
-V path\to\VarForceFile.txt
```

All of these arguments can be summarized as:

```
Mexml -X path\to\program\files\*.xml ^
      -V path\to\VarForceFile.txt ^
      -rV
      -rD c:\path\to\all\reports
      -rF path\to\VarReportFile.txt ^
      -fA iRw ^
     -fRN "^[^\#]"
```

Where the trailing "^" is the Windows Command file line continuation character.

Unused I/O

Machine Edition does not identify unused I/O – inputs that have not been read, or output variables that have not written to. `Mexml` can detect such items through a combination of processing the Program Blocks and then filtering the resulting `Variables report`.

The program blocks are included using the "`Program files (.xml) exported from the ME project`" argument (`–X <ProgramFiles(s)>`).

```
-X path\to\programfiles\*.xml
```

The `Variables report` will be generated when the "`Required to report on Variables`" argument (`–rV`) is specified.

By default the `Variables report` will be sent to the console output, however it can be directed to a specific file by using the "`Report Filename to save the report data to`" argument (`-rF <ReportFile>`), which can be any valid Windows file designation. If the file was specified using a relative path, then the `-rF` argument can be combined with the "`Base directory for all reports`" argument (`-rD <ReportDir>`) which will prepend the `<ReportFile>` with the `<ReportDir>` path.

```
-rV  -rD c:\path\to\all\reports  -rF path\to\VarReportFile.txt
```

In order to better highlight the unused items in the Variables report, that report should be filtered to only include the desired results. This can be done only including variables that are a part of an input (I/O or EGD) scan that have not been read, or are a part of an output scan (I/O or EGD) that have not been written to. The filtering is done using the "`Filter report by variables that meet the access criteria`" argument (`-fA <AccessCriteria>`), specified as:

```
-fA Ir/Qw
```

Where:

- "`/`" is interpreted as an "`OR`" conjunction
- A lower case criteria implies "`Not Set`", while upper case implies "`Set`"
- "`Ir`" represents "`Input Scan (Set)`" and "`Read (Not set)`"
- "`Qw`" represents "`Output Scan (Set)`" and "`Write (Not set)`"

All of these arguments can be summarized as:

```
Mexml -X path\to\programfiles\*.xml ^
      -rV
      -rD c:\path\to\all\reports
      -rF path\to\VarReportFile.txt ^
      -fA Ir/Qw ^
```

Where the trailing "^" is the Windows Command file line continuation character.

Unused variables

Machine Edition does not identify unused variables — items that have not been read or written to. `Mexml` can detect such items through a combination of processing the Program Blocks, Preloaded variables and then filtering the resulting `Variables report`.

The program blocks are included using the "`Program files (.xml) exported from`

the ME project" argument (–X <ProgramFiles(s)>).

```
–X path\to\program\files\*.xml
```

Because the processing of the Program Blocks will only consider variables that have been referenced one way or another, Mexml also has to reference the complete set of variables in order to identify the unused variables. These variables are included using the "Preload Variables (.xml) exported from the ME project" argument (–P <PreloadFile(s)>).

```
–P path\to\preload\files\*.xml
```

The Variables report will be generated when the "Required to report on Variables" argument (–rV) is specified.

By default the Variables report will be sent to the console output, however it can be directed to a specific file by using the "Report Filename to save the report data to" argument (–rF <ReportFile>), which can be any valid Windows file designation. If the file was specified using a relative path, then the –rF argument can be combined with the "Base directory for all reports" argument (–rD <ReportDir>) which will pre-pend the <ReportFile> with the <ReportDir> path.

```
–rV  –rD c:\path\to\all\reports  –rF path\to\VarReportFile.txt
```

In order to better highlight the unused items in the Variables report, that report should be filtered to only include the desired results. This can be done only including variables that are neither read nor written, and are not a part of an input or output scan. The filtering is done using the "Filter report by variables that meet the access criteria" argument (–fA <AccessCriteria>), specified as:

```
–fA rwiq
```

Where:

- A lower case criteria implies "Not Set", while upper case implies "Set"
- "rwiq" represents "Read (Not Set)" and "Write (Not set)" and "InputScan (Not set)" and "Output Scan (Not set)".

All of these arguments can be summarized as:

```
Mexml -X path\to\program\files\*.xml ^
      -P path\to\preload\files\*.xml ^
      -rV ^
      -rD c:\path\to\all\reports ^
      -rF path\to\VarReportFile.txt ^
      -fA rwiq
```

Where the trailing "^" is the Windows Command file line continuation character.

Program Block Interface

In many situations a group of Program Blocks is developed as a self contained set of instructions that can be considered a single unit of processing. The splitting of the overall task into multiple blocks aids in design, implementation, code re-use and run time fault finding or debugging. `Mexml` further aids in this process by being able to scan that set of Program Blocks and list out all variables that represent data that flows into or out of them — thus documenting the effective "Data" Interface. This can be done through a combination of processing the Program Blocks and then filtering the resulting `Variables report`.

The program blocks are included using the "`Program files (.xml) exported from the ME project`" argument (`-X <ProgramFiles(s)>`).

```
-X path\to\interface\files\*.xml
```

Note that the path should lead only to those Program Blocks that are a part of the Interface being documented.

The `Variables report` will be generated when the "`Required to report on Variables`" argument (`-rV`) is specified.

By default the `Variables report` will be sent to the console output, however it can be directed to a specific file by using the "`Report Filename to save the report data to`" argument (`-rF <ReportFile>`), which can be any valid Windows file designation. If the file was specified using a relative path, then the `-rF` argument can be combined with the "`Base directory for all reports`" argument (`-rD <ReportDir>`) which will pre-pend the `<ReportFile>` with the `<ReportDir>` path.

```
-rV  -rD c:\path\to\all\reports  -rF path\to\VarReportFile.txt
```

In order to highlight the Interface items in the Variables report, that report should be filtered to only include the desired results. This can be done only including variables that are:

- Part of an Input Scan
- Part of an Output Scan
- Read but not Written to, and not a part of an Input or Output Scan
- Written to but Not Read, and not a part of an Input or Output Scan

Note that the above will not include variables that are Written to and Read by the set of Program Blocks, but are referenced by other Program Blocks.

The filtering is done using the "`Filter report by variables that meet the access criteria`" argument (`-fA <AccessCriteria>`), specified as:

```
-fA I/Q/Rwiq/Wriq
```

Where:

- "`/`" is interpreted as an "`OR`" conjunction
- A lower case criteria implies "`Not Set`", while upper case implies "`Set`"
- "`I`" represents "`Input Scan (Set)`"
- "`Q`" represents "`Output Scan (Set)`"
- "`Rwiq`" represents "`Read (Set)`" and "`Write (Not set)`" and "`Input Scan (Not set)`" and "`Output Scan (Not set)`"
- "`Wriq`" represents "`Write (Set)`" and "`Read (Not set)`" and "`Input Scan (Not set)`" and "`Output Scan (Not set)`"

All of these arguments can be summarized as:

```
Mexml -X path\to\interface\files\*.xml ^
      -rV
      -rD c:\path\to\all\reports
      -rF path\to\VarReportFile.txt ^
      -fA I/Q/Rwiq/Wriq ^
```

Where the trailing "^" is the Windows Command file line continuation character.

Simultaneous Multiple Filtering Criteria

The previous examples have all focused on performing a single analysis of a Machine Edition project. In order to perform multiple, different analysises of the same project, then it appears that you would have to run each one separately. However `Mexml` supports simultaneous and independent filtering a set of Program Blocks, with the results of each analysis being written to a different report file. Thus you could process a Machine edition project and in one action generate separate reports for

- Uninitialized Variables
- Unused I/O
- Unused Variables

The key to this functionality is the use of the "`List of report parameter definitions in plain text file`" argument (`-rP <ReportParamFile>`). This argument reads a text file in which each line defines separate a set of reporting arguments. These arguments are applied to a common set of source files that are defined as a part of the main execution of `Mexml`.

In order to generate the reports previously mentioned, the `Mexml` arguments common to all the reports would be defined as:

```
-X path\to\program\files\*.xml
-P path\to\preload\files\*.xml
-V path\to\VarForceFile.txt
-rV
-rD c:\path\to\all\reports
```

The `ReportParamFile` (`RepParam.txt`) would be defined as:

```
-fA iRw  -fRN "^[^\#]"  -rF UninitializedVars.txt
-fA Ir/Qw  -rF UnusedIO.txt
-fA rwiq  -rF UnusedVars.txt
```

Where each line corresponds to a separate report. Note that as this is a pure text file, the Windows Command file continuation character is irrelevant. This file would be referred to by using the `-rP` argument as:

```
-rP path\to\report\definition\files\RepParam.txt
```

Thus the complete set of `Mexml` arguments would be:

```
Mexml -X path\to\program\files\*.xml  ^
      -P path\to\preload\files\*.xml  ^
      -V path\to\VarForceFile.txt  ^
      -rV  ^
      -rD c:\path\to\all\reports  ^
      -rP path\to\report\definition\files\RepParam.txt
```

Where the trailing "^" is the Windows Command file line continuation character.

Running this command will result in the following files being created:

```
c:\path\to\all\reports\UninitializedVars.txt
c:\path\to\all\reports\UnusedIO.txt
c:\path\to\all\reports\UnusedVars.txt
```

With each file only containing its corresponding data, and Mexml only having processed the Program Blocks the one time.

Compare two systems

Because Mexml produces reports in a pure text format, it is trivial to use a 3rd party program to compare the reports generated by two different versions of the same Machine edition project. In general it is useful to compare the code, the I/O and the variables being used in order to identify the functional differences between the two versions.

The program blocks are included using the "Program files (.xml) exported from the ME project" argument (–X <ProgramFiles(s)>).

```
–X path\to\program\files\*.xml
```

In order to generate a code listing that is geared towards comparing versions, the following arguments are suggested:

```
–xLB
–xLRS
–xSB
```

Where:

- Ladder Logic – Body (Code) messages, (–xLB)
- Ladder Logic – Suppress rung numbers, (–xLRS)
- Structured Text – Body (Code) messages, (–xSB)

These will generate clean Ladder Logic and Structured Text code listings. The suppression of the Ladder Logic rung numbers is suggested in order to reduce any comparison "noise" that would be created if a small change in rung numbers occurred. Such a change could result in a "mis-match" between every line starting at the initial change and propagating until the end of the Program Block, even if the actual logic itself remained unchanged.

The EGD exchange data is included by using the "EGD configuration file (.egd) exported from the ME project" argument (–E <EGDFile>).

