

Configuring Math and Logic

Math and Logic programs are user designed solutions providing control of the interactions of the FactoryLink tasks and system activity.

A Math and Logic program is comprised of one or more procedures which can contain variables and triggers. These variables and triggers are defined within the Math and Logic editor or in the FactoryLink tables prior to writing the procedures. Variables are tags which can be global to FactoryLink use or local to the program that contains the procedure. Triggers define the method to invoke a one or more procedures.

Configuration of procedures includes definition as a Compiled Mode Logic (CML) or Interpreted Mode Logic (IML) procedure. FactoryLink loads the IML program files (the user defined .prg file) into memory at startup. Each time the IML procedure (proc) is triggered, the task interprets the procedure's instructions and then performs the actions required. With CML, the original .PRG file remains unaltered. At startup the CML program files are compiled into executable C source code files and linked with FactoryLink and externally supplied libraries. When CML procedures are triggered, the CML executable files are invoked. The FactoryLink designer must determine the type of mode to use.

In addition to defining procedures there are two tasks, the IML and CML tasks, associated with Math and Logic that must be operational to support the respective modes.

Table 1-1 Comparison of IML and CML Modes

IML	CML
Interpreted Mode	Compiled mode
No compiler required	External compiler application required
Cannot access any C functions	C, C++ functions can be accessed
Less efficient	More efficient and faster processing
Debug switches can be used	No debug switches
Validate procedures before running program file	Cannot validate entire procedure specifically the data between cbegin and cend

A new program, when saved, is automatically stored at: c:\{FLAPP}\SHARED\procs. It is essential that the file remains in this directory.

- **CONFIGURING MATH AND LOGIC**
-
-
-

The number of programs and their respective components is limited only by the amount of available memory and the operating system. There are additional compiler limitations when using Compiled Mode Logic (CML). Using CML is optional and requires a compiler program in addition to the FactoryLink software. There are no additional compiler requirements for using Interpreted Mode Logic (IML).

The Microsoft® Visual C++ compiler is the recommended compiler to use with the FactoryLink CML processing. Refer to the documentation supplied with the compiler for details on the compiler limitations for your system. Refer to the FactoryLink installation instructions for the compatible version.

This chapter describes how to use the Configuration Explorer program to configure the Math and Logic tables, variables, triggers and tasks plus the Math and Logic editor functions. The requirements to accommodate CML procedures are also detailed. A basic comprehension of how to use the Configuration Explorer program is necessary to use the instructions in this chapter.

Note: All configuration for FactoryLink 7.0 is in the SHARED domain. USER domain indicators are provided for upgrades from older FactoryLink versions.

There are ten sections to describe Math and Logic functionality:

- Locating Math and Logic Functions in Configuration Explorer
- Configure Math and Logic Tasks
- Configure Math and Logic Tables
- Program Files and Procedures
- Modify Makefiles
- Compiled Math and Logic
- CML Utilities Call Sequence
- CML Variables
- Calling C Code
- The Math and Logic Editor Reference Pages

For additional information about other Math and Logic topics, see the FactoryLink 6.6 documentation as indicated below:

Operational concepts

**FactoryLink Fundamentals Link
to Chapter 12**

Syntax, tokens, naming conventions, structure, declarations, data types and data conversions, expressions, operators, precedence, statements and directives

**FactoryLink Configuration
Reference Link to Chapter 7**

Procedures, functions, arguments

**FactoryLink Configuration
Reference Link to Chapter 8**

Makefiles, CML process and CML executables

**FactoryLink Configuration
Reference Link to Chapter 9**

Error code list

**FactoryLink Reference Guide
Link to Chapter 11**

- **CONFIGURING MATH AND LOGIC**
- *Locating Math and Logic Functions in Configuration Explorer*
-
-

LOCATING MATH AND LOGIC FUNCTIONS IN CONFIGURATION EXPLORER

The FactoryLink Server and the FactoryLink Application directory (FLAPP) must be defined before configuring the Math and Logic variables, triggers, procedures, and programs. Ensure that these functions are completed before proceeding with the following instructions. Your FactoryLink configuration may differ from the configuration used to generate the figure examples provided in this section.

The FactoryLink server and the FactoryLink application are set up using the Configuration Explorer program. For a complete description on how to use the Configuration Explorer program and all of its features, see the [Configuration Explorer manual](#).

Note: It is not advisable to have FactoryLink in the run-time mode to configure Math and Logic tasks, tables and programs.

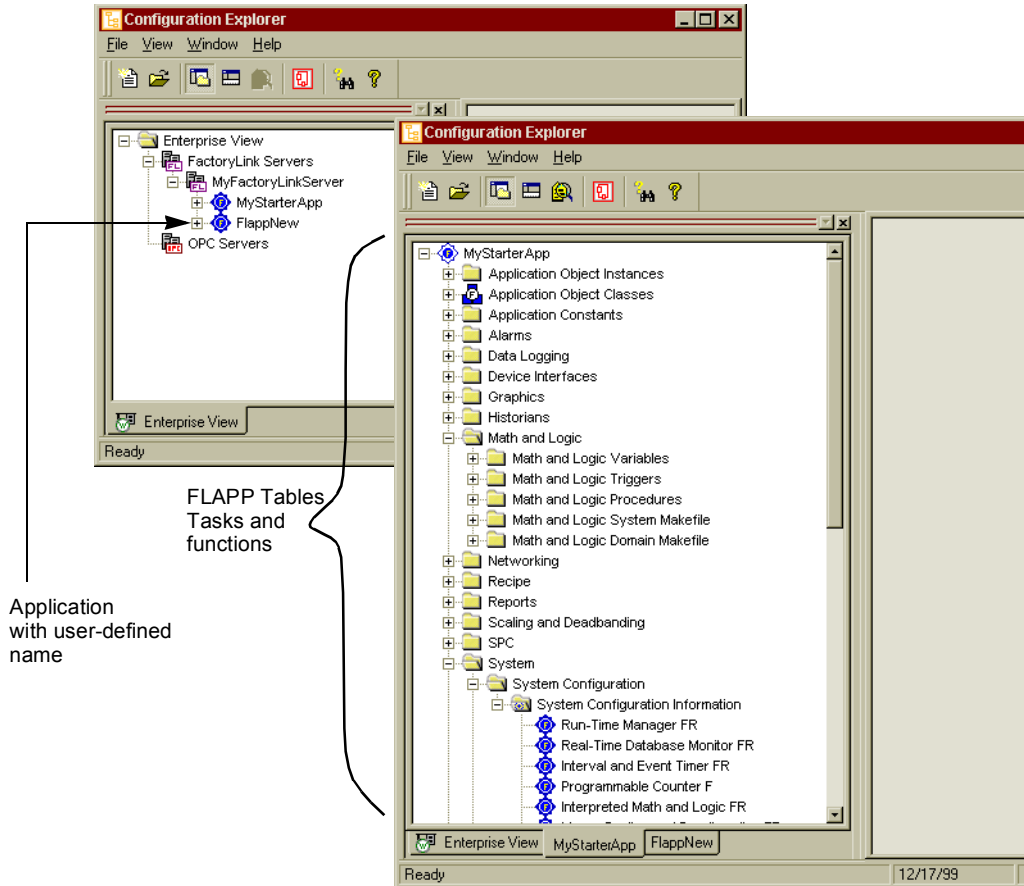
This section explains how to use the Configuration Explorer program to locate the Math and Logic functions:

- Launch Configuration Explorer
- Locate Math and Logic Tasks, Tables and Editor

Launch Configuration Explorer

To launch Configuration Explorer: click the Configuration Explorer icon displayed on the desktop. The Configuration Explorer window appears with the workspace list expanded (see [Figure 1-1](#)).

Figure 1-1 The Workspace and the Application Tree



Click FactoryLink Servers to display one or more servers. Expand the server you will use to configure Math and Logic programs. One or more of the applications available on this server appear. Right-click the application you will use to configure Math and Logic programs. From the menu that appears, select Open in new Tab. The expanded application appears in a new workspace window.

- **CONFIGURING MATH AND LOGIC**
- *Locating Math and Logic Functions in Configuration Explorer*
-
-

The Math and Logic item in the application tree provides access to configure all properties and requirements of Math and Logic programs.

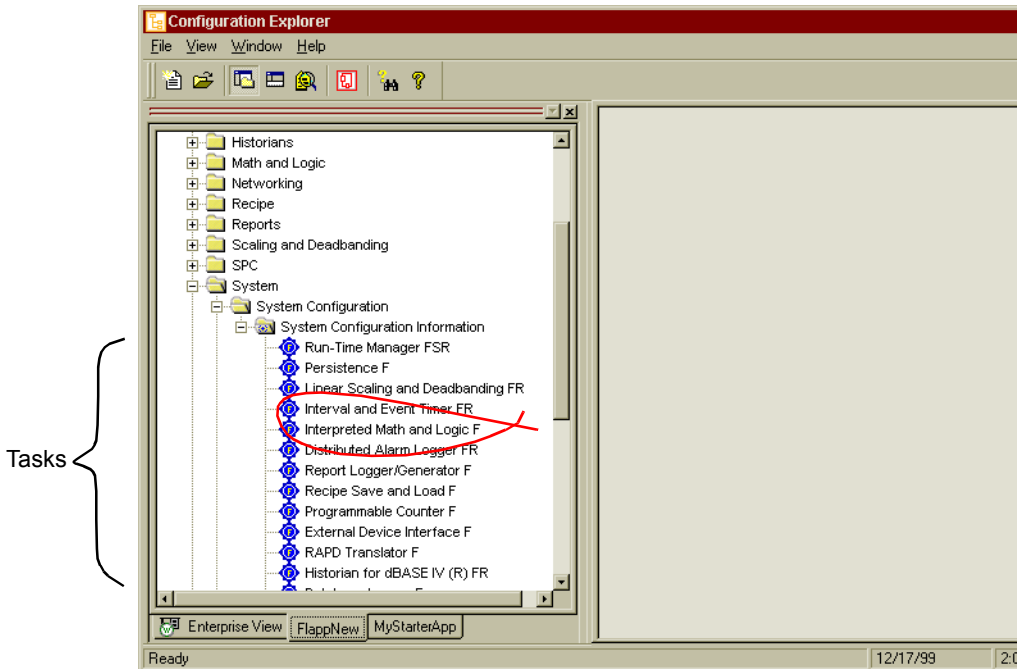
Locate Math and Logic Tasks, Tables and Editor

Math and Logic Tasks

There is one task preconfigured in FactoryLink to accommodate IML processing. If your installation is using CML processing, the CML task must be added.

To view the Math and Logic tasks, expand the System item in the application tree. Then expand the System Configuration and the System Configuration Information items respectively. Double-click the Interpreted Math and Logic item. The System Configuration Information dialog box displays. See [Figure 1-2](#) for an example of the expanded application tree.