```
-E path\to\egd\configurations\*.egd
```

In order to generate an EGD definition that is geared towards comparing versions, the following arguments are suggested:

```
-eM
-eDS
```

Where:

- `EGD Configuration messages`, (`–eM`)
- `Suppress the description in the EGD Configuration`, (`–eDS`)

These will generate a clean EGD definition. The suppression of the descriptions is suggested as differences in descriptions between program versions are purely non-functional. However, it may useful to identify such changes for example, where the changing description indicates changed functionality. In which case the -eDS argument could be left off.

The Hardware I/O is included by using the "`Hardware Configuration file (.hwc) exported from the ME project`" argument (`–H <HardwareFile>`).

```
-H path\to\hardware\configurations\*.hwc
```

In order to generate a Hardware definition that is geared towards comparing versions, the following arguments are suggested:

```
-hM
-hDS
```

Where:

- `Hardware Configuration messages`, (`–hM`)
- `Suppress the description in the Hardware Configuration`, (`–hDS`)

These will generate a clean Hardware definition. The suppression of the descriptions is suggested as differences in descriptions between program versions are purely non-functional. However, it may useful to identify such changes for example, where the changing description indicates changed functionality. In which case the -hDS argument could be left off.

The results from analyzing the program blocks, EGD and Hardware configurations will appear in the `Program Structure` report. By default this report will be sent to the console output. However it should be directed to a specific file by using the "`Output file for all Source Messages and Structure Reports`" argument (`–O`

<OutputFile>), which can be any valid Windows file designation.

```
-O path\to\ProgramReportFile.txt
```

The Variables report will be generated when the "Required to report on Variables" argument (-rV) is specified.

By default the Variables report will be sent to the console output, however it should be directed to a specific file by using the "Report Filename to save the report data to" argument (-rF <ReportFile>), which can be any valid Windows file designation. If the file was specified using a relative path, then the -rF argument can be combined with the "Base directory for all reports" argument (-rD <ReportDir>) which will pre-pend the <ReportFile> with the <ReportDir> path.

```
-rV  -rD c:\path\to\all\reports  -rF path\to\VarReportFile.txt
```

In order to better highlight differences in the Variables report between program versions, only functional information should be included such as:

- Variable Name
- Data Type
- PLC Address
- Access Criteria

This is done through the use of the "Select columns included in the report" argument (-rCS <ColumnSelect>), which dictates what columns appear in the Variables report.

```
--rCS NDAC
```

Where:

- "N" represents the Variable Name column
- "D" represents the Data Type column
- "A" represents the PLC Address column
- "C" represents the Access Criteria column

It is suggested to leave out the Description column as differences in descriptions between program versions are purely non-functional. However, it may useful to identify such changes for example, where the changing description indicates changed functionality. In which case the Description could be included through the addition of the "E" column identifier.

Thus the complete set of `Mexml` arguments would be:

```
Mexml -X path\to\program\files\*.xml  ^
      -E path\to\egd\configurations\*.egd  ^
      -H path\to\hardware\configurations\*.hwc  ^
      -xLB  -xLRS  -xSB  -eM  -eDS  -hM  -hDS
      -O path\to\ProgramReportFile.txt
      -rV  -rD c:\path\to\all\reports  ^
      -rF path\to\VarReportFile.txt  ^
      -rCS NDAC
```

Where the trailing "^" is the Windows Command file line continuation character.

Running this command will produce a set of report files that are geared towards comparing different versions of the same Machine Edition PLC program. Once this command has been run on both versions, simply use a 3rd party program to compare the report files and identify the differences.

Program Arguments

`Mexml` is a command line based program and as such all of its functionality is defined by its command line arguments. These arguments can be grouped into seven main sections:

Source File

The Source File arguments determine the raw data that `Mexml` will process in order to generate its reports. These include options for Program files as well as Hardware and EGD configuration, and additional user defined information.

Source Processing

The Source Processing arguments provide control over whether certain Source File items will be processed by `Mexml`.

Source Message

The Source Message arguments control the generation of Information, Warning and Error messages produced when processing the Source File arguments.

Program Structure Report

The Program Structure Report arguments are used to select reporting on selected aspects of the Source File items.

Variable Access Report Generation

The Variable Access Report Generation arguments control the production of the Variable Access Report.

Variable Access Report Parameters

The Variable Access Report Parameter arguments determine what variables will be selected for inclusion in the Variable Access report.

Other

These other arguments are used to display information about the `Mexml` program itself, such as version and licensing status.

The following sections provide details on each of the `Mexml` argument groups. Each argument within a section is given as the command line flag required to invoke the argument, then any required parameters, followed by the name of the argument.

Note that arguments surrounded by square brackets (“`[`“ and “`]`”) are optional.

Source File Arguments

Determines the raw data that will be processed by Mexml.

[–X <ProgramFile(s)>] Program Files to process

When this option is specified, Mexml reads and process the contents of the <ProgramFile(s)> , expecting them to be XML formatted program files exported from Machine Edition, and typically having the extension "**.**xml".

Each of these files contain the instructions and variable definitions that define a single program block. Mexml extracts the variable definitions and analyses each instruction to see how it access each particular variable.

The <ProgramFile(s)> can specify any valid Windows path and filename, including the use of Windows filename wildcards. In cases where the <ProgramFile(s)> would match multiple files, then all matching files will be processed.

For example:

```
–X TestData\Code\*.xml

–X TestData\Code\Function1*.xml
```

In the first case, this will include all XML files in the TestData\Code directory. In the second case, this will include all program files that start with "Function1".

When the –X option is not specified, then Mexml will only identify variables that it encountered through processing the other source file options.

The –X argument is optional.

[–E <EGDFile>] EGD Configuration file

When this option is specified, Mexml reads and processes the contents of the <EGDFile>, expecting it to be an EGD Configuration file as exported from Machine Edition, and typically has the extension "**.**egd".

This file is used to identify variables that were not encountered by Mexml when processing the program files, and also to set the Input and Output Scan access criteria of variables that are included in Consumed and Produced exchanges.

The <EGDFile> can specify any valid Windows path and filename, including the use of Windows filename wildcards. However, in cases where the <EGDFile> would match multiple files, only the first matching file will be processed.

For example:

```
–E TestData\EGD\*.egd
```

This will include the first file in the TestData\EGD directory that has the extension "**.**egd".

When the –E option is not specified, then Mexml will only identify variables that it encountered through processing the program files.

The –E argument is optional.

The –E argument requires better than a Trial level license in order to be used.

[–H <HardwareFile>] Hardware Configuration file

When this option is specified, Mexml reads and processes the contents of the <HardwareFile>, expecting it to be a Hardware configuration file as exported from Machine Edition, and typically has the extension "**.**hwc".

This file is used to identify I/O variables that were not encountered by Mexml when processing the program files.

The <HardwareFile> can specify any valid Windows path and filename, including the use of Windows filename wildcards. However, in cases where the <HardwareFile> would match multiple files, only the first matching file will be processed.

For example:

```
–H TestData\Hardware\*.hwc
```

This will include the first file in the TestData\Hardware directory that has the extension "**.**hwc".

When the –H option is not specified, then Mexml will only identify variables that it encountered through processing the program files.

The –H argument is optional.

The –H argument requires better than a Trial level license in order to be used.

[–P <PreloadFile(s)>]+ Preload Variables

When this option is specified, `Mexml` reads and processes the contents of the `<PreloadFile(s)>`, expecting them to be an XML file in the format of variables exported from Machine Edition.

These variables are included prior to processing the program files, and allows for the inclusion of variables that have not been encountered through the file processing.

The `<PreloadFile(s)>` can specify any valid Windows path and filename, including the use of Windows filename wildcards. In cases where the `<PreloadFile(s)>` would match multiple files, then all matching files will be processed.

The '+' append to the `–P` option indicates that it can also be repeated as many times as required in order to include as many different `<PreloadFile>` files as required.

For example:

```
–P TestData\Variables\*.xml –P BaseData\Variables\base.xml
```

This will include all XML files in the `TestData\Variables` directory and the `base.xml` file in the `BaseData\Variables` directory.

When the `–P` option is not specified, then `Mexml` will only identify variables that it encountered through processing the program files.

The `–P` argument is optional.

The `–P` argument requires better than a Trial level license in order to be used.

[-V <VarForceFile(s)>]+ Variable Force Definitions

When this option is specified, `Mexml` reads and processes the contents of the `<VarForceFile(s)>`, expecting them to be plain text files that contain a list of variable names and the Access states to force on those variables.

These forces are processed prior to generating the Variable Access reports, and allow for the specification of access states that were not explicit in the program files. For example variables that are read or written by an HMI system.

The Variable Force Definitions file has a simple text format that follows well defined rules:

1. All blank lines will be ignored.
2. All text including and after "`//`" or "`#`" will be deemed as a comment and will be ignored.
3. Each force definition is made on a single line and can in one of two formats: "`Variable Name`" or "`PLC Address`". Both formats can be freely mixed within the Force Definition file.
4. The `Variable Name` format consists of two fields separated by whitespace. The first fields is the list of forces to apply to the variable and the second field is the name of the variable to which the forces will be applied. This name can be any valid variable name that `Mexml` has encountered when processing the source files.
5. The `PLC Address` format consists of three fields separated by whitespace. The first field is the same as for the Variable Name format and is the list of forces to apply. The second field is the base address at which to start applying those forces, and can be any valid PLC address. The third field is the number PLC addresses to which the forces will be applied. For example for fields 2 and 3 being: `%M101 100`, the same forces will be applied to the PLC memory addresses `%M101` through to `%M200`.
6. In both formats the list of forces can be selected from the following criteria. At least one force has to be specified for each definition, and multiple forces have to be grouped together with no whitespace between them.

   The Force criteria are specified as:

`i`	Is Not Input Scan
`I`	Is Input Scan
`q`	Is Not Output Scan
`Q`	Is Output Scan
`r`	Is Not Read

R	Is Read
w	Is Not Written
W	Is Written
s	Is Not Computed Read
S	Is Computed Read
x	Is Not Computed Write
X	Is Computed Write
b	Is Not Block Access
B	Is Block Access

Where each of the different actions are interpreted as:

Input Scan	A physical input signal (EG %I or %AI), or included in an EGD Consumed exchange.
Output Scan	A physical output signal (EG %Q or %AQ), or included in an EGD Produced exchange.
Read	Has been explicitly read from anywhere in a program file.
Written	Has been explicitly written to anywhere in a program file.
Computed Read	An array element that has been read from by an instruction where the index to the array was a value contained in another variable. As such the actual array index that was read from is only know during program execution.
Computed Write	An array element that has been written to by an instruction where the index to the array was a value contained in another variable. As such the actual array index that was written to is only know during program execution.
Block Access	A Function Block variable that contains all local variables used by that block.

An example of the files format is:

```
// Forces for program

// HMI interface
WQ Variable3
WQ Variable4
RI Variable5
RI Variable6

W  VarArray[10]  // We know element 10 of VarArray is written

I %M101 100  // Input block from Quickpanel
Q %M201 100  // Output block to Quickpanel
```

The <VarForceFile(s)> can specify any valid Windows path and filename, including the use of Windows filename wildcards. In cases where the <VarForceFile(s)> would match multiple files, then all matching files will be processed.

The '+' append to the –V option indicates that it can also be repeated as many times as required in order to include as many different <VarForceFile> files as required.

For example:

```
–V TestData\Data\Forces_*.txt –V BaseData\Data\Test.txt
```

This will include all files that start with Forces_ in the TestData\Data directory, and the Test.txt file from the BaseData\Data directory.

When the –V option is not specified, then the user defined forces will not be included for processing

The –V argument is optional.

The –V argument requires better than a Trial level license in order to be used.

`[-I <InterfaceFile(s)>]+` Function Block Interface Definitions

When this option is specified, `Mexml` reads and processes the contents of the `<InterfaceFile(s)>`, expecting them to be plain text files that contain a list of Function Block Interface definitions.

These definitions are included prior to processing the `-X <ProgramFile(s)>` option and allow for the specification of `View-Locked` Function Blocks for which there is no available program file.

The Interface Definition file has a simple text format that follows well defined rules:

1. All blank lines will be ignored.
2. All text including and after "`//`" or "`#`" will be deemed as a comment and will be ignored.
3. The Interface Definitions for multiple Function Blocks can be included in a single file. Each Function Block definition terminates when a new one starts, and a Function Block can only ever be defined once.
4. The name of a Function Block at the start of a line in the file designates the start of the definitions for that function block.
5. The parameter definitions for a single function block are split into 4 columns, each separated by whitespace. In addition the first column has to have whitespace between itself and the start of the line it is on. Each column is defined as follow and must be in this order:
    A) Class: `I` for Input, `O` for Output and `X` for Input/Output.
    B) Name: Any valid ME name, that is unique within its own Function Block.
    C) Type: Any valid ME data type, such as `INT`, `BOOL`, `WORD`, etc. Array datatypes can be included through the use of square brackets: `BOOL[24]`, indicating the number of elements in the array
    D) Description: Any valid text description of the parameter.

An example of the files format is:

```
// Estop Detection block
ESTOPDET
 I IN  BOOL[32] Estop Status Array
 I RST BOOL     Estop alarm reset
 O Q   BOOL[32] Estop alarm Status
 O NE  BOOL     No Estop Active
```

The <InterfaceFile(s)> can specify any valid Windows path and filename, including the use of Windows filename wildcards. In cases where the <InterfaceFile(s)> would match multiple files, then all matching files will be processed.

The '+' append to the –I option indicates that it can also be repeated as many times as required in order to include as many different <InterfaceFile> files as required.

For example:

```
–I TestData\Library\BlockDef?.txt –I BaseData\Library\Base.txt
```

This will include all files in the TestData\Library directory that start with BlockDef and have an additional character in their name, as well as the Base.txt file from the BaseData\Library directory.

When the –I option is not specified, then the user defined Function Block Interface definitions will not be included for processing.

The –I argument is optional.

The –I argument requires better than a Trial level license in order to be used.

Source Processing Arguments

Determines how the Source Files should be processed.

[–xKA] Skip analysis of all Program Blocks

When this option is specified, the –X <ProgramFile(s)> option program blocks will be read, but not processed. When used in-conjunction with the –xF0 option, this enables the dependency order of a group of program block to be easily determined.

When the –xKA option is not specified, all program blocks from the –X option will be processed.

The –xKA argument is optional.

The –xKA argument requires better than a Trial level license in order to be used.

[–xKF] Skip processing of Function Blocks

When this option is specified, program blocks from the –X <ProgramFile(s)> option that are Function Blocks will not be processed.

When the –xKF option is not specified, all program blocks from the –X option will be processed.

The –xKF argument is optional.

The –xKF argument requires better than a Trial level license in order to be used.

[–aBD] Auto-Generated Variables Blank Descriptions

When this option is specified, the descriptions of variables that are automatically generated by `Mexml` will be blank.

When the `–aBD` option is not specified, the descriptions of automatically generated variables will document the circumstances by which `Mexml` decided to generate that variable.

The `–aBD` argument is optional.

The `–aBD` argument requires better than a Trial level license in order to be used.

Source Message Arguments

Determines what types of Informational, Warning and Error messages will be generated by `Mexml` when it is processing the Source Files.

`[-mV]` Generate All Program Structure messages

When this option is specified, all possible messages will be added to the Program Structure report.

Note that this option can't be used with the `–mQ` option.

The `–mV` argument is optional.

The `–mV` argument requires better than a Trial level license in order to be used.

[–mNW] Suppress Program Structure Warning messages

When this option is specified, no Warning messages will be added to the Program Structure report. However Informational and Error messages will still be added.

Note that this option can't be used with the –mQ option.

The –mNW argument is optional.

The –mNW argument requires better than a Trial level license in order to be used.

[-mQ] Suppress All Program Structure messages

When this option is specified, no messages will be added to the Program Structure report.

Note that this option can't be used with the `-mV` or `-mNW` options.

The `-mQ` argument is optional.

The `-mQ` argument requires better than a Trial level license in order to be used.

[–mRN <NameRegex>] Filter source file messages by Name Regex

When this option is specified, only program block names from the –X <ProgramFile(s)> option that match the given <NameRegex> regular expression will generate Informational messages.

For example:

```
-X Testdata\Code\*.xml -mRN ^Device1.* -xLB -XLI
```

Indicates that all program blocks (*.xml) in the Testdata\Code directory will be processed, but that only those with names that match the ^Device1.* regular expression (EG start with "Device1") will generate the -xLB and -xLI Informational messages that will be added to the Program Structure report.

When the –mRN option is not specified, then all specified messages will be added to the Program Structure report.

The –mRN argument is optional.

The –mRN argument requires better than a Trial level license in order to be used.

[–xLB] Ladder Logic – Body (Code) messages

When this option is specified, a processed version of the code in the body of any encountered Ladder Logic program block will be added to the Program Structure report. This processed version will list all instructions in a textual format, grouped by their rung. By default (unless the -xLRS option is also invoked) the each instruction will be annotated with the rung number.

Note that these instructions will be listed in a left-to-right, top-to-bottom order with respect to how the rung is implemented. Thus in complex rungs it will be typical to see coil instructions listed before the contacts that drive them. If it is desired to understand the actual processing order, then the –xLA option will annotate each instruction with its ID number, that indicates which instructions are the precursors to others.

For example:

```
===================================
  AABlock  [ Block,  LadderLogic ]
===================================

 ~~~~~~~~
   Body
 ~~~~~~~~
  Ladder Logic
   LeftPowerRail

    1  COMMENT Executes the main logic of the AA function

    2  NOCON PumpA_Running
    2  NCCON ControlRmEmpty
    2  COIL PumpLight_1
    2  NOCON PumpB_Running
    2  COIL PumpLight_2

    3  ...
```

When the –xLB option is not specified, the code will not be added to the Program Structure report.

Note that all Error and Warning messages related to the processing of the code will be included in the Program Structure report regardless of the inclusion of this option. Such messages can only be suppressed through the use of the –mNW and –mQ options.

The –xLB argument is optional.

[-xLI] Ladder Logic - Interface messages

When this option is specified, information messages describing the Interface section of any encountered Ladder Logic program blocks will be added to the program structure report.

The Interface lists all of the variables that are referenced by a particular program block. (However it only indicates the data types of non-derived variables. To see the data types of the derived types, the –xLD option should be used in addition to this option.)

For example:

```
====================================
  AABlock  [ Block,  LadderLogic ]
====================================

 ~~~~~~~~~~~~~
   Interface
 ~~~~~~~~~~~~~

   --------------
     globalVars
   --------------
    Var1l           Derived
    Var1l.SubVar1   Derived  Comment for SubVar1
    Var1l.SubVar2   Derived  Comment for SubVar2
    Var1l.SubVar3   Derived  Comment for SubVar3
    Var1l.SubVar4   Derived  Comment for SubVar4
    Var1l.SubVar5   Derived  Comment for SubVar5
    ...
    Variable3       REAL     Comment for Variable3
    Variable4       INT      Comment for Variable4
    Variable5       INT      Comment for Variable5
    ...
```

When the –xLI option is not specified, the informational messages will not be added to the Program Structure report.

Note that all Error and Warning messages related to the processing of the Interface will be included in the Program Structure report regardless of the inclusion of this option. Such messages can only be suppressed through the use of the –mNW and –mQ options.

The –xLI argument is optional.

[-xLD] Ladder Logic - Derived data type and UDT messages

When this option is specified, information messages describing the Derived data type and User Defined data type variables used by any encountered Ladder Logic program blocks will be added to the Program Structure report.

Derived and User defined data types are both forms of compound data types that can be defined in a program. They are defined in a program block as a GUID combined with a list of sub-elements that compose their makeup. The program block then defines a list of variables that are of that data type. This option lists both the datatype with its GUID and component parts, as well as the list of variables defined as each data type.

For example:

```
====================================
  AABlock  [ Block,  LadderLogic ]
====================================

 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   Derived Type To Variable Mappings
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  -----------------
    Derived Types
  -----------------
    {9F9BDCE1-4B31-4204-8BDB-D82E53A8C7A2}
        SubVar1     INT
        SubVar2     BOOL
        SubVar3     INT
        SubVar4     INT
        SubVar5     BOOL
        ...

  ---------------------
    Derived Variables
  ---------------------
    Var1 {9F9BDCE1-4B31-4204-8BDB-D82E53A8C7A2} Standard
        Var1.SubVar1     INT
        Var1.SubVar2     BOOL
        Var1.SubVar3     INT
        Var1.SubVar4     INT
        Var1.SubVar5     BOOL
        ...

    Var2 {9F9BDCE1-4B31-4204-8BDB-D82E53A8C7A2} Standard
        Var2.SubVar1     INT
        Var2.SubVar2     BOOL
        Var2.SubVar3     INT
        Var2.SubVar4     INT
        Var2.SubVar5     BOOL
        ...
```

Note that in order to avoid duplicating redundant information, `Mexml` will only fully expand a particular Derived data type, or variable of that type, the first time it is encountered. For all subsequent encounters, the reference will be marked as "`(previously processed)`".

When the `–xLD` option is not specified, the informational messages will not be added to the Program Structure report.

Note that all Error and Warning messages related to the processing of derived data types will be included in the Program Structure report regardless of the inclusion of this option. Such messages can only be suppressed through the use of the `–mNW` and `–mQ` options.

The `–xLD` argument is optional.

The `–xLD` argument requires better than a Trial level license in order to be used.

[–xLA] Ladder Logic - Annotate instructions with their IDs

When this option is specified in conjunction with the –xLB option, the instructions in the program listing generated by the –xLB option will be annotated with their ID numbers, as well as the ID numbers of their parameters. This allows the actual data flow of a rung to be better visualized from the non-ordered list of instructions.

For example, the following is the same code without and with annotations:

```
====================================
  AABlock  [ Block,  LadderLogic ]
====================================

 ~~~~~~~~
   Body
 ~~~~~~~~
  Ladder Logic
   LeftPowerRail

   1  COMMENT Executes the main logic of the AA function

   2  NOCON PumpA_Running
   2  NCCON ControlRmEmpty
   2  COIL PumpLight_1
   2  NOCON PumpB_Running
   2  COIL PumpLight_2

   3  ...
```

```
====================================
  AABlock  [ Block,  LadderLogic ]
====================================

 ~~~~~~~~
   Body
 ~~~~~~~~
  Ladder Logic
   {0}.LeftPowerRail

   1  COMMENT Executes the main logic of the AA function

   2  NOCON {32}.PumpA_Running <= (LeftPwrRail)
   2  NCCON {33}.ControlRmEmpty <= {32}, {63}
   2  COIL {34}.PumpLight_1 <= {33}
   2  NOCON {63}.PumpB_Running <= (LeftPwrRail)
   2  COIL {65}.PumpLight_2 <= {33}

   3  ...
```

Thus in rung 2, both `COIL 34` and `COIL 65` are driven by `NOCON 33`, which in

turn is driven by `NOCON 32` or `NOCON 63`.

When the `–xLA` option is not specified, the program listing generated by the `–xLB` option will not be annotated with these ID's.

The `–xLA` argument is optional.

The `–xLA` argument requires better than a Trial level license in order to be used.

[–xLRS] Ladder Logic – Suppress Rung Numbers

When this option is specified in conjunction with the –xLB option, the instructions in the program listing generated by the –xLB option will have their rung numbers suppressed. This facilitates comparing the listing files of two versions of the same program by removing visual data that is likely to differ between them, but not actually impact on functionality

For example, the following is the same code without and with rung numbers:

```
====================================
  AABlock  [ Block,  LadderLogic ]
====================================

 ~~~~~~~~
   Body
 ~~~~~~~~
  Ladder Logic
   LeftPowerRail

   1  COMMENT Executes the main logic of the AA function

   2  NOCON PumpA_Running
   2  NCCON ControlRmEmpty
   2  COIL PumpLight_1
   2  NOCON PumpB_Running
   2  COIL PumpLight_2

   3  ...
```

```
====================================
  AABlock  [ Block,  LadderLogic ]
====================================

 ~~~~~~~~
   Body
 ~~~~~~~~
  Ladder Logic
   LeftPowerRail

      COMMENT Executes the main logic of the AA function

      NOCON PumpA_Running
      NCCON ControlRmEmpty
      COIL PumpLight_1
      NOCON PumpB_Running
      COIL PumpLight_2

      ...
```

When the –xLRS option is not specified, the program listing generated by the

–xLB option will not be annotated with rung numbers.

The –xLRS argument is optional.

The –xLRS argument requires better than a Trial level license in order to be used.

[–xSB] Structured Text – Body (Code) messages

When this option is specified, a processed version of the code in the body of any encountered Structured Text program block will be added to the Program Structure report. This processed version will have all program comments and blank lines of the original code stripped out from it.

For example:

```
=========================================
  StructText1  [ Block,  StructuredText ]
=========================================

 ~~~~~~~~
  Body
 ~~~~~~~~
  Structured Text
      Variable1 := 1024.0
      Variable2 := Variable1
      Variable3 := 139.21
      ...
```

When the –xSB option is not specified, the code will not be added to the Program Structure report.

Note that all Error and Warning messages related to the processing of the code will be included in the Program Structure report regardless of the inclusion of this option. Such messages can only be suppressed through the use of the –mNW and –mQ options.

The –xSB argument is optional.

[-xSI] Structured Text - Interface messages

When this option is specified, information messages describing the Interface section of any encountered Structured Text program blocks will be added to the program structure report.

The Interface lists all of the variables that are referenced by a particular program block. (However it only indicates the data types of non-derived variables. To see the data types of the derived types, the –xSD option should be used in addition to this option.)

For example:

```
=========================================
  StructText1  [ Block,  StructuredText ]
=========================================

 ~~~~~~~~~~~~~
   Interface
 ~~~~~~~~~~~~~

   --------------
     globalVars
   --------------
    Var1l           Derived
    Var1l.SubVar1   Derived  Comment for SubVar1
    Var1l.SubVar2   Derived  Comment for SubVar2
    Var1l.SubVar3   Derived  Comment for SubVar3
    Var1l.SubVar4   Derived  Comment for SubVar4
    Var1l.SubVar5   Derived  Comment for SubVar5
    ...
    Variable3       REAL     Comment for Variable3
    Variable4       INT      Comment for Variable4
    Variable5       INT      Comment for Variable5
    ...
```

When the –xSI option is not specified, the informational messages will not be added to the Program Structure report.

Note that all Error and Warning messages related to the processing of the Interface will be included in the Program Structure report regardless of the inclusion of this option. Such messages can only be suppressed through the use of the –mNW and –mQ options.

The –xSI argument is optional.

[-xSD] Structured Text - Derived data type and UDT messages

When this option is specified, information messages describing the Derived data type and User Defined data type variables used by any encountered Structured Text program blocks will be added to the Program Structure report.

Derived and User defined data types are both forms of compound data types that can be defined in a program. They are defined in a program block by a GUID combined with a list of sub-elements that compose their makeup. The program block then defines a list of variables that are of that data type. This option lists both the datatype with its GUID and component parts, as well as the list of variables defined as each data type.

For example:

```
==========================================
  StructText1  [ Block,  StructuredText ]
==========================================

 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   Derived Type To Variable Mappings
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  -----------------
    Derived Types
  -----------------
    {9F9BDCE1-4B31-4204-8BDB-D82E53A8C7A2}
        SubVar1     INT
        SubVar2     BOOL
        SubVar3     INT
        SubVar4     INT
        SubVar5     BOOL
        ...

  ---------------------
    Derived Variables
  ---------------------
    Var1 {9F9BDCE1-4B31-4204-8BDB-D82E53A8C7A2} Standard
        Var1.SubVar1     INT
        Var1.SubVar2     BOOL
        Var1.SubVar3     INT
        Var1.SubVar4     INT
        Var1.SubVar5     BOOL
        ...

    Var2 {9F9BDCE1-4B31-4204-8BDB-D82E53A8C7A2} Standard
        Var2.SubVar1     INT
        Var2.SubVar2     BOOL
        Var2.SubVar3     INT
        Var2.SubVar4     INT
        Var2.SubVar5     BOOL
        ...
```

Note that in order to avoid duplicating redundant information, `Mexml` will only fully expand a particular Derived data type, or variable of that type, the first time it is encountered. For all subsequent encounters, the reference will be marked as "`(previously processed)`".

When the `–xSD` option is not specified, the informational messages will not be added to the Program Structure report.

Note that all Error and Warning messages related to the processing of derived data types will be included in the Program Structure report regardless of the inclusion of this option. Such messages can only be suppressed through the use of the `–mNW` and `–mQ` options.

The `–xSD` argument is optional.

The `–xSD` argument requires better than a Trial level license in order to be used.

[–xSA] Structured Text - Annotate code with Read/Write markers

When this option is specified in conjunction with the –xSB option, the program listing generated by the –xSB option will be annotated with markers that indicate whether a variable had Read or Write access. The annotations involve enclosing each variable name with a specific type of bracket:

< .. **>** Write Access

{ .. **}** Read Access

For example, the following is the same code without and with annotations:

```
...
Variable1 := 1024.0
Variable2 := Variable1
Variable3 := 139.21
...
```

```
...
<Variable1> := 1024.0
<Variable2> := {Variable1}
<Variable3> := 139.21
...
```

When the –xSA option is not specified, the program listing generated by the –xSB option will not be annotated with these markers.

The –xSA argument is optional.

The –xSA argument requires better than a Trial level license in order to be used.

[–xF0] File Processing Order messages

When this option is specified, informational messages related to the order in which the –X <ProgramFile(s)> option files will be processed, will be added to the Program Structure report.

For example:

```
################################
  Program File Processing Order
################################