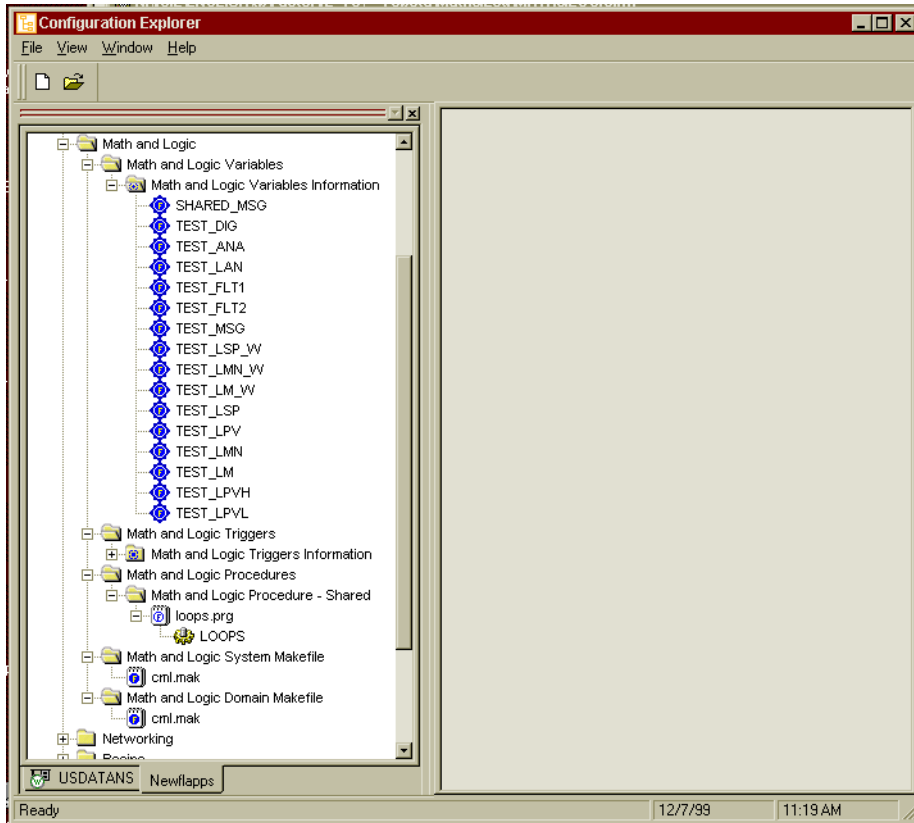
Figure 1-2 Locate Tasks in Configuration Explorer



Math and Logic Tables and Editor

To view the Math and Logic tables and text editor functions, expand the Math and Logic item in the application tree. [Figure 1-3](#) shows an example of a Math and Logic tree with predefined variables, triggers and editor files.

Figure 1-3 Expanded Math and Logic Tree Items



Five items are listed under Math and Logic in the application tree:

- Math and Logic Variables Information
- Math and Logic Triggers
- Math and Logic Procedures
- Math and Logic System Makefile
- Math and Logic Domain Makefile

- **CONFIGURING MATH AND LOGIC**

- *Locating Math and Logic Functions in Configuration Explorer*
-
-

The first two items contain configuration tables and the remaining three items launch the Math and Logic editor to enable creation or modification of executable files.

The default Configuration Explorer settings only display the Share domain options. To view both the user and share domain items, right-click the FLAPP name and select **Shared+User** from the menu.

Names supporting the Math and Logic functions, for example: constants, variables (tags), procedures and files, must be unique. They are also case sensitive. Not all special characters are supported. For CML procedures the unique naming is also limited by the effects of the compiled C code. For example, special characters (\$.*) become _ by the parsing routine. Therefore, the declarations:

```
declare short lu$lu
```

```
declare short lu@lu
```

```
declare short lu_lu
```

all become declare short lu_lu with potentially confusing result such as duplicate definition errors or changes in one variable are reflected in another.

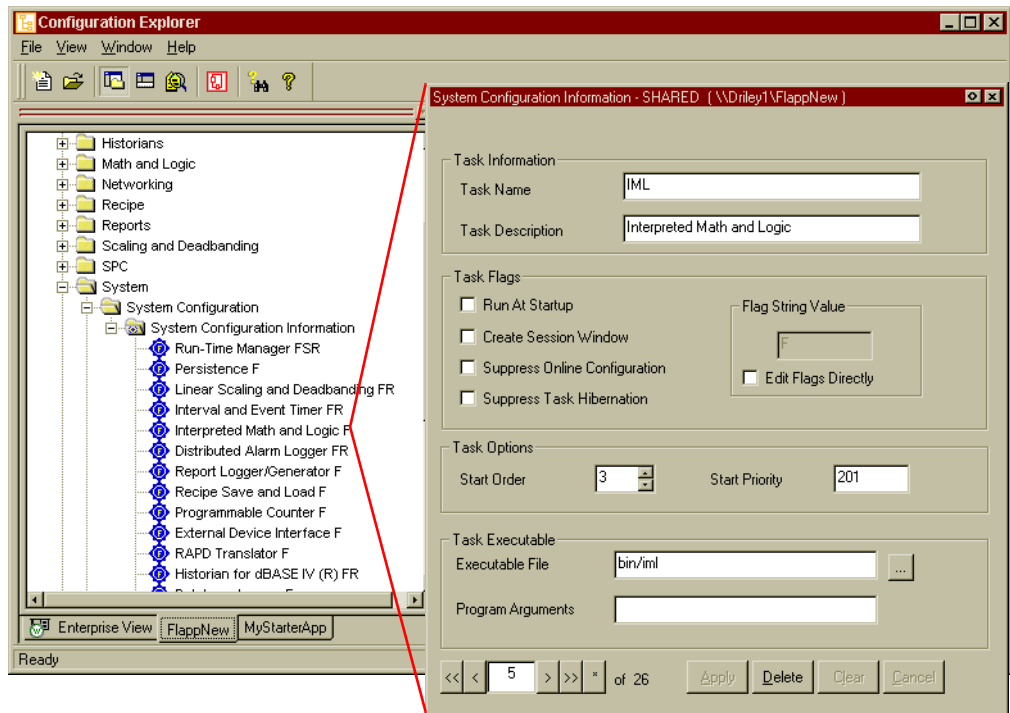
The variables, triggers, and procedures can be configured and managed from the Math and Logic editor as the program is developed. However, the variables and triggers can be defined before developing the procedures. The following section describes setting up tasks, configuring the tables and using the editor. The view illustrated for each section , either form view or grid view, is the default view for the item.

CONFIGURE MATH AND LOGIC TASKS

The preconfigured IML task is located in the System Configuration Information dialog box.

To view the Interpreted Math and Logic task dialog box, double-click the task in the System Configuration Information list (see [Figure 1-4](#)). The dialog box displays.

Figure 1-4 IML System Configuration Task Dialog Box



To enable CML processing, the CML task must be added to the task item list. The IML task does not need to be removed. Both tasks can be enabled and run at the same time.

The CML task can be added anywhere in the task list. Adding a task requires displaying an existing task to use the dialog box as a template. The new task is added to the list below the task displayed to create it. The position of the task in the list does not determine its rank in the run-time process. Run-time rank is determined by the Start Order field.

To add a task: double-click an existing task in the list, for example, the Interpreted Math and Logic task, to display the System Configuration Task dialog box. Click the Asterisk button in the lower left portion of the dialog box. Complete the fields using the information from [Table](#)

- **CONFIGURING MATH AND LOGIC**

- *Configure Math and Logic Tasks*

-
-

1-2. Click the Apply button and the new task is completed. Refresh the application tree to display the new task in the list.

Table 1-2 shows the optional and required entries to run both the IML and the CML tasks.

Table 1-2 IML and CML Task Dialog Box Fields (Sheet 1 of 2)

Field Name	Definition		Default Field Data	
			IML	CML
Domain	For new FactoryLink 7.0 users, with no application conversion requirements from previous FactoryLink versions, shared is the recommended option. See the FactoryLink 6.6 system documentation for legacy USER domain requirements.		Shared	Shared
Task Information	Task Name	Predefined name for the task which cannot be changed.	IML	CML
	Task Description	Optional alphanumeric description.	Interpreted Math and Logic	Compiled Math and Logic
Task Flags	Run at Startup	R: Invokes task at FactoryLink startup. This must be set to use IML or CLM.	No	No
	Create Session Window	S: Provides the process with its own tab window. Output prints to the Configuration Explorer Output window.	Optional	Optional
	Suppress Online Configuration	O: Suppress online updates for this process.	Optional	Optional
	Suppress Task Hibernation	H: Not applicable for Math and Logic functions.	Not applicable	Not applicable
Flag String Value	Input box	Displays value code of selected Task Flags. F: Foreground Flag. Puts this task in the foreground at startup.	Yes	No
	Edit Flags Directly	If selected, allows user input of string values to input box.		

Table 1-2 IML and CML Task Dialog Box Fields (Sheet 2 of 2)

Field Name	Definition		Default Field Data	
			IML	CML
Task Options	Start Order	Specifies the runtime rank for invoking the task when FactoryLink is started. ¹	3	3
	Start Priority	Processing priority.	201 (default)	201 (default)
Task Executable	Executable File	Path and name of the file which executes this task. CML: See the FactoryLink 6.6 system documentation for legacy USER domain requirements.	bin/iml	{FLAPP}/share d/cml/c {FLDO MAIN}.exe NOTE: Above path is typed in the field exactly as it appears.
	Program Arguments	Codes which add customization to the functions of the task.	-L: log error messages to a file -L-V#: log error messages with more information (increased verbose level) to a file	None

1. Start order is coordinated with the function requirements in the procedures. If Math and Logic procedures use the information from another process, that process must start first.

For more information about creating and modifying tasks, see the [Configuration Explorer Manual](#).

- **CONFIGURING MATH AND LOGIC**
- *Configure Math and Logic Tables*
-
-

CONFIGURE MATH AND LOGIC TABLES

There are two types of tables used to configure the variables and triggers in Math and Logic program and procedures:

- Math and Logic Variables
- Math and Logic Triggers

Math and Logic Variables

Variables in Math and Logic are known as tags in other FactoryLink functions. All variables can be defined using the Math and Logic Information table or defined during the editing process while creating procedures in a program file. Any tags defined in this table are globally available to any FactoryLink process. Defined variables automatically appear in the FactoryLink global Tag Browser, the Object Table and the Xref Table. However, when deleted from the Math and Logic Variables Information table, variables are not deleted from the global Tag Browser or the Object Table.

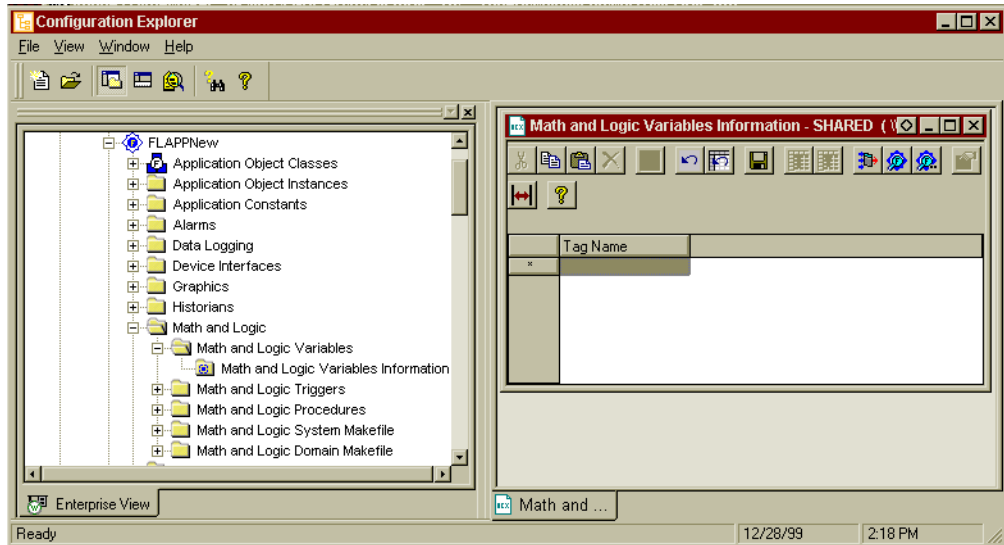
Local variables, specific to a file, can be defined within the program and cannot be used or referenced outside of the program.

The Math and Logic Variables Information Table

The Math and Logic Variables Information table stores the definitions of all of the variables defined for use in the Math and Logic programs associated with the current FLAPP.

Expand the Math and Logic Variables item in the application tree to display the Math and Logic Variables Information item (see [Figure 1-5](#)). Double-click this item and the grid view of the table appears. One field is required for the definition of each variable (tag) name.

Figure 1-5 Math and Logic Variables Information Table



Define variables (tags) using the default grid, the record generation box or application objects. See the [Configuration Explorer Manual](#) for a complete explanation and attributes of each method.

There is one field for each tag definition in the table:


Tag Name Name of the tag to be used in a Math and Logic procedure. Do not use a Math and Logic reserved keyword as an element name. If the tag is to be defined as an array, specify 0 for each array dimension when entering its name here, for example, batch[0][0].

Valid Entry: standard tag name, 1 through 16 characters

Valid Data Type: digital, analog, longana, message, float

Enter one or more tag names to the grid and click the Save button. The Tag dialog box displays for each tag in succession to enable complete definition of each tag. If the tag is defined as an array, specify the number of elements for each array dimension in the Dimension field of the Tag Definition dialog.

Once defined, the tag name appears in the Xref Table, the Tag Browser Table and the Object Table in addition to the Math and Logic Variables table.

Click the Save  button to save the information after configuring this table in the grid or form views.

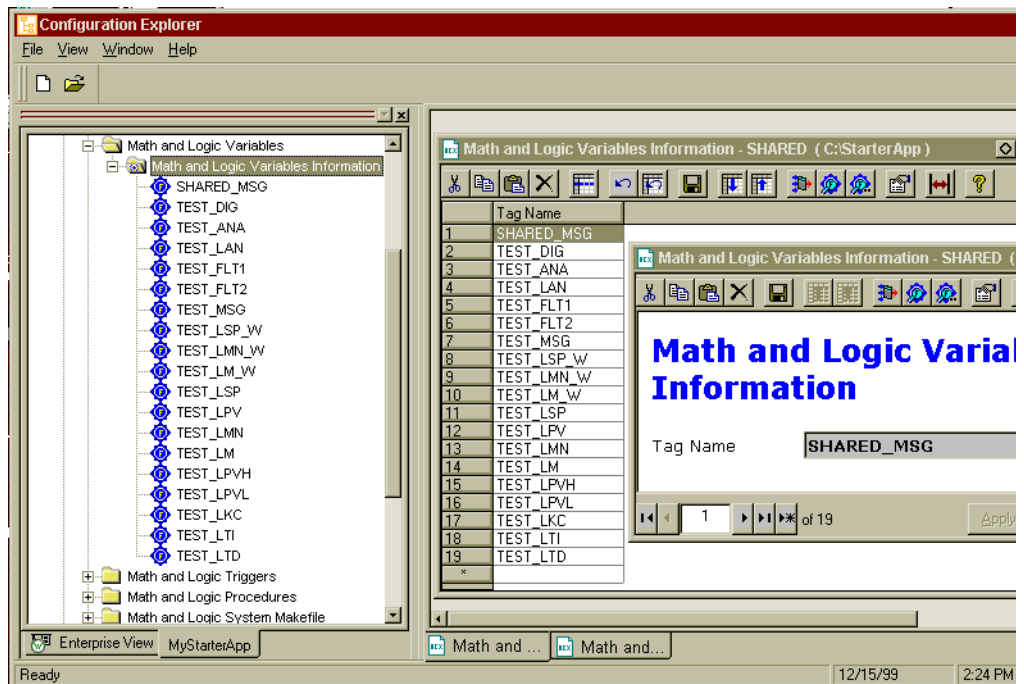
- **CONFIGURING MATH AND LOGIC**
- *Configure Math and Logic Tables*
-
-

The Math and Logic Variables Information Table Example

Once variables are defined, each variable is listed below the Math and Logic Variables Information item in the application tree and in the grid view of the table. See [Figure 1-6](#) for an example showing an application with defined Math and Logic variables.

After a variable has been defined it can be displayed in form view as an individual item: Double-click any of the listed variables in the application tree and the form view of the variable appears.

Figure 1-6 Example of Defined Math and Logic Variables



Math and Logic Triggers

Math and Logic Triggers are used to invoke the Math and Logic procedures in both IML and CML programs. Triggers can be defined before or during the procedure design process.

All procedures must be listed in the Math and Logic Trigger Information table to identify the procedure for FactoryLink. Procedures that do not require triggers are listed with no entry to the Trigger field.

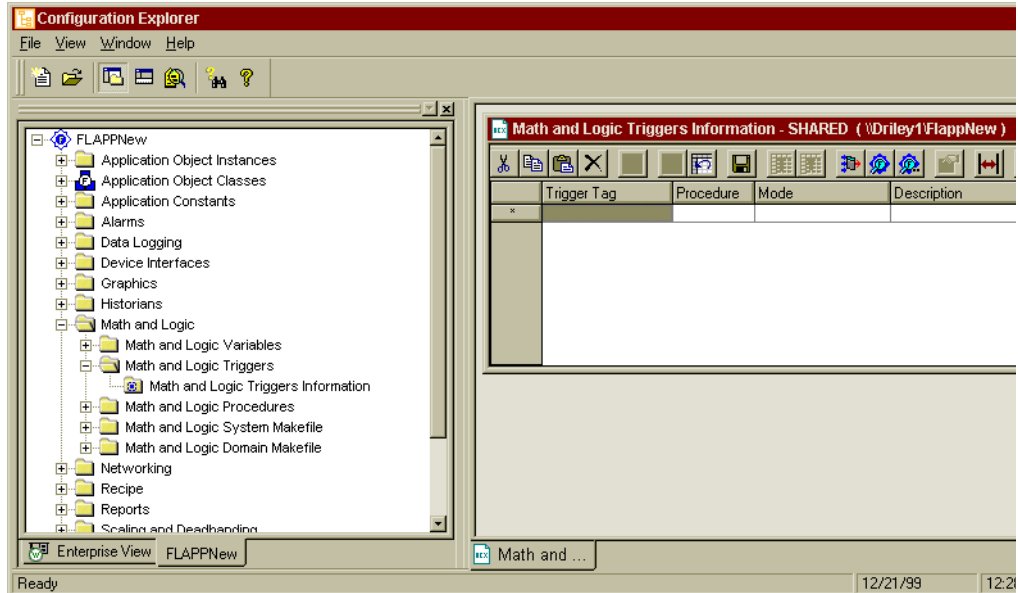
A trigger can be defined and never appear in the body of a procedure. The procedure is invoked when the value changes in the trigger tag. Digital tags are the most common trigger type but any tag type can be specified except the mailbox type. Digital tags only trigger a procedure when the value is true. Other tag types trigger procedures whenever their value changes.

Not all procedures require a trigger. An alternative method is to call a procedure from another procedure. However, a procedure can contain a defined trigger to invoke another procedure. Also a single trigger can be used to invoke multiple procedures. Triggers which appear in procedures must also be defined in the Math and Logic Variables table.

The Math and Logic Triggers Information Table

Expand the Math and Logic Triggers item in the application tree to display the Math and Logic Triggers Information item (see [Figure 1-7](#)). Double-click this item and the grid view of the table appears. Define triggers using this Math and Logic Triggers Information table grid, the record generation box, or application objects. See the [Configuration Explorer Manual](#) for a complete explanation and attributes of each method.

Figure 1-7 Math and Logic Trigger Information Table



- **CONFIGURING MATH AND LOGIC**

- *Configure Math and Logic Tables*
-
-

Four fields are provided for the definition of each trigger:

Trigger Tag Name of the tag whose value can trigger a Math and Logic procedure.

Valid Entry: standard tag name, 1 through 16 characters

Valid Data Type: digital, analog, longana, message

Procedure Unique name of the Math and Logic procedure exactly as entered in the procedures (proc) statement within the program file.

Valid Entry: Alphanumeric string: 1 to 16 characters, case sensitive, cannot be the same as a defined tag name and must begin with an alphabetic character.

Mode Determines how the Math and Logic procedure instructions are executed.

Valid Entry: INTERPRETED
or COMPILED

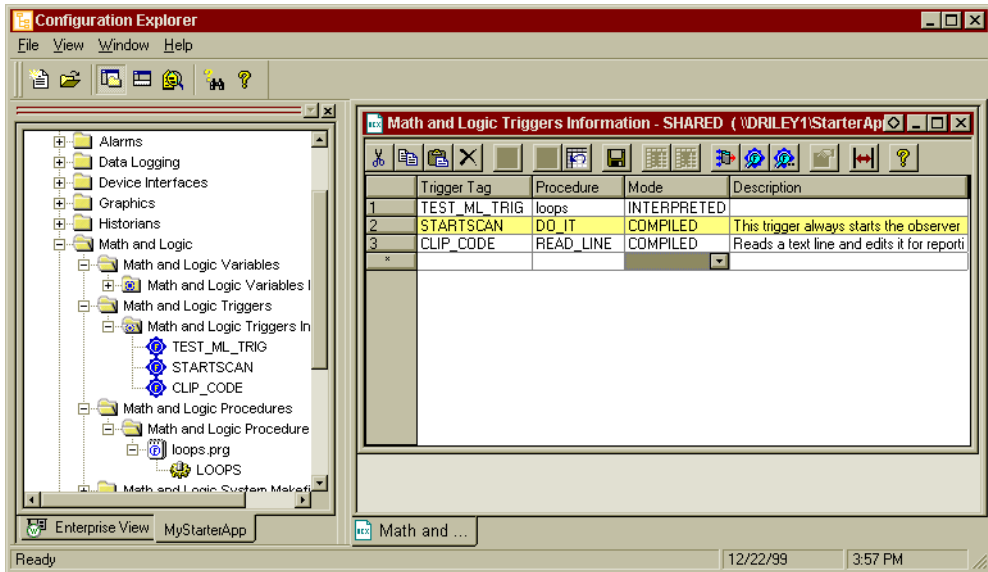
Description Indicates the intended use of the tag specified in the Trigger Tag field.