File path: J:\Mexml\TestData\Code

Files sorted from least to most dependent
-----------------------------------------
J:\Mexml\TestData\Code\Setup.xml
J:\Mexml\TestData\Code\AABlock.xml
J:\Mexml\TestData\Code\BFunction.xml
J:\Mexml\TestData\Code\CCBlock.xml
J:\Mexml\TestData\Code\DDBlock.xml
J:\Mexml\TestData\Code\BBBlock.xml
J:\Mexml\TestData\Code\_MAIN.xml
J:\Mexml\TestData\Code\EEBlock.xml
```

Combining this option with the –xKA option indicates the order that a block of program files needs to be imported into another project.

When not specified, the informational messages will not be added to the Program Structure report.

Note that all Error and Warning messages related to the processing of the –X <ProgramFile(s)> option files will be included in the Program Structure report regardless of the inclusion of this option. Such messages can only be suppressed through the use of the –mNW and –mQ options.

The –xF0 argument is optional.

The –xF0 argument requires better than a Trial level license in order to be used.

[-eM] EGD Configuration messages

When this option is specified, informational messages related to the processing of the –E <EGDFile> option will be added to the Program Structure report. The main use of this option is to document the data that is consumed or produced by an EGD exchange.

For example:

```
#########################
  EGD Configuration File
#########################

Processing 1 file from
J:\Mexml\TestData\_DBGReports\..\Raven\Data\EGD\

 ========================================
   Processing J:\Mexml\TestData\EGD\PLC.egd
 ========================================

  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    Exch01_Prod_Refs [ Produced => 192.168.7.156 ]
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   Named: Exch01_Status
                    WORD      Exch01_Status

   Named: Source1.Refs
                    WORD[10]  Produced.Refs
                    WORD      Produced.Refs[0]
                    WORD      Produced.Refs[1]
                    WORD      Produced.Refs[2]
                    WORD      Produced.Refs[3]
                    WORD      Produced.Refs[4]
                    WORD      Produced.Refs[5]
                    WORD      Produced.Refs[6]
                    WORD      Produced.Refs[7]
                    WORD      Produced.Refs[8]
                    WORD      Produced.Refs[9]

   Addr: %Q00001 32 BOOL
         %Q00001    BOOL      System_CtrlOn_IL7
         %Q00002    BOOL      Q00002
         %Q00003    BOOL      System_CtrlOn_IL8
         %Q00004    BOOL      ActTrp_Up_IL8 BOOL
         %Q00005    BOOL      ActTrp_Dn_IL8 BOOL
         %Q00006    BOOL      ActTrp_StckPos_IL8
```

When not specified, the informational messages will not be added to the Program Structure report.

Note that all Error and Warning messages related to the processing of the –E <EGDFile> option will be included in the Program Structure report regardless of

the inclusion of this option. Such messages can only be suppressed through the use of the `–mNW` and `–mQ` options.

The `–eM` argument is optional.

The `–eM` argument requires better than a Trial level license in order to be used.

[–eDS] Suppress EGD Configuration Descriptions

This option only works in conjunction with the –eM option. When the –eM option is specified, messages describing how variables are allocated to produced and consumed EGD exchanges are added to the Program Structure report. This information includes the variables Address, Data Type, Name and Description.

When the –eDS option is specified along with the –eM option, then Description aspect of the variables will not be added to the Program Structure report. Using this option can benefit in comparing two different EGD configurations to one another by removing information that is not relevant to the PLC program.

When the –eM option is specified, and the –eDS option is not specified, then the Description aspect of the variables will be included in the Program Structure report.

The –eDS argument is optional.

The –eDS argument requires better than a Trial level license in order to be used.

[-hM] Hardware Configuration messages

When this option is specified, informational messages related to the processing of the –H <HardwareFile> option will be added to the Program Structure report. The main use of this option is to document the data that is mapped to each I/O card.

For example:

```
##############################
  Hardware Configuration File
##############################

Processing 1 file from J:\Mexml\TestData\Hardware\

  ======================================================
    Processing J:\Mexml\TestData\Hardware\PLC_SR.hwc
  ======================================================

  ~~~~~~~~~~~~~~~~~~~~~~~~~
    Rack 0 - IC695CHS016
  ~~~~~~~~~~~~~~~~~~~~~~~~~

  Slot  Card Type
    0   IC695PSA040
    1
    2   IC695CPE305
         ...
    3

    4   IC695ETM001
         Main   %I01793 80 BOOL                   Status Addr
                %I01793    BOOL[80] ETM_Status    Ethernet I/F
                %I01793    BOOL     ETM_Status[0] 1A full dplx
                %I01794    BOOL     ETM_Status[1] 1A 100Mbps
                %I01795    BOOL     ETM_Status[2] 1B full dplx
                %I01796    BOOL     ETM_Status[3] 1B 100Mbps
                   ...
```

When not specified, the informational messages will not be added to the Program Structure report.

Note that all Error and Warning messages related to the processing of the –H <HardwareFile> option will be included in the Program Structure report regardless of the inclusion of this option. Such messages can only be suppressed through the use of the –mNW and –mQ options.

The –hM argument is optional.

The –hM argument requires better than a Trial level license in order to be used.

[-hDS] Suppress Hardware Configuration Descriptions

This option only works in conjunction with the –hM option. When the –hM option is specified, messages describing how variables are allocated to explicit hardware I/O locations are added to the Program Structure report. This information includes the variables Address, Data Type, Name and Description.

When the –hDS option is specified along with the –hM option, then Description aspect of the variables will not be added to the Program Structure report. Using this option can benefit in comparing two different Hardware configurations to one another by removing information that is not relevant to the PLC program.

When the –hM option is specified, and the –hDS option is not specified, then the Description aspect of the variables will be included in the Program Structure report.

The –hDS argument is optional.

The –hDS argument requires better than a Trial level license in order to be used.

[-pM] PreLoad Variable messages

When this option is specified, informational messages related to the processing of the –P <PreloadFile(s)> option will be added to the Program Structure report.

For example:

```
#########################
  Preload Variables File
#########################

Processing 1 file from J:\Mexml\TestData\Variables\

  ====================================================
    Processing J:\Mexml\TestData\Variables\AllVars.xml
  ====================================================
    $DrvCommChkTm      INT      Drive comm check time [ms]
    $ystem_FaultRst    BOOL     System reset pulse
    AccReg7            WORD[3]
    AccReg7[0]         WORD
    AccReg7[1]         WORD
    AccReg7[2]         WORD
```

When not specified, the informational messages will not be added to the Program Structure report.

Note that all Error and Warning messages related to the processing of the –P <PreloadFile(s)> option will be included in the Program Structure report regardless of the inclusion of this option. Such messages can only be suppressed through the use of the –mNW and –mQ options.

The –pM argument is optional.

The –pM argument requires better than a Trial level license in order to be used.

[-vM] Variable Force messages

When this option is specified, messages related to processing the –V <ForceFile(s)> option will be added to the Program Structure report.

For example:

```
######################
  Variable Force File
######################

Processing 1 file from J:\Mexml\TestData\Forces\

  ================================================
    Processing J:\Mexml\TestData\Forces\Forces.txt
  ================================================
    I => Aux1.AckFaultTrig
    I => Aux1.ClrFaultTrig
    I => Aux2.AckFaultTrig
    I => Aux2.ClrFaultTrig
    ...
```

When not specified, the messages will not be added to the Program Structure report.

Note that all Error and Warning messages related to the processing of the –V <ForceFile(s)> option will be included in the Program Structure report regardless of the inclusion of this option. Such messages can only be suppressed through the use of the –mNW and –mQ options.

The –vM argument is optional.

The –vM argument requires better than a Trial level license in order to be used.

[-iM] Function Block Interface Definition messages

When this option is specified, messages related to processing the –I <InterfaceFile(s)> option will be added to the Program Structure report.

For example:

```
##########################################
  Function Block Interface Definition File
##########################################

Processing 1 file from J:\Mexml\TestData\

  ================================================
    Processing J:\Mexml\TestData\ExternalBlocks.txt
  ================================================

    ~~~~~~~~~~~~
      ESTOPDET
    ~~~~~~~~~~~~
      I  IN   BOOL[32]  Estop Status Array
      I  RST  BOOL      Estop alarm reset
      O  Q    BOOL[32]  Estop alarm Status
      O  NE   BOOL      No Estop Active

     ...
```

When not specified, the messages will not be added to the Program Structure report.

Note that all Error and Warning messages related to the processing of the –I <InterfaceFile(s)> option will be included in the Program Structure report regardless of the inclusion of this option. Such messages can only be suppressed through the use of the –mNW and –mQ options.

The –iM argument is optional.

The –iM argument requires better than a Trial level license in order to be used.

[-kCP] Linked Variable Chaining and Propagating messages

When this option is specified, information messages related to the Chaining and Propagating of values between Linked Variables will be added to Program Structure report. There are three classes of such variables:

1. Variables that have the name of another variable as their PLC address (as opposed to having a numerical value). Thus the access state of any one variable in the chain is the combination of access states of all variables in the chain.
2. Individual array elements that are implicitly linked to the array as a whole. Thus the access state of an element is that of itself combined with that of the overall array.
3. Variables that are mapped onto the same address value as another variable (For example booleans that are mapped onto a word). Thus the access state of the mapped variables is that of themselves combined with the that of the variable they are mapped onto.

For example:

```
##########################################
  Chaining and Propagating variable status
##########################################

  ============================================================
    Extending array non-numeric addresses to elements
  ============================================================
    (No arrays needed their non-numeric addresses extended)

  ============================================================
    Propagating array definition status values to elements
  ============================================================
    BoolDataIn BOOL[8] Irswxbq
       => BoolDataIn[0]  IRswxbq OR Irswxbq = IRswxbq
       => BoolDataIn[1]  IRswxbq OR Irswxbq = IRswxbq
       => BoolDataIn[2]  Irswxbq OR Irswxbq = Irswxbq
       => BoolDataIn[3]  IRswxbq OR Irswxbq = IRswxbq
       => BoolDataIn[4]  Irswxbq OR Irswxbq = Irswxbq
       => BoolDataIn[5]  IRswxbq OR Irswxbq = IRswxbq
       => BoolDataIn[6]  IRswxbq OR Irswxbq = IRswxbq
       => BoolDataIn[7]  IRswxbq OR Irswxbq = Irswxbq
    ...