Valid Entry: Alphanumeric string; 1 to 80 characters


The Math and Logic Triggers Information Table Example

An example of the grid view of a Math and Logic table with defined triggers is provided in [Figure 1-8](#).

Figure 1-8 Math and Logic Triggers Information Table Example



The Trigger Tag is not automatically added to the Math and Logic Variables Information table. It is, however, added to the global Tag browser table after definition in the Math and Logic Triggers Information table.

To save the table entries at any time click the save icon  at the top of the edit window or use the keycode.

- **CONFIGURING MATH AND LOGIC**

- *Program Files and Procedures*

-
-

PROGRAM FILES AND PROCEDURES

Program files consist of one or more procedures. Procedures within a program file can be totally unrelated in functionality as they are individually invoked by the predefined trigger or a function call embedded in another procedure. All procedures within a program must be defined as either IML or CML procedures.

The Math and Logic editor, located within the Configuration Explorer environment, is used to create and modify the program files and associated procedures using the C-type programming language standards. If you use another editor, validate the procedures you write with the Math and Logic editor.

Program files provide the FactoryLink user the ability to control the interactions FactoryLink tasks. The convention for program file naming is the name must be in lower case and have the .prg extension. Program file names are not referenced in any tables.

Note: In FactoryLink versions before the 7.0 release, the program file name matched the eight-character name of a procedure within the program file. This restriction is no longer applicable.

All procedures must be listed in the Math and Logic Triggers Information table. If a procedure is not listed in the Math and Logic Triggers table it will never be invoked regardless of the method used to invoke it. The IML task uses this table to locate and load IML procedures. The CML compiling process at startup also uses this table. Procedure names are case sensitive. Ensure that the naming of the procedures is always consistent to avoid errors.

As global variables (tags) and triggers are defined they are added to FactoryLink reference lists (see [Figure 1-9.](#)) These variables are available to all FactoryLink tasks. Local variables are defined in the procedure, not in any of the tables, are only available to the procedures or program files dependent on the type of declaration.

Figure 1-9 Data Locations for a Global Shared Tag

Tag name defined in the:

- Math and Logic program file
- Math and Logic Variable Information table
- Xref Table
- Tag browser

A Trigger Tag is also found in every location that any other global tag is found.

```

C:\StarterApp\SHARED\procs\loops.prg
# Purpose: Loop simulator
#-----
proc loops()
begin

declare float _lpv, _lsp, _lmn, _err

# loop mode change
if (!TEST_LM_W) then
# setpoint bumpless transfer to auto mode
if (TEST_LM_W = AUTO) then
TEST_LM = 1
end if
end if
end
    
```

Tag Name	Tag Dimension	CT Domain	Task N.
8 TEST_LSP_W			
9 TEST_LMN_W			
10 TEST_LM_W			
11 TEST_LSP			
12 TEST_LPV			
13 TEST_LMN			
14 TEST_LM			
15 TEST_LPVH			
16 TEST_LPVL			
17 TEST_LXC			
18 TEST_LTI			

Trigger Tag	Procedure	Mode	Description
1 TEST_ML_TRIG	loops	INTERPRETED	

Tag Name	Tag Dimension	CT Domain	Task N.
819 TEST_LMN		SHARED	IML
820 TEST_LMN_W		SHARED	IML
821 TEST_LM_W		SHARED	IML
822 TEST_LPV		SHARED	IML
823 TEST_LPVH		SHARED	IML
824 TEST_LPVL		SHARED	IML
825 TEST_LSP		SHARED	IML
826 TEST_LSP_W		SHARED	IML
827 TEST_LTD		SHARED	IML
828 TEST_LTI		SHARED	IML
829 TEST_ML_TRIG		SHARED	IML
830 TEST_ML_TRIG		SHARED	TIMER
831 TEST_MSG		SHARED	IML
832 TIME		SHARED	RUNMGR
833 TIME0		USER	RUNMGR
834 TOPWINDOW_U		USER	DATASRVD

Name	Domain	Description	Type
TEST_LM	SHARED	Loop mode test tag	DIGITAL
TEST_LMN	SHARED	Loop output test tag	FLOAT
TEST_LMN_W	SHARED	Loop output write test tag	FLOAT
TEST_LM_W	SHARED	Loop mode write test tag	DIGITAL
TEST_LPV	SHARED	Loop process variable test tag	FLOAT
TEST_LPVH	SHARED	Loop PV High Alarm test tag	FLOAT
TEST_LPVL	SHARED	Loop PV Low Alarm test tag	FLOAT
TEST_LSP	SHARED	Loop setpoint test tag	FLOAT
TEST_LSP_W	SHARED	Loop setpoint write test tag	FLOAT
TEST_LTD	SHARED	Loop Rate test tag	FLOAT
TEST_LTI	SHARED	Loop Rest test tag	FLOAT
TEST_ML_TRIG	SHARED	Math and logic test trigger	DIGITAL
TEST_MSG	SHARED	Message test tag	MESSAGE
TIME	SHARED	Time (HH:MM:SS)	MESSAGE
TIME0	SHARED	Time (HH:MM:SS)	MESSAGE

Tags	Global	View Only
GLOBAL	U	
DS_WRTAG		0

It is recommended that definition of new programs and procedures be accomplished in an off-line mode (without FactoryLink running).

There are many methods to perform file functions, for example: Save, Redo, Find, and Validate, using the Math and Logic editor. For a complete reference of these options see the section [The Math and Logic Editor Reference Pages](#) at the end of this chapter.

- **CONFIGURING MATH AND LOGIC**

- *Program Files and Procedures*

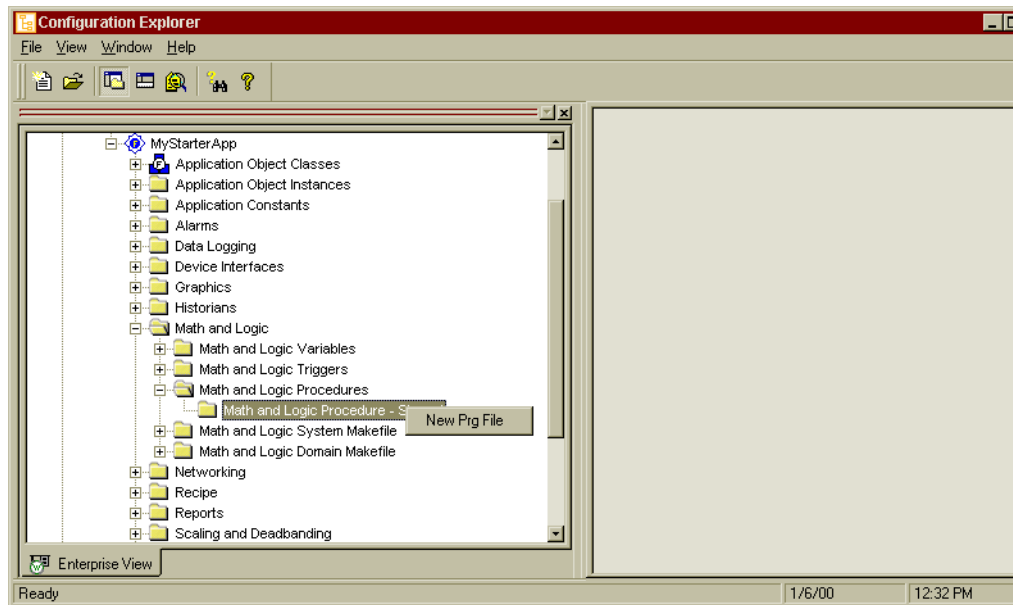
-
-

Defining a New Program File

Program files are defined and created in Configuration Explorer. Locate the FLAPP in the Configuration Explorer Workspace window, expand the FLAPP and expand the Math and Logic item.

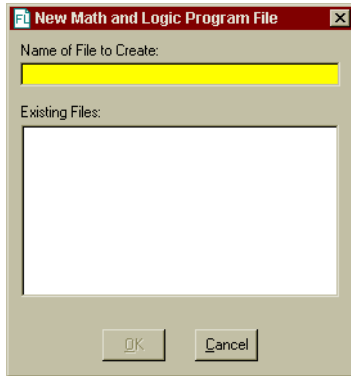
To open a new program file, right-click the Math and Logic Procedure - Shared item. Select the only option: New Prg file. See [Figure 1-10](#).

Figure 1-10 New Program File Selection



A New Math and Logic Program File dialog box appears (see [Figure 1-11](#)).

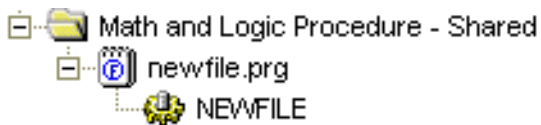
Figure 1-11 New Math and Logic Program File



Type the program file name in the Name of File to Create field. The .prg extension will be added automatically. If there are any previously created files they display in the Existing Files field also. Click the OK button and then click the Yes button of the Confirm Creation of New File dialog box.

In the Configuration Explorer tree, expand the Math and Logic Procedure - Shared item and then expand the program name. A new procedure appears under the program name with the same name minus the extension. The program and the procedures have unique icons (see [Figure 1-12](#)).

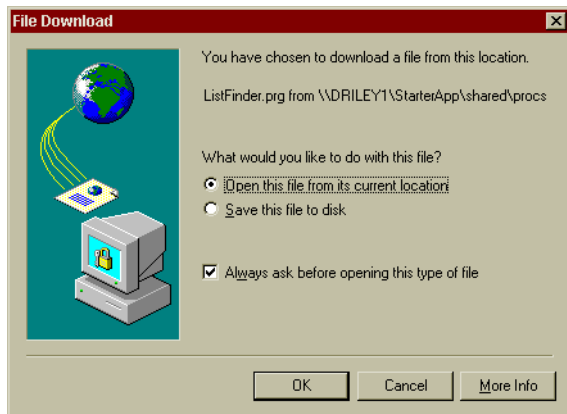
Figure 1-12 New Program and Procedures Items in Configuration Explorer



Double-click the program name to open the program. The File Download dialog box appears (see [Figure 1-13](#)).

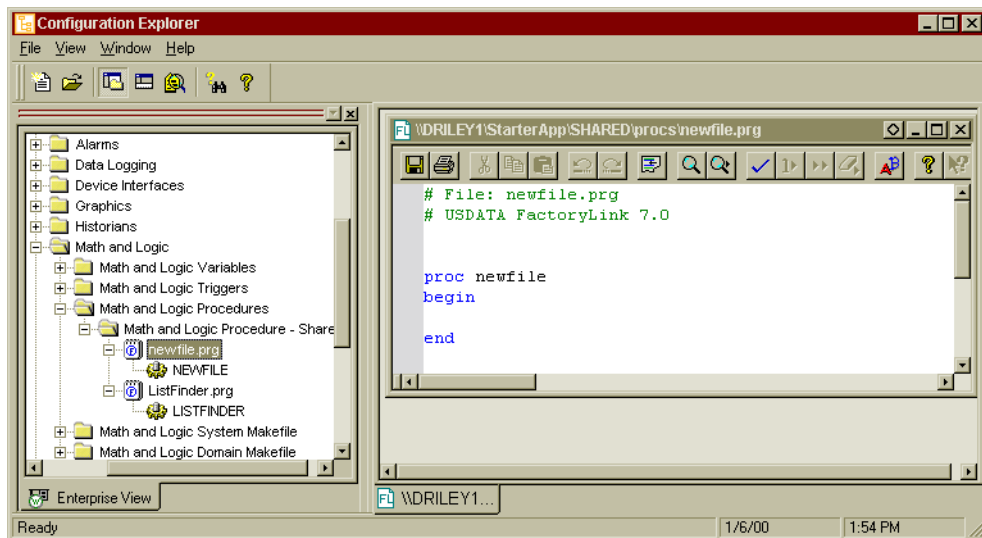
- **CONFIGURING MATH AND LOGIC**
- *Program Files and Procedures*
-
-

Figure 1-13 File Download Dialog Box




Select the radio button: Open this file from its Current Location. Click the OK button. The program file displays. See [Figure 1-14](#) for an example of a new program file.

Figure 1-14 The New Program File in Configuration Explorer

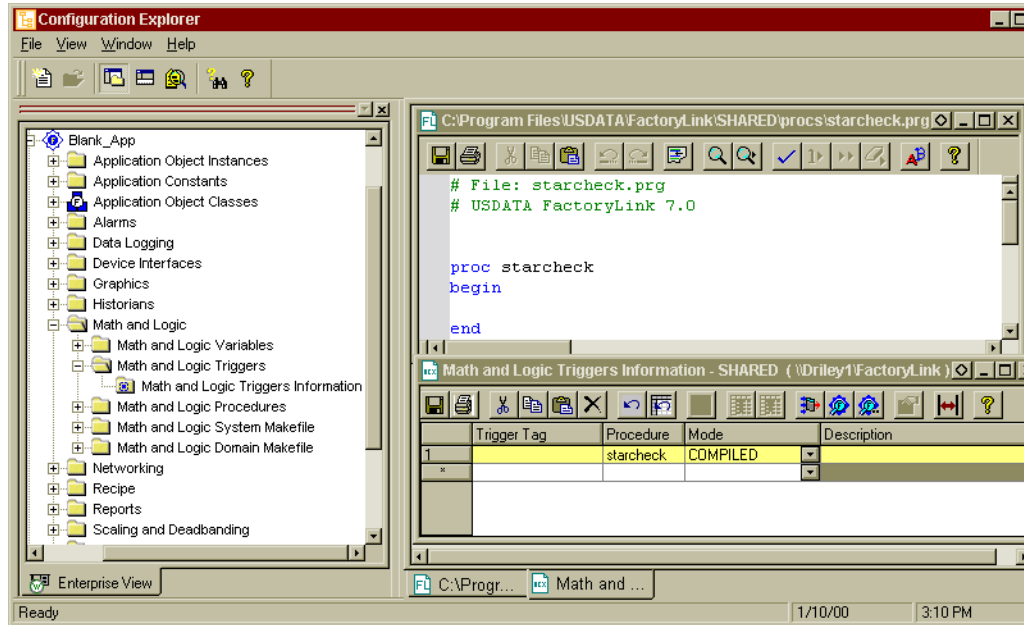



You can add code to the new procedure that was created when the program file was defined. Regardless, the procedure must remain in the program to enable FactoryLink to locate the program file.

To save the program file at any time click the save icon  at the top of the edit window.

The program information must be listed in the Math and Logic Triggers table (see [Figure 1-15](#)). Type the procedure name in the Procedure field and select the mode from the Mode drop-down field. A Trigger Tag is not necessary but can be added later if code is added to the procedure. See [Math and Logic Triggers](#) in this chapter for more information about the Math and Logic Triggers Information table.

Figure 1-15 Updated Math and Logic Trigger Information Table




To save the table entries at any time click the save icon  at the top of the edit window or use the keycode.

One or many additional procedures can be added to the program file.

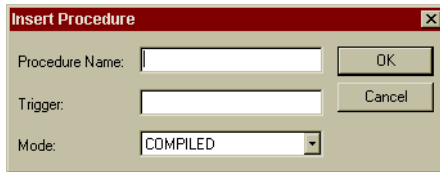
Add New Procedures

With the program file displayed in Configuration Explorer, position the cursor at the beginning of the line where you will add the new procedure.

To add a procedure: click the insert procedure icon . The Insert Procedure dialog box displays (see [Figure 1-16](#)).

- **CONFIGURING MATH AND LOGIC**
- *Program Files and Procedures*
-
-

Figure 1-16 Insert Procedure Dialog Box



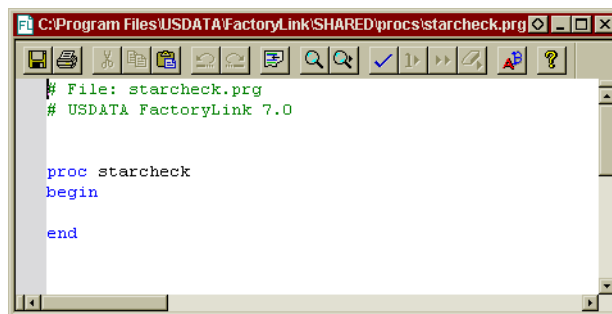
Use the guidelines in [Table 1-3](#) to type the Procedure Name, the Trigger tag name (if applicable) and select the Mode. Click the OK button. A template is inserted in the file to assist you to write the procedure.

Table 1-3 Procedure Definitions

Field Name	Description
Procedure Name	Procedure names are case sensitive. Use a consistent naming convention, for example, all caps.
Trigger	A Trigger tag name is required if the procedure is invoked by the change in the value of a tag. If the procedure is to be invoked by a call statement or used as a function to return values, leave this field blank. Tag names are case sensitive.
Mode	The choices are Interpreted or Compiled. All procedures in a program file must be the same.

Procedure definitions contain a minimum of three statements: proc, begin, and end. These statements are inserted into the program file (see [Figure 1-17](#)). For more information on the Math and Logic features, for example, text color, see [The Math and Logic Editor Reference Pages](#) at the end of this chapter.

Figure 1-17 New Procedures Template Inserted in Program File

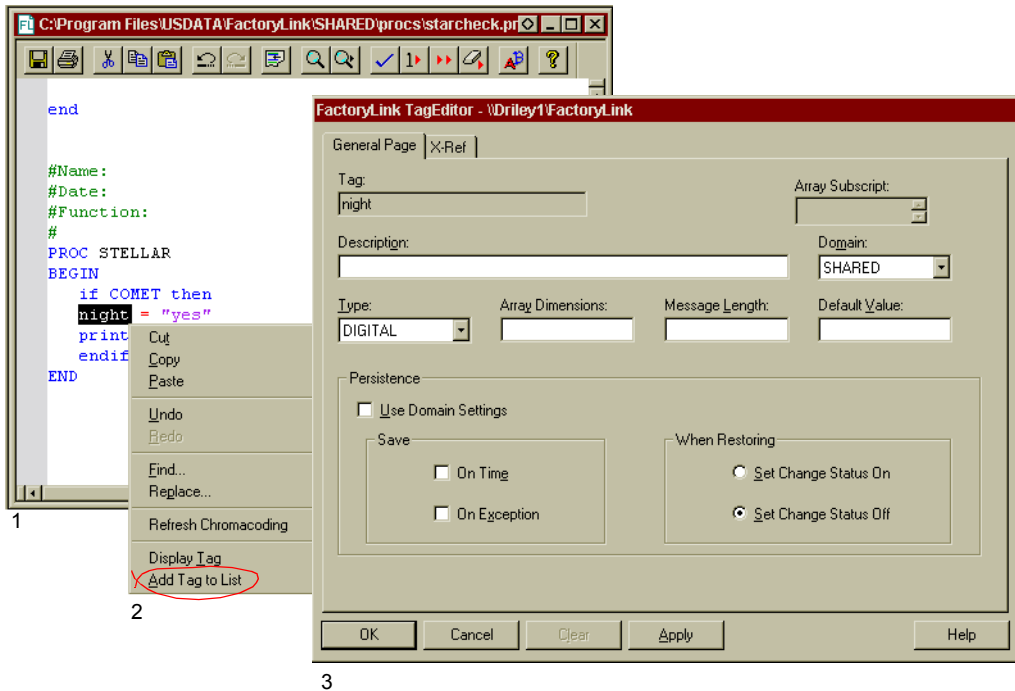


Add code to the template information to create the new procedure. Keywords are not case-sensitive, but tag names are. Use the [FactoryLink Configuration Guide](#) as a reference for syntax, keywords, structure and CML requirements.

A local variable (tag) can be declared in the program. If it is added to the top of the file above the initial BEGIN keyword, it is available to all procedures in all program files. Or, the local variables can be added at the procedure level.


Global variables (tags) are added to a procedure by typing the tag name in the procedure, highlighting the name. Right-click the tag name select Add to Tag List from the menu. The FactoryLink Tag Editor dialog box appears to enable definition of the tag. See [Figure 1-18](#). For more information regarding tag definitions see the [Configuration Explorer Manual](#). The tag color changes to blue when definition is completed.