  ============================================
    Chaining variables to their parent variable
  ============================================
    Simple: ChildVar1 => ParentVar1
    Simple: ChildVar2 => Parentvar2
    ...
```

```
================================================
  Propagating variable status up and down chains
================================================

  ~~~~~~~~~~~~~~~~~~~
    Child to Parent
  ~~~~~~~~~~~~~~~~~~~
    ChildVar1 => ParentVar1
    ChildVar2 => Parentvar2
    ...

  ~~~~~~~~~~~~~~~~~~~
    Parent To Child
  ~~~~~~~~~~~~~~~~~~~
    ParentVar1
      => ChildVar1

    ParentVar2
      => ChildVar2
    ...


==========================================================
  Propagating mapped address variable to sub-variables
==========================================================
  %M00105 WORD InputAsWord 16 Bit irsWxbq
    %M00105
    %M00106 BOOL LostKeyBit =>iRsWxbq
    %M00107
    %M00108 BOOL OnFireBit =>iRsWxbq
    %M00109 BOOL SinkHoleBit =>iRsWxbq
    %M00110
    %M00111
    %M00112 BOOL MorningBit =>iRsWxbq
    %M00113
    %M00114
    %M00115
    %M00116
    %M00117
    %M00118
    %M00119
    %M00120

==========================================================
  Propagating array element status values to definition
==========================================================

    BoolDataIn BOOL[8] Irswxbq OR Irswxbq = Irswxbq
    ...
```

When not specified, the messages will not be added to the Program Structure report.

Note that all Error and Warning messages related to the Chaining and Propagation will be included in the Program Structure report regardless of the inclusion of this option. Such messages can only be suppressed through the use of the `–mNW` and `–mQ` options.

The `–kCP` argument is optional.

The `–kCP` argument requires better than a Trial level license in order to be used.

[-rM] Variable Report messages

When this option is specified, statistics about the Variable Access report will be added to the Program Structure report.

For example:

```
================
  Report 1 of 1
================
  Directed to file C:\reports\VariableUsage.txt
  Reporting as: Plain text
  Report is built:
    With a name Regex '^(?!(W0|#|T000)).*'
    With all addresses
    With all comments
    With access criteria 'iRw'
    Without memory criteria
    Without type criteria
    Without sort criteria
  7 variables matched the selection criteria
```

Note that this option will not include any of the actual variable access data.

When not specified, the statistics will not be added to the Program Structure report.

The –rM argument is optional.

[–O <OutputFile>] Program Structure report file

When this option is specified, the output of the report will be saved to this filename, overwriting any previously existing file of the same name. Note that this does not include data that is generated by the Variable Access report, however it can include some statistics about that report. See the "–rF" option for saving the Variable Access report to a file.

The Program Structure report file can be specified using any valid Windows path and filename combination.

For example:

```
–O c:\data\reports\StructReport.txt

–O ..\..\reports\StructReport.txt
```

When the –0 option is not specified, the Program Structure report data will be sent to the console output.

The –0 argument is optional.

The –0 argument requires better than a Trial level license in order to be used.

Program Structure Report Arguments

Reports on various aspects of the program and variable structure (not including variable access)

[-sCO] Program Block Call Order

When this option is specified, the order of execution of all Program Blocks specified in the -X <ProgramFiles(s)> option will be listed in the Program Structure report. In addition this option will also identify any Program Blocks that were not called for execution.

For example:

```
============================
  Blocks that are not called
============================
  EEBlock

=========================
  Blocks that are called
=========================
  _MAIN
    Setup
    CCBlock
      AABlock
      BFUNCTION
    DDBlock
      AABlock
      BFUNCTION
    BBBlock
```

Shows that the _Main block calls Setup, CCBlock etc. And CCBlock calls AABlock etc. While the EEBlock has not been called by any of the other POUs.

When not specified, the Call Order report will not be generated.

The -sCO argument is optional.

The -sCO argument requires better than a Trial level license in order to be used.

[-sMA] Mapped Address variables

When this option is specified, the mappings of all variables that have overlapping addresses will be listed in the Program Structure report.

This option is useful to provide a list of items such as bits that have been mapped onto a Word based variable.

For example:

```
%M00105 WORD InputAsWord
    %M00105
    %M00106 BOOL LostKeyBit
    %M00107
    %M00108 BOOL OnFireBit
    %M00109 BOOL SinkHoleBit
    %M00110
    %M00111
    %M00112 BOOL MorningBit
    %M00113
    %M00114
    %M00115
    %M00116
    %M00117
    %M00118
    %M00119
    %M00120
```

Shows 4 boolean variables that have been mapped onto the same address space as a word variable.

When not specified, the Mapped Address report will not be generated.

The –sMA argument is optional.

The –sMA argument requires better than a Trial level license in order to be used.

Variable Access Report Generation Arguments

Determines if and where Variable Access reports will be generated.

[-rV] Generate Variable Access Reports

When this option is specified, `Mexml` will generate Variable Access reports as defined by all other reporting arguments.

When not specified, then no Variable Access reports will be generated.

The `-rV` argument is optional.

[-rD <ReportDir>] Base directory for all report files

When this option is specified, the <ReportDir> will be prepended to all defined "-rF <ReportFile>" name options. The combined name will define the actual filename to where the report data will be saved.

For example:

```
-rD c:\data\reports -rF ThisReport.txt
```

Will save the report data to the file "c:\data\reports\ThisReport.txt". As such, the -rF option can only specify the file using a relative path.

The benefits of combining the -rD and -rF options come into play when used in conjunction with the "-rP <ReportParaFile>" option. This enables multiple reports to be generated from the same source files, with the data from each report being written to its own specific file.

When the -rD option is not specified, then the destination of the report data will solely be determined by the -rF option.

The -rD argument is optional.

The -rD argument requires better than a Trial level license in order to be used.

[-rP <ReportParamFile>] Specify parameters for multiple reports

When this option is specified, Mexml will process the source files and then generate multiple Variable Access reports using the Report Parameters that are stored within the <ReportParamFile>.

The <ReportParamFile> is a simple text file of any arbitrary name and can be specified using absolute or relative paths. Each line of the file defines the Report Parameters of an additional report, and the "definition" is simply the list of Report Parameters as would have been specified directly on the Mexml command line. The choice of allowable options is any option that is mentioned in the "Variable Access Report Parameter Arguments" section of this document.

In addition, any text following after either "//" or "#" is consider a comment and will be ignored, as will any blank lines or extra whitespace.

For example, if the file contained:

```
// Define all reports
-fM I -rS D -rF Report1.txt
-fM O -rS n -fA w -rF Report2.txt
-fA Q -fRN Output.* -rF Report3.txt
```

Then this would specify 3 reports with the following Report Parameters:

Report #1	`-fM I -rS D -rF Report1.txt`
Report #2	`-fM O -rS n -fA w -rF Report2.txt`
Report #3	`-fA Q -fRN Output.* -rF Report3.txt`

Because the -rF option was used, each report will be saved to a different file.

When not specified, only a single Variable Access report will be generated, based on the Report Parameters specified on the Mexml command line.

Note that if the -rP option is specified, then Mexml will still generate a report based on any Report Parameters that were specified on it s command line.

The -rP argument is optional.

The -rP argument requires better than a Trial level license in order to be used.

Variable Access Report Parameter Arguments

Determines how the variable data will be filtered in order to generate the Variable Access reports, as well as in what format the report will be generated.

Each of the filter criteria types can be independently specified, with the overall report containing data that matches all of the criteria AND'ed together. For example, if these criteria are all specified for a specific report:

- Access Criteria of Input Scan
- Memory Criteria of %M
- Type Criteria of Global

Then only variables that are defined as Input Scan AND %M AND Global will appear in the report.

An example of a Variable Access report is:

```
#######################################################
  Variables Report, created at 3/30/2014 2:35:56 PM
#######################################################

  ====================
    Report Parameters
  ====================

    ~~~~~~~~~~~~~~~~
      Source Files
    ~~~~~~~~~~~~~~~~
      Program files:             J:\Mexml\TestData\...
       ...

    ~~~~~~~~~~~~~~~~~~~~
      Filter Parameters
    ~~~~~~~~~~~~~~~~~~~~
      Name Regex:       -fRN ^(?!(W0|#|T000)).*
      Access Criteria:  -fA iRw
                            iRw [Not InputScan] [Read]
                                  [No Write]

    ==============
      Sort Order
    ==============
      (Default to By Name)

    =========
      Other
    =========
      Selected Columns: -rCS NDAVCM     ([Name] [Data Type]
```

```
                            [Address] [Variable Definition]
                            [Access Criteria] [Memory Group])


#########################################################
                      Report Data
#########################################################

                                 Definit. Access  Mem.
Name              Type Address   12345678 1234567 1234
----------------- ---- ------- -------- ------- ----
BoolDataIn2       BOOL          IGEV---- -R----- MS--
BoolDataIn4       BOOL          IGEV---- -R----- MS--
Data_5            INT  %R00154 IGEV--P- -R----- MNR-
Data_6            INT  %R00155 IGEV--P- -R----- MNR-
DDBlock.CLR_Fancy BOOL          ILEV---- -R----- MS--
Number1           DINT %R00021 IGEV--P- -R----- MNR-
Number2           DINT %R00022 IGEV--P- -R----- MNR-
```

This shows the source data that was used to compile the report, the parameters used to select the variables for reporting and the details of each variable. In this case the report is selecting variables that have been read and not written, IE Uninitialized variables.

The following sections describe all the parameters used to define a single Variable Access report.

[-fA <AccessCriteria>] Filter report by each variables Access Criteria

When this option is specified, the report data will only include variables that have been accessed by the given Access Criteria action. All other variables will be excluded from the report.

The Access Criteria actions are specified as:

- `i` Is Not Input Scan
- `I` Is Input Scan
- `q` Is Not Output Scan
- `Q` Is Output Scan
- `r` Is Not Read
- `R` Is Read
- `w` Is Not Write
- `W` Is Write
- `s` Is Not Computed Read
- `S` Is Computed Read
- `x` Is Not Computed Write
- `X` Is Computed Write
- `b` Is Not Block Access
- `B` Is Block Access

Where each of the different actions are defined as:

`Input Scan`	A physical input signal (EG %I or %AI), or included in an EGD Consumed exchange.
`Output Scan`	A physical output signal (EG %Q or %AQ), or included in an EGD Produced exchange.
`Read`	Has been explicitly read from anywhere in a program file.
`Write`	Has been explicitly written to anywhere in a program file.
`Computed Read`	An array element that has been read from by an instruction where the index to the array was a value contained in another variable. As such the actual array index that was read from is only know during program execution.
`Computed Write`	An array element that has been written to by an instruction where the index to the array was a value contained in another variable. As such the actual array index that was written to is only know during program execution.

`Block Access`	A Function Block variable that contains all local variables used by that block.

Multiple Access Criteria can be specified, in which case only variables that match all of the specified criteria will be included in the report.

For example:

```
-fA iRw
```

Will only include variables that are not a part of an Input Scan, that have been Read from, but have not been Written to – which finds variables have potentially not been initialized.

This option also allows multiple groups of Access Criteria to be specified. Within each group only variables matching all of the criteria of that group will be included, but the report will contain the inclusions from all the specified groups. Multiple groups are specified by concatenating each group with the "`/`" character.

For example:

```
-fA Ir/0w
```

Defines two Access Criteria groups: "`Ir`" and "`0w`". The `Ir` group will include variables that have been defined as an Input scan, but have not been Read from. The `0w` group will include variables that have been defined as an Output scan, but have not been Written to. Combining the results of the two groups will include all of the Inputs and Outputs of a project that have not been utilized.

When this option is not specified, variables of all Access Criteria types will be included in the report data.

The `-fA` argument is optional.

The `-fA` argument requires better than a Trial level license in order to be used.

[-fM <MemoryCriteria>] Filter report by each variables Memory Criteria

When this option is specified, the report data will only include variables of the given Memory Criteria. All other variables will be excluded from the report.

The Memory Criteria is specified as:

- `I` Input Variables: `%I` and `%AI`
- `O` Output Variables: `%Q` and `%AQ`
- `M` Memory Variables: `%L`, `%M`, `%P`, `%R`, `%T`, `%W` and `<Symbolic>`
- `C` Chained/Aliased to another variable
- `G` Global Variables: `%G`, `%GA`, `%GB`, `%GC`, `%GD` and `%GE`
- `S` System Variables: `%S`, `%SA`, `%SB`, `%SC`

Multiple Memory Criteria can be specified, in which case the variables from all matching types will be included.

For example:

```
-fM IO
```

Will only include variables that are defined as Inputs or Outputs.

When this option is not specified, variables of all Memory Criteria types will be included in the report data.

The `-fM` argument is optional.

[-fRN <NameRegex>] Filter report by Name Regex

When this option is specified, the report data will only include variables whose names match the given Regular Expression (Regex).

For example:

```
-fRN ^Unit1.*
```

Will match all variables whose names start with “Unit1”.

Note that internally `Mexml` specifies all Regex patterns as being case insensitive.

When this option is not specified, variables of all Names will be included in the report data.

The `-fRN` argument is optional.

The `-fRN` argument requires better than a Trial level license in order to be used.

[–fRA <AddressRegex>] Filter report by PLC Address Regex

When this option is specified, the report data will only include variables whose PLC address matches the given Regular Expression (Regex).

For example:

```
–fRA ^%Q0*3\d{2}$
```

Will match all %Q variables whose address is in the range 300 to 399.

Note that internally `Mexml` specifies all Regex patterns as being case insensitive. In addition using this option automatically excludes all symbolic variables – EG variables that do not have a PLC address.

When this option is not specified, variables of all Address values will be included in the report data.

The `–fRA` argument is optional.

The `–fRA` argument requires better than a Trial level license in order to be used.

[-fRD <DescriptionRegex>] Filter report by Description Regex

When this option is specified, the report data will only include variables whose Description matches the given Regular Expression (Regex).

For example:

```
-fRD “open valve”
```

Will match all variables that have the text “open valve” anywhere in their Description.

Note that internally `Mexml` specifies all Regex patterns as being case insensitive.

When this option is not specified, variables of all Descriptions will be included in the report data.

The `-fRD` argument is optional.

The `-fRD` argument requires better than a Trial level license in order to be used.

[-fT <TypeCriteria>] Filter report by each variable's Type Criteria

When this option is specified, the report data will only include variables of the given Type Criteria. All other variables will be excluded from the report.

The Type Criteria is specified as:

- `I` Function Block Input Variables
- `O` Function Block Output Variables
- `X` Function Block Input/Output Variables
- `M` Function Block (private) Member Variables
- `G` Program Global Variables
- `L` Program Block Local Variables,
- `U` Unknown Variable Types

Multiple Type Criteria can be specified, in which case the variables from all matching types will be included.

For example:

```
-fT GL
```

Will only include variables that are Program Global or Program Block Local. All others will be excluded from the report.

When this option is not specified, variables of all Type Criteria types will be included in the report data.

The `-fT` argument is optional.

[-rS <SortCriteria>] Sort report by specified order

When this option is specified, the report data is sorted by the given order. The report data can be sorted on any of three aspects: Name, Address and Data Type, all of which can be specified either as Ascending (a to z, or low to high) or Descending order (z to a, or high to low).

Each aspect and its order are specified as:

- `n` Name, Descending order
- `N` Name, Ascending order
- `d` Data Type, Descending order
- `D` Data Type, Ascending order
- `a` Address, Descending order
- `A` Address, Ascending order

The `<SortCriteria>` can be specified as combinations of each of these aspects, with the report data being sorted in order by each clause.

For example:

```
-rS DaN
```

Will result in the report data being sorted in order by

1. Data Type, Ascending
2. Then within each Data Type, by Address, Descending
3. Then within in each Address, by Name Ascending

When this option is not specified, the report data is sorted by Name Ascending order.

The `-rS` argument is optional.

The `-rS` argument requires better than a Trial level license in order to be used.

[-rCS <ColumnSelect>] Select columns included in the report

When this option is specified, only the specified report columns will be included in the Variable report data.

Each column is identified as:

- `N` Name
- `D` Data Type
- `A` Address
- `V` Variable Definition
- `C` Access Criteria
- `M` Memory Group
- `E` Description

The `<ColumnSelect>` can be specified as combinations of each of these identities, with the report data including only the matched columns.

For example:

```
-rCS NDA
```

Will result in the report data only including the Name, Data Type and Address columns.

When this option is not specified, the columns included in the Variable report default to `NDACE`.

Note that the order of the columns is always fixed as `NDAVCME`.

The `-rCS` argument is optional.

[-rCW <ColumnWidth>] Select minimum width of report columns

When this option is specified, the width of the identified columns is set to be at least the specified number of characters.

Each column for which the width can be specified is identified as:

N	Name
D	Data Type
A	Address
E	Description

The width of the column is specified by appending an integer value to the column identifier. For example `D34` requests that the Data Type column should be at least 34 characters wide. The widths of multiple columns can be specified by concatenating each column definition with a colon. Thus `D34:E100` requests the widths of the Data Type and Description columns.

For example:

```
-rCW N50:D32:A24:E100
```

Requests that the width of the Name column is at least 50 characters wide, the Data Type column at least 32, the Address 24 and the Description 100.

Note:

1. The `-rCW` option only applies to text formatted reports, and is irrelevant for CSV formatted reports.
2. The other possible columns in a report all have fixed widths, thus there is no need to be able to set their widths.
3. This option only requests the minimum width of a particular column. If any data within a column is larger than this minimum, then the actual column width will be expanded to include the widest data width.
4. The width of a report column cannot be smaller than the title for that column.

When this option is not specified (or a particular column width is not defined), the columns included in the Variable report default to the width of the widest data in that column.

The `-rCW` argument is optional.

[-rC] Generate reports in CSV format

When this option is specified, the report data is generated in a CSV format.

When this option is not specified, the report data is generated in a pure text format.

The `-rC` argument is optional.

The `-rC` argument requires better than a Trial level license in order to be used.

[-rE] Explain the meanings of columns in the report data

When this option is specified, the meanings of several columns in the report will be explicitly described. These columns include:

Variable Definition	Describes where `Mexml` found the definition and details about the variable itself, such as from an Interface definition or Hardware configuration.
Access Criteria	Describes how the variable has been accessed, such as if it has been read or written.
Memory Group	Describes how and where in the PLC memory sections the variable is defined, such as if it is in Global or Local memory, or is a Bit or Register based variable.

The Variable Definition comprises 8 different columns of data, with three columns having multiple values while the others are simple binary flags. The columns are numbered 1 through 8 and have the definition of:

1. Source of the Variables definition. This describes where in the analysis process that `Mexml` first encountered a particular variable. This column can take on the following values:

`D`	When processing the Derived variable section of a program block.
`E`	When processing the EGD Configuration
`H`	When processing the Hardware Configuration
`I`	When processing the Interface section of a program block
`L`	When processing instructions in a Ladder Logic program block.
`S`	When processing instructions a Structured Text program block.
`–`	Unknown/undefined definition source.

2. Type of variable. This describes the class of variable, such as being a Global or Local variable. This column can take on the following values:

`G`	Global to the entire program
`L`	Local to a particular program block.
`I`	Input parameter to a Function Block.
`X`	Input/Output parameter of a Function Block.
`M`	Member of a Function Block.
`O`	Output from a Function Block.
`–`	Unknown/undefined definition class.

3. Generation of Variable. This column indicates how the variable was encountered/generated. This column can take on the following values:
    - A — Automatically generated by `Mexml` when it encountered a reference to a PLC address that did not have a variable already defined at that location.
    - E — Explicitly generated for a source such as a program block Interface.
    - – — Unknown/undefined generation.
4. Definition is Valid. The internal `Mexml` quality status of a variables definition. This column can take on the following values:
    - V — The definition is Valid.
    - – — The definition is not valid.
5. Array definition. Indication that the variable is an array definition. This column can take on the following values:
    - A — The variable is an Array definition.
    - – — The variable is not an Array definition.
6. Array Element definition. Indicates that the variable is an element of an array. This column can take on the following values:
    - E — The variable is an Array Element definition.
    - – — The variable is not an Array Element definition.
7. PLC Address. Indicates that the variable has a defined PLC address. This column can take on the following values:
    - P — The variable has a PLC address.
    - – — The variable does not have a PLC address.
8. Forced Access Criteria. Indicates that the Access Criteria of the variable has been forced. This column can take on the following values:
    - F — The Access criteria of the variable has been forced.
    - – — The Access criteria of the variable has not been forced.

An example of the Variable Definition columns is:

```
Definit.
12345678
--------
IGEV----
IGEV----
IGEV--P-
IGEV--P-
ILEV----
IGEV--P-
IGEV--P-
```

The Access Criteria comprises 7 different columns of data, each being simple binary flags. The columns are numbered 1 through 7 and have the definition of:

1. PLC Input Scan. Indicates that the variable is a part of an Input scan, for example being a %I or being a part of a consumed EGD exchange. This column can take on the following values:
   - P The variable is a part of an Input scan.
   - – The variable is a not part of an Input scan.
2. Read. Indicates that that the variable has been explicitly read. This column can take on the following values:
   - R The variable has been explicitly read.
   - – The variable has not been explicitly read.
3. Written. Indicates that the variable has been explicitly written. This column can take on the following values:
   - W The variable has been explicitly written.
   - – The variable has not been explicitly read.
4. Computed Read. Indicates that the variable may have been read by a computed read. This is a read of an array element where the index actually being used is computed at execution time. This column can take on the following values:
   - S The variable has been read via a computed read.
   - – The variable has not been read via a computed read.
5. Computed Write. Indicates that the variable may have been written by a computed read. This is a write of an array element where the index actually being used is computed at execution time. This column can

take on the following values:

X The variable has been written via a computed write.

– The variable has not been written via a computed write.

6. Block access. Indicates that the variable was accessed as a part of a Function Block definition. This can include being read or written.Computed Read. Indicates that the variable may have been read by a computed read. This is a read of an array element where the index actually being used is computed at execution time. This column can take on the following values:

   B The variable has been accessed via a block access.

   – The variable has not been accessed via a block access.

7. PLC Output Scan. Indicates that the variable is a part of an Output scan, for example being a %Q or being a part of a produced EGD exchange. This column can take on the following values:

   Q The variable is a part of an Output scan.

   – The variable is not a part of an Output scan.

An example of the Access Criteria columns are:

```
Access
1234567
-------
--W---Q
IR-----
-RW----
IR----Q
-RW---Q
--W---Q
-RW----
```

The Memory Group comprises 4 different columns of data, all of which have multiple values. The columns are numbered 1 through 4 and have the definition of:

1. Memory Group. Indicates the Group that the variable is defined in, such as %I being an input and %G being a global. This column can take on the following values:
    - – Does not have a Memory Group, for example it is a symbolic variable
    - I Input Memory Group, as it has an %I or %AI address.
    - O Output Memory Group, as it has a %Q or %AQ address.
    - G Global Memory Group, as it has a %G, %GA, %GB, %GC, %GD or %GE address
    - M Memory Memory Group, as it has a %L, %M, %P, %R, %T or %W address.
    - C Chained Memory Group, as its address is the name of another variable.
    - S System memory Group, as it has a %S, %SA, %SB or %SC address.
2. Memory Address Type. Indicates the Type of address that a variable has. This column can take on the following values:
    - – Does not have a Memory Address Type.
    - N Numerical Memory Address Group, as it has and address such as %M01234 or %Q0567.
    - S Symbolic Memory Address Group, as it does not have any address.
    - I Indirect Memory Group, as its address is the name of another variable.
3. Memory Size. Indicates the numerical size of a variable. This column can take on the following values:
    - – Does not have a Memory Size, as it is an indirect variable, or a # variable.
    - B Bit Memory Size, as it has a %I, %Q, %G, %GA, %GB, %GC, %GD, %GE, %M, %T, %SA, %SB, or %SC address.
    - R Register memory Size, as it has a %AI, %AQ, %L, %P, %R or %W address.

4. Parent Memory Group. Indicates the Memory Group of the parent of chained variable. This column can take on the following values:

   - `-` Does not have a Parent Memory Group, for example it is not a chained variable
   - `I` Input Memory Group, as its parent has an `%I` or `%AI` address.
   - `O` Output Memory Group, as its parent has a `%Q` or `%AQ` address.
   - `G` Global Memory Group, as its parent has a `%G`, `%GA`, `%GB`, `%GC`, `%GD` or `%GE` address
   - `M` Memory Memory Group, as its parent has a `%L`, `%M`, `%P`, `%R`, `%T` or `%W` address.
   - `C` Chained Memory Group, as its parents address is the name of another variable.
   - `S` System memory Group, as its parent has a `%S`, `%SA`, `%SB` or `%SC` address.

An example of the Memory Group columns is:

```
Mem.
1234
----
MS--
MS--
MNR-
MNR-
MS--
MNR-
MNR-
```

When the `-rE` option is not specified, the column explanation is not added to the Variable Access report.

The `-rE` argument is optional.

[-rF <ReportFile>] Variable Access report file

When this option is specified, the output of the report will be saved to this filename, overwriting any previously existing file of the same name. The file can be specified using any valid Windows path and filename combination.

For example:

```
-rF c:\data\reports\variableReport.txt

-rF ..\..\reports\variables.csv
```

If the "-rD <ReportDir>" directory is also specified, it will be prepended to the "<ReportFile>" name in order to create the complete report filename. (In which case the report file can only be specified using a relative path.) An example of these options and the final report filename is:

```
-rD c:\data -rF this_report\variableReport.txt

c:\data\this_report\variableReport.txt
```

The benefits of combining the -rD and -rF options come into play when used in conjunction with the "-rP <ReportParaFile>" option. This enables multiple reports to be generated from the same source files, with the data from each report being written to its own specific file.

When this option is not specified, then any generated report data will be sent to the console.

The -rF argument is optional.

Other Arguments

These arguments relate to `Mexml` itself.

[-pv] Display the program version

Displays the current version of `Mexml` that is installed.

```
C:\>mexml -pv
Mexml version 2.1.0.7
```

The –v argument is optional.

[-lS] Display the chosen license

Displays details of the license currently being used by `Mexml`. If there is more than one license available to `Mexml`, the program will choose what it considers to be the least restrictive license and use that license.

```
C:\>mexml -lS
File: C:\Program Files (x86)\JoalahDesigns\Mexml\Mexml\↩
     Mexml_License_Standard_FredNurk_Never.xml
ID = 974262f9-8de5-493e-a986-6c275c37ef7f, Type = Standard, ↩
Status = InCompliance
Expiration: Never Expires
Application: Mexml
Registered to:
  Name:    Fred Nurk
 Company: Joalah Designs LLC
  Email:   Fred@JoalahDesigns.com
```

The `-lS` argument is optional.

[-lD] Display all licenses

Displays the details of all the licenses that are available to `Mexml`, as well as the license that has been selected.

```
C:\>mexml -lD
Looking for licenses in:     C:\Program Files (x86)\↩
                              JoalahDesigns\Mexml\Mexml
For licenses that match:     Mexml*.xml
With public keys that match: *PublicKey.xml

Discovered 2 potential out of 2 total licenses

All Licenses
------------
    File: C:\Program Files (x86)\JoalahDesigns\Mexml\Mexml\↩
          Mexml_License_Standard_FredNurk_Never.xml
    ID = 974262f9-8de5-493e-a986-6c275c37ef7f,↩
          Type = Standard, Status = InCompliance
    Expiration: Never Expires
    Application: Mexml
    Registered to:
      Name:    Fred Nurk
      Company: Joalah Designs LLC
      Email:   Fred@JoalahDesigns.com

    File: C:\Program Files (x86)\JoalahDesigns\Mexml\Mexml\↩
          Mexml_License_Trial_JoalahDesigns_Never.xml
    ID = f3b96be6-e715-4fdf-92f7-f36e2de16855,↩
          Type = Trial, Status = InCompliance
    Expiration: Never Expires
    Application: Mexml
    Registered to:
      Name:    Joalah Designs
      Email:   Info@JoalahDesigns.com

Selected License
----------------
    File: C:\Program Files (x86)\JoalahDesigns\Mexml\Mexml\↩
          Mexml_License_Standard_FredNurk_Never.xml
    ID = 974262f9-8de5-493e-a986-6c275c37ef7f,↩
          Type = Standard, Status = InCompliance
    Expiration: Never Expires
    Application: Mexml
    Registered to:
      Name:    Fred Nurk
      Company: Joalah Designs LLC
      Email:   Fred@JoalahDesigns.com
```

The `-lD` argument is optional.

[-?] Display the program help

Displays the full help data as shown in the previous sections, along with a brief description of the program, licensing information and where you can get help online or via email.

```
C:\>mexml -?
Mexml - Process GE Machine Edition XML files and report on
        usage of variables and program blocks

Copyright (c) Joalah Designs LLC 2014
For support and to report bugs,
send emails to info@JoalahDesigns.com
See www.JoalahDesigns.com for more information

Licensed Level: Standard

Usage: Mexml [-X <ProgramFile(s)>] [-E <EGDFile>]
             [-H <HardwareFile>] [-P <PreloadFile(s)>]
             [-V <ForceFile(s)>] [-D <DefinitionFile(s)>]
             [-kAN] [-kFB] [-aBD] [-mV] [-mNW] [-mQ]
             [-xRN <NameRegex>] [-xLB] [-xLI] [-xLD] [-xLA]
             [-xLRS] [-xSB] [-xSI] [-xSD] [-xSA] [-xFO] [-eM]
             [-eDS] [-hM] [-hDS] [-pM] [-vM] [-dM] [-kCP] [-rM]
             [-O <OutputFile>] [-sCO] [-sMA] [-rV]
             [-rD <ReportDir>] [-rP <ReportParamFile>]
             [-fA <AccessCriteria>] [-fM <MemoryCriteria>]
             [-fRN <NameRegex>] [-fRA <AddressRegex>]
             [-fRD <DexcriptionRegex>] [-fT <TypeCriteria>]
             [-rS <SortCriteria>] [-rC] [-rCS <ColumnSelect>]
             [-rCW <ColumnWidth] [-rF <ReportFile>] [-rE] [-pv]
             [-lS] [-lD] [-?]

Where:
  Source File Arguments

      [-X <ProgramFile(s)>]     Program Files (wildcards
                                  allowed in filename,
                                  all matches will be used)

      ...
```

The -? argument is optional.