Figure 1-18 Adding a Global Tag to a Procedure



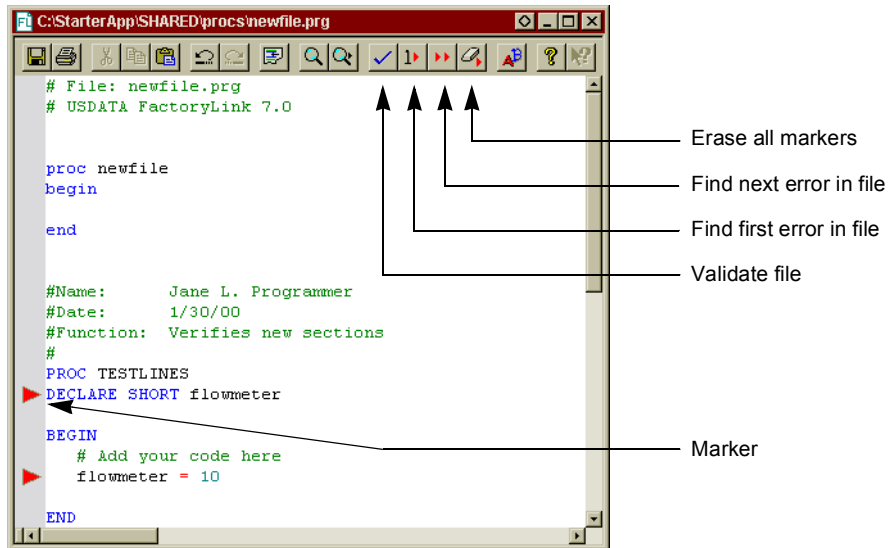
Once defined, the tag name appears in the Xref Table, the Tag browser, Table and the Object Table in addition to the Math and Logic Variables table. Global variables (tags) can also be added at any time to the Math and Logic Variables Information table and then typed into the procedure. Type the variable (tag) name and the variable text color changes from black to blue indicating it has already been defined.

- **CONFIGURING MATH AND LOGIC**
- *Program Files and Procedures*
-
-

After you have typed the procedure, you must validate it to check for syntax errors. Click the validate icon  to verify the syntax, for example, matching braces, parens and brackets and correct use of operators. The correct definition of local and global variables (tags) is checked plus the essential keywords are present (begin, end, proc).


If there are no errors, the system reports nothing. If there are errors, red triangle markers display in the left hand margin for each line with an error (see [Figure 1-19](#)). Correct the errors and revalidate the program.

Figure 1-19 Errors Indicated in the Math and Logic Editor



User inserted markers and error markers appear the same as error markers. Markers can be toggled with the keycode <Ctrl + F2> on a specific line or added to one to many lines using the find function. All previously set markers are erased when the validate function is performed.

For more information on the edit icons and keycodes, see [The Math and Logic Editor Reference Pages](#) in this chapter.

Click the save icon  after the editing session is complete.

MODIFY MAKEFILES

A makefile is a file containing all the information needed by the CCCML utility to compile the .C files produced by PARSECML and create an executable for the current domain. The name of the makefile used by CCCML is cml.mak, and is unique for each operating system.

The cml.mak file, located in the {FLINK}/CML directory, typically contains the following information to create the final executable file:

- Name of the C compiler to use for a given operating system
- Command-line switches to be used when compiling
- Name of the operating system's object linker
- Linker command-line switches
- References to the FactoryLink libraries to be linked
- References to the developer-supplied libraries to be linked

???????????????? The CML file does not exist in any directory. What's the deal??
 ?????????????????

As an aid for advanced users, CML provides a method for editing the cml.mak file. You can change the compiler and linker options, specify command-line switches, and specify which object files and libraries to link, giving you the flexibility to create a makefile unique to an application for a given domain.

A domain-specific makefile does not exist until you create one. When the Math and Logic System Makefile item is expanded, this item contains placeholder for a system makefile named cml.mak. This file opens in the Math and Logic editor and can be used to set the defaults to control the compile job instructions for CML procedures.

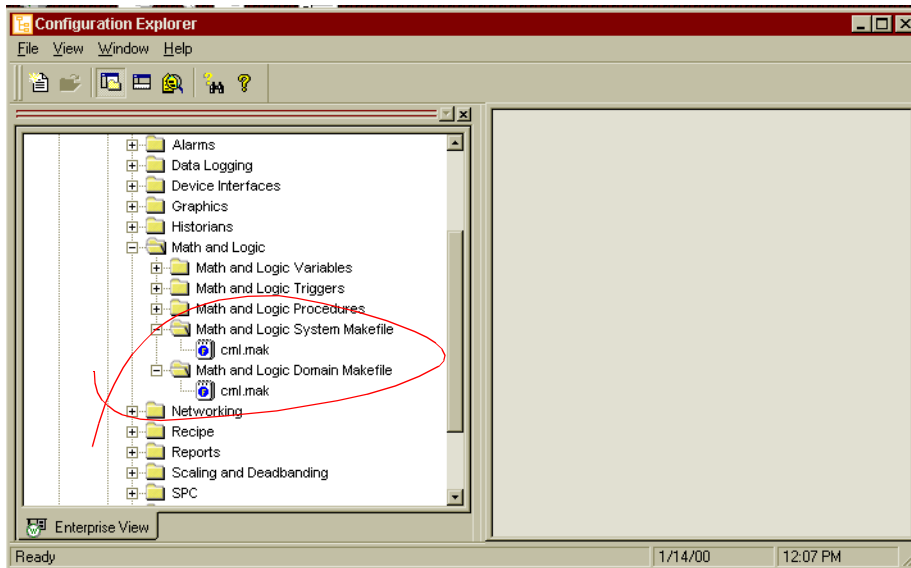
CML provides two file options:

- Math and Logic System Makefile
- Math and Logic Domain Makefile

Configuration Explorer provides access and the Math and Logic editing tools for the makefiles (see [Figure 1-20](#)).

- **CONFIGURING MATH AND LOGIC**
- *Modify Makefiles*
-
-

Figure 1-20 Access to the Makefiles




Math and Logic System Makefile

Any changes made to this file are global; they apply to all applications on the system.

Copy the cml.mak file from the {FLINK}/CML directory to the {FLAPP}/SHARED/CML directory for the SHARED domain.

To edit cml.mak: Expand the Math and Logic System Makefile item to display cml.mak. Double-click cml.mak to open the file. The file opens in the Math and Logic editor containing the cml.mak file from the {FLINK}/CML directory. Edit the file as required.

Any definitions in the system-specific makefile in the application directory override the definitions in the master makefile in /FLINK/CML directory.

Click the save icon  after the editing session is complete.


Math and Logic Domain Makefile

This option is provided for previous versions of FactoryLink installations that have been converted and still retain the user domain configurations. New installations and installations without user definitions should use the system makefile options.

Copy the cml.mak file from the {FLINK}/CML directory to the {FLAPP}/USER/CML directory for the USER domain.

To edit cml.mak: Expand the Math and Logic System Makefile item to display cml.mak. Double-click cml.mak to open the file. The file opens in the Math and Logic editor containing the cml.mak file from the {FLINK}/CML directory. Edit the file as required.

Any definitions in the domain-specific makefile in the application directory override the definitions in the master makefile in {FLINK}/CML directory.

Click the save icon  after the editing session is complete.

- **CONFIGURING MATH AND LOGIC**
- *Compiled Math and Logic (CML)*

-
-

COMPILED MATH AND LOGIC (CML)

CML is a combination of utilities and libraries that, at run time, create binary executable files from the program files you specified to run in the compiled mode.

The compile process begins at run time:

- 1 Translates the program (.prg) files into C source code
- 2 Puts the C code into files with an extension of .c
- 3 Compiles the .c files to produce object (.obj) files
- 4 Links the object files to the appropriate libraries to create binary executable (.exe) files
- 5 Runs the executable file as each program's associated trigger(s) are set.

Because FactoryLink applications can be configured in both the SHARED and USER domains, CML creates one executable file for each domain if it contains .prg files.

The file name of each executable is unique. The filename begins with a C and is followed by the domain name:

- {FLAPP}/SHARED/CML/CSHARED.EXE for the SHARED domain
- {FLAPP}/USER/CML/CUSER.EXE for the USER domain

CML Requirements

CML requires the following software and hardware:

- FactoryLink software version 4.1.3 or later
- Option bits:
 - Run-time-only systems—CML run-time option
 - Development systems—CML run-time and development options
- An ANSI-compatible C-language compiler for the development and run-time systems. Refer to the user manual for the particular compiler in use for information about compiler switches and setup options.

Option bits:

- Development Systems---the CML run-time and development options.
- Run-Time only Systems--the CML run-time option.

Compiled Math and Logic for Windows NT requires the environment variable FL_COMPILE to be set. This environment variable points to the directory of the compiler you are using. Set the FL_COMPILE variable from the Control Panel, System, Environment, and System Variable panels.

For example:

For Microsoft C++ 4,

```
FL_COMPILE=c:\MSDEV
```

For Microsoft C++ 5,

```
FL_COMPILE=c:\DEVSTUDIO\VC
```

Installing to c:\DEVSTUDIO directory is recommended.

Windows NT:

```
FL_COMPILE=DRIVE:\COMPILER_DIRECTORY
```

Running CML

CML compiles and runs on both development systems and run-time systems.

CML on a Development System

Before starting the Run-Time Manager, FLRUN invokes several utilities to compile programs into a single executable file. The compiled programs will have COMPILED entered in the Mode field of the Math and Logic Triggers Information table.

CML on a Run-Time-Only System

The CML development system executables must be transferred from the development system to the run-time system to run CML on a run-time-only system. How you do this depends on whether the development and run-time systems run on the same operating system:

Perform the following steps to run CML on a run-time-only system if the operating system for the development and run-time system is the same:

- 1 Use either of the following methods to transfer the CML executables to the run-time system:
 - Use the FLSAVE and FLREST utilities to perform a save and restore of the application from the development system to the run-time system. This saves and restores the compiled CML task along with the rest of the application.
 - Copy the executables from {FLAPP}/USER/CML or {FLAPP}/SHARED/CML on the development system to the same path on the run-time system.

- **CONFIGURING MATH AND LOGIC**

- *Compiled Math and Logic (CML)*

-
-
-
- 2 Start CML. Depending on whether the R flag was set in the System Configuration Information panel, do one of the following. If the R flag was:
 - Set, enter FLRUN.
 - Not set, start CML from the Run-Time Manager.

The compile process begins and CML creates the executables. Because the development and run-time operating systems are the same, CML runs as is.

Different Development and Run-time Operating Systems

Perform the following steps to run CML on a run-time-only system if the operating systems for the development and run-time systems are different:

- 1 Use the FLSAVE utility to perform a save of the application from the development system.

Because of the different operating systems, CML will not run as originally compiled and must be recompiled either on the run-time system or on a system with the same operating system as the run-time system. A compiler is required for the system you will recompile CML on.

- 2 Use the FLREST utility to perform a restore of the application to the system you will recompile CML on.
- 3 Enter FLRUN to begin the compile process CML creates the executables during.
- 4 Copy the CML executables from /FLAPP/USER/CML and/or /FLAPP/SHARED/CML to the same path on the run-time system if you recompiled on a system other than the run-time system.

CML does not recompile every time you enter FLRUN. Once CML has compiled the program files into executable files, it recompiles only if you change a program file.

CML UTILITIES CALL SEQUENCE

CML includes three utilities that create the executables CML uses at run time:

- MKCML
- PARSECML
- CCCML

At run time FLRUN is executed and starts a specific utilities call sequence (see [Figure 1-21](#)):

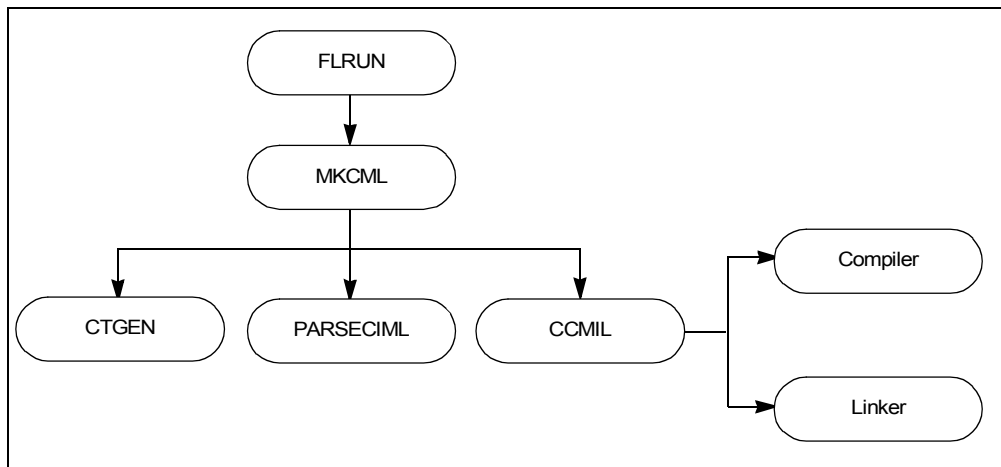
- 1 FLRUN calls the MKCML utility. The FLRUN command sets the FactoryLink path, the application directory path, the user name, and the domain name to the environment variables and turns off the verbose-level and clean-build parameters.

Note: CTGEN (and GENDEF) run normally as part of FLRUN. If you are debugging and need to run the items separately, you should always run CTGEN and GENDEF before running MKCML.

Is this true as MKCML calls ctgen???? Do they both call ctgen???
What does gendef do??

- 2 MKCML calls CTGEN, which ensures the Math and Logic .ct file is up to date.
- 3 MKCML calls PARSECML to produce .c files from the program files.
- 4 MKCML then calls CCCML to compile the .c files into object files using an external compiler. Using an object linker, the object files are linked with library files into binary executable files.

Figure 1-21 CML Utilities Call Hierarchy



- **CONFIGURING MATH AND LOGIC**

- *CML Utilities Call Sequence*

-
-

MKCML

The MKCML utility is a shell that calls the PARSECML and CCCML utilities as needed for the current application. For each domain, MKCML checks the dependencies between the configuration tables (named `iml.ct` for both IML and CML) and the program files. MKCML performs these tasks:

- Calls CTGEN which compares `iml.ct` against the database files. If the database files have a later time/date stamp than `iml.ct`, CTGEN rebuilds `iml.ct` to bring it up to date.
- Determines whether the time/date of `iml.ct` has changed. If so, MKCML reproduces and recompiles all of the `.C` files by calling PARSECML and CCCML.

When you redirect the output of MKCML to a file, the messages displayed in the dump seem to be out of order because of the method used by the operating system to buffer and output messages. If you do not redirect the output of MKCML, the messages are reported to the standard output in the correct order.

PARSECML

The PARSECML utility parses the application program files and produces `.c` files for each domain. It produces a `.c` file for each program file if the program Mode field is set to COMPILED in the Math and Logic Triggers Information table.

This utility also checks the dependencies between the program files and the `.c` files to see if any procedures have been updated since the `.c` files were last produced.

PARSECML has various levels of debugging via the `-Vx` parameter that can generate more detailed output or even add debugging statements to the C code.

CCCML

The CCCML utility compiles each `.c` file produced by PARSECML into an object file using an external compiler. It then links the object files with the FactoryLink and developer-supplied libraries into a binary executable. To determine the name of the compiler to use for a specific operating system, CCCML uses a special file called a makefile that is named: `{FLINK}/CML/CML.MAK`.

Its debugging levels provide minimal information, for example, the exact command line used to compile and link the code.

`{FLINK}/CML/CML.MAK` is platform dependent and is the heart of the portability of FactoryLink. `{an oxymoron????}` The following command lines are an example from OS/2:

`SRCSUFFIX - c`

OBJSUFFIX - obj

Dave/Mike -- What do above lines tell me???????

- **CONFIGURING MATH AND LOGIC**

- *CML Variables*

CML Variables

CML variables provide manipulation of the CML environment (see [Table 1-4](#)).

Table 1-4 Miscellaneous CML Commands

Command	Definition	Example
CC	Designates the command line compiler.	CC
CCFLAGS	Specifies all of the command line options for compiling the .C files.	CCFLAGS - -dos2 -AL -Au -Od -Zp -G2s -nologo -c -I{FLINK}\inc
CMLOBJS	The stock CML files that are not application dependent. They are liable to change as versions change	CMLOBJS glvars.obj CMLOBJS cmlprocs.obj
USEROBS	USEROBS allows for the inclusion of user-defined object module at link time.	USEROBS
LINK	Designates the command line linker	LKFLAGS - /NOE/ST:16384/se:512
CMLLKOBJS USERLKOBJS	These variables also allow for the inclusion of object modules that are not usually part of CML.	CMLLKOBJS USERLKOBJS
CMLLIBS USERLIBS	These variables also allow for the inclusion of libraries not usually part of CML.	CMLLIBS {FLINK}\LIB\FLIB.LIB\ CMLLIBS {FLINK}\LIB\CML.LIB USERLIBS
DEFFILE	This specifies the link definition file. It contains special information about window attributes, resources, or compiled output options. If this is altered, your customer support representative no longer supports it.	DEFFILE - {FLINK}\CML\CML.DEF
TARGET	This is the destination executable.	TARGET {FLAPP}\ {FLDOMAIN}\ cm,\lc{FLDOMAIN}.exe

User-defined C language includes files that use quotes; for example, #include "sample.h" should be placed in the {FLAPP}/{DOMAIN}/CML directory. Include files of the form #include <sample.h> should be placed in the path searched by the compiler. You may want to place them in the {FLINK}/INC directory. However, the include files are not saved with the

application during a multi-platform save (MPS). The best place to put the include files is in the `{FLAPP}/{DOMAIN}/CML` to ensure the fields are saved with the application when an MPS is performed. If you place the include files in the latter directory, you should add the following to the `cml.mak` file on the line `CFLAGS`:

```
-dos2 -AL -Au -Od -Zp -G2s -nologo -c -I{FLINK}\inc -I{FLAPP}\ {FLDOMAIN}\CML
```

Running the Utilities from the Command Prompt Window

CML is designed so each of the CML utilities can be started from the command prompt window. This is useful when only a portion of the compile process needs to be processed.

The command line parameters used by all CML utilities are described in the [Table 1-5](#).

Table 1-5 CML Command Line Parameters

Parameter	Description
-P	Sets the path to the FactoryLink program files.
-A	Sets the path to the application directory.
-U	Sets the user name.
-N	Sets the domain name.
-Vx	Sets the verbose (debug) level to <i>x</i> .
-C	Performs a clean build, reproducing all files from scratch.

- **CONFIGURING MATH AND LOGIC**

- *CML Variables*

-
-

Verbose-Level Parameters

When you use a verbose-level parameter, the utility displays messages about its progress as it performs its part of the compile process. This serves as a debugging aid.

[Table 1-6](#) shows the messages produced by each utility at the verbose level indicated.

Table 1-6 Verbose Setting Messages by Level

Utility	Verbose Level	Result Displayed	Run-time Effects
MKCML	1 or higher	Application name and domains as they are processed.	None
CCCML	1 or higher	—Message, “Not authorized to run Math and Logic” if the system cannot find the run-time bit. —Current application and domain being processed. —Message, “No .PRG files are configured as COMPILED”. —Message echoing the command line that calls the compiler or linker before making the call. —Names of each file as it is compiled. —Message indicating all files are up to date.	None
PARSCML	1	Name of each .c file name as it is produced.	None
PARSCML	2	Verbose level 1 message, plus: —The comments, containing the original source lines of the Math and Logic program, placed by the utility at the start of the generated C code. —All programs as they are parsed.	None
PARSCML	3 or higher	Verbose level 1 and 2 messages.	Print a statement upon entry and exit of procedure.
PARSCML	4	Verbose Level 1, 2.	Print each line as it executes.

CALLING C CODE

Use the following CML-specific keywords to call C code in a Math and Logic program:

- `cfunc`
- `cbegin`
- `cend`

Using `cfunc`

Use the keyword `cfunc` to declare standard C functions and user-defined C functions as callable in-line functions within a CML program. In-line C functions allow a CML program to call a C function directly without opening a C code block. The function must be declared before it is called.

The C code generated by CML provides prototypes for standard library functions; however, it does not include prototypes for user-defined C functions. You must provide function prototypes for all user-defined functions. Including a function without a prototype may result in compiler warnings regarding the missing functions.

Use only C functions that use the Math and Logic data types of `SHORT`, `LONG`, `FLOAT`, and `STRING` with `cfunc`. Although a C function may use any data type internally, its interface to Math and Logic must use only these types.

In the following example, `testfunc` is declared to use four arguments whose values are `SHORT`, `LONG`, `FLOAT`, and `STRING` data types and to return a value with a `SHORT` data type:

```
DECLARE cfunc SHORT testfunc(SHORT, LONG, FLOAT, STRING)
```

You may declare C functions to return the following data types:

Function: **Value returned:**

<code>SHORT</code>	Short-integer
<code>LONG</code>	Long-integer
<code>FLOAT</code>	Floating-point
<code>STRING</code>	String
<code>VOID</code>	None

The `VOID` data type is unique to CML. Use `VOID` when declaring a function not required to return a value.

Do not use `VOID` in programs designed to run in interpreted mode.

- **CONFIGURING MATH AND LOGIC**

- *Calling C Code*

-
-

cfunc Examples

The following examples show how to use `cfunc`:

Example 1—uses `cfunc` to declare the standard C function `strcmp()` for use within a CML program:

```
DECLARE cfunc SHORT strcmp(STRING, STRING)

PROC TEST(STRING s1)
BEGIN

  IF strcmp(s1,"QUIT")=0 THEN
    PRINT "QUITTING\n"
  ENDIF

END
```

The function `strcmp()` compares two strings and returns a value that indicates their relationship. In this program, `strcmp` compares the input string `s1` to the string `QUIT` and is declared to have a return value of the data type `SHORT`.

- If the return value equals 0, then `s1` is identical to `QUIT` and the program prints the message `QUITTING`.
- If the return value is less than or greater than 0, the program prints nothing.

C functions declared using `cfunc` have full data conversion wrapped around them, meaning any data type can be passed to and returned from them.

Given the previous sample code, the following program is legal within CML:

```
PROC MYPROC
BEGIN

  DECLARE FLOAT f
  DECLARE LONG k
  DECLARE STRING buff

  buff=strcmp(f,k)

END
```

In this program, `strcmp` converts the `FLOAT` value `f` and the `LONG` value `k` to strings, compares the two strings, and then returns a number (`buff`) that indicates whether the comparison was less than, greater than, or equal to zero. This comparison is

If $f < k$, then `buff` is a number less than 0.

If $f = k$, then `buff` is equal to 0.

If $f > k$, then `buff` is a number greater than 0.

Example 2—uses `cfunc` to declare the function `testfunc` which has a return data type of `VOID`:

```
DECLARE cfunc VOID testfunc(FLOAT)

PROC MYPROC
BEGIN

DECLARE FLOAT flp
  flp=100.0
  testfunc(flp)

END
```

In this program, the declared floating-point variable `flp` is set to 100.0 and this value is passed to the function `testfunc`. Note that `VOID` is entered in place of the data type for the function's return value. This is because the program is only passing a value to `testfunc` and the function is not required to return a value.

Using `cbegin` and `cend`

You can use the keywords `cbegin` and `cend` to embed C code directly into a CML procedure. Between these keywords, you can call external library functions and manipulate structures and pointers Math and Logic does not support; however, you cannot declare C variables inside a `cbegin/cend` block already within the scope of a procedure. When you declare a C variable, the declaration block from `cbegin` to `cend` must be displayed outside the procedure, above the `PROC` statement. Refer to the declaration of static FILE `*Fp=stderr`; in Example 2.

The `cbegin` and `cend` statement must each be on a line by itself with no preceding tabs or spaces. All lines between these two keywords (the C code block) are passed directly to the `.C` file `PARSECML` produces for this program.

The following examples show how to use the `cbegin` and `cend` keywords.

Example 1:

```
PROC TEST(String message)
BEGIN

DECLARE STRING buff

IF message="QUIT" THEN
  PRINT "FINISHED.\n"
ENDIF
```

- **CONFIGURING MATH AND LOGIC**

- *Calling C Code*

-
-

```
cbegin
  sprintf(buff,"The message was %s\n",message);
  fprintf(stderr,buff);
cend

END
```

In this program, the `sprintf` and `fprintf` functions, called between `cbegin` and `cend`, are passed directly to the .C file PARSECML generates for TEST. Note that local variables are within the scope of the C code block and can be accessed during calls to external functions.

Any C code blocks outside the body of a CML program are collected and moved to the top of the generated .C file, as shown in Example 2:

Example 2:

```
cbegin
#include "mylib.h"
cend

PROC TEST(String s1)
BEGIN
  PRINT "The message is ",s1
END

cbegin
static FILE *Fp=stderr;
cend

PROC SOMETHING (FLOAT f1)
BEGIN
cbegin
  fprintf(Fp,"%6.2g\n",f1);
cend

END
```

In this program file, the statement:

```
static FILE *Fp=stderr;
```

is moved to the top of the program file just after the line

```
include "mylib.h"
```

The following example shows how to access real-time database elements from within embedded C code blocks. It increments the values of two analog elements, Tag1 and Tag2[5], by 10.

```
cbegin

    int fl_tagname_to_id(TAG*, int, ...); /* function
    prototype missing from CML.H*/
    cend

PROC example
BEGIN
cbegin
{

    TAG tag[2];
    ANA value[2];

    fl_tagname_to_id(tag,2, "TAG1","TAG2[5]");
    fl_read(Task_id,tag,2,value);
    value[0] += 10;
    value[1] += 10;
    fl_write(Task_id,tag,2,value);

}
cend
END
```

Note: The variable `task_id` is a predefined global CML variable and does not need to be declared.

The following example shows how to manipulate message tags within embedded C code (`cbegin/cend` code blocks). This example reads from TAG1, adds X to the string, then writes the result to TAG2.

```
PROC ADD_X
BEGIN
cbegin
{

    #define MAX_LEN 80      /* default maximum
                           message length */
    int fl_tagname_to_id(TAG*, /* function prototype
    int,...);              missing from CML.H */
```

- **CONFIGURING MATH AND LOGIC**

- *Calling C Code*

-
-

```
TAG tags[2];
MSG tag1, tag2;
char string_buff[MAX_LEN+1]; /* max length plus
                               terminating 0 */

tag1.m_ptr=tag2.m_ptr=string_buf;
tag1.m_max=tag2.m_max=MAX_LEN;

fl_tagname_to_id(tags,2,"TAG1","TAG2");
fl_read(Task_id,&tags[0],1,&tag1);

strcat(string_buf,"X");
tag2.m_len=strlen(string_buf);
fl_write(Task_id,&tags[1],1,&tag2);

}
cend
END
```

When values are assigned to and read from MESSAGE tags in the normal syntax for the procedure files the MAX LEN field is limited to 1023 characters. All message values are truncated at 1023 characters. The function fl_write () must be called directly to store values longer than 1023 characters into a MESSAGE tag. The following example shows how to use a C macro to call the procedure msgstest to store a 90-character constant into the MESSAGE tag msgtag:

MSGTEST.PRG:

```
cbegin
void fl_tagname_to_id(TAG*,int,...);
#define assign_msg(tagname, value) {\
    TAG tag; \
    MSG msg; \
    char buf[] = value; \
    fl_tagname_to_id(&tag,1,tagname); \
    msg.m_ptr = buf; \
    msg.m_len = strlen(buf); \
    msg.m_max = strlen(buf)+100; /* leave plenty of room */
    fl_write(Task_id,&tag,1,&msg); \
}
cend

PROC msgstest
BEGIN
```

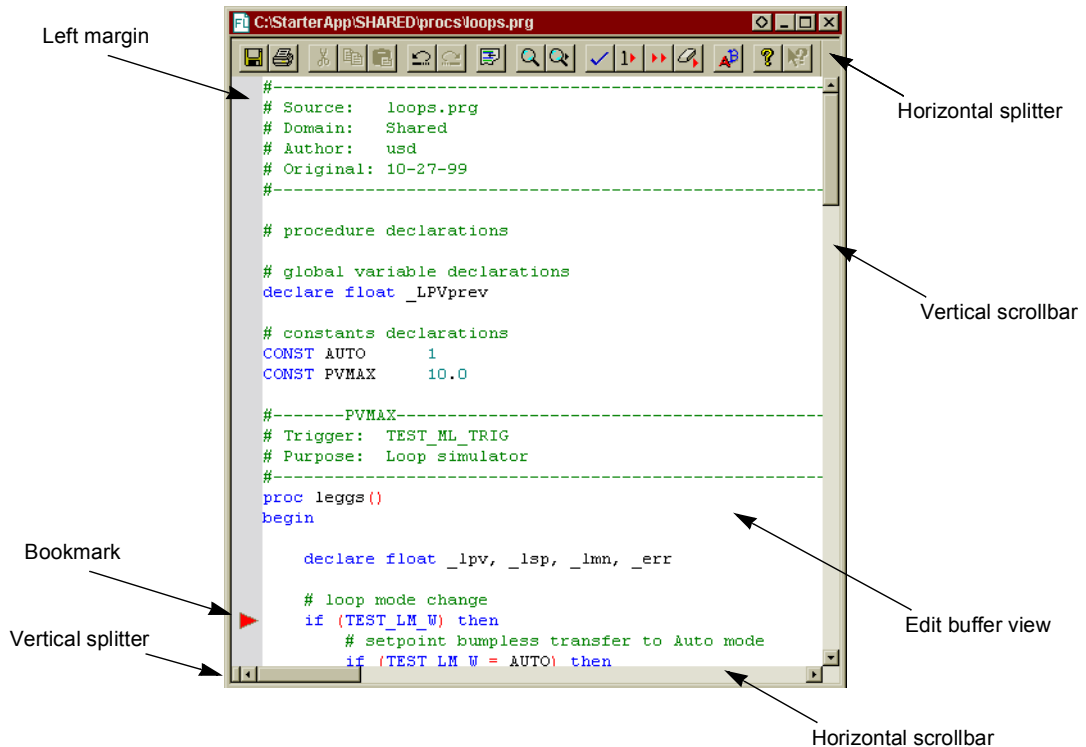

- **CONFIGURING MATH AND LOGIC**
- *The Math and Logic Editor Reference Pages*

THE MATH AND LOGIC EDITOR REFERENCE PAGES

The Math and Logic editor is a Single Document Interface (SDI) application that allows multiple concurrent instances to run within the FactoryLink environment. The editor is accessed using the Configuration Explorer program and acts as an in-place active server.

Chromocoding, color assigned according to syntax, is provided by the editor to assist the user in identifying keywords, operators, declarations, global tags, commented lines and strings. See the example Math and Logic Editor file in [Figure 1-22](#). Find the definitions of the screen icons in [Table 1-7](#).

Figure 1-22 Math and Logic Editor Screen Example













Other features supported by the Math and Logic Editor are:

- Keystroke macros
- Split views of the same edit buffer which can be scrolled separately
- Column selection and manipulation to provide editing across columns

- Configurable window properties for customizable views
- Numerous edit commands with keystroke functions enabling, for example, quick copy-paste, drag and drop, find-replace, tab-to-space/space-to-tab conversion.
- Microsoft IntelliMouse® support for quick scrolling.







The syntax and keywords for the simplified language supported by the Math and Logic Editor are described in the [FactoryLink Configuration Guide](#). This language is supported for all IML programs. For CML programs, non compliant code is translated by FactoryLink to ANSI C and C++ standards by an interpreter during the FactoryLink start-up process.

Table 1-7 Math and Logic Editor Icons (Sheet 1 of 2)

Icon	Function	Description
	Save	Saves the changes in the selected (active) file
	Print	Prints the selected (active) file
	Cut	Copies the selected text
	Copy	Copies the selected text
	Paste	Pastes the information that has been saved in the clipboard
	Undo	Remove the last edit(s) in reverse sequence they were made. The undo function buffers the last XX edits to undo.
	Redo	Read last edit(s) in reverse sequence they were removed. The redo function buffers the last XX undo functions.
	Insert procedure	Inserts the text and initiates a new procedure.
	Find	Search for whole word, fragment and match case. This standard find function provides the addition of setting bookmarks at each find location to facilitate editing.
	Find next	Repeat last find with same properties.

- **CONFIGURING MATH AND LOGIC**
- *The Math and Logic Editor Reference Pages*
-
-

Table 1-7 Math and Logic Editor Icons (Sheet 2 of 2)

Icon	Function	Description
	Validate	Checks the file for: matching braces, parens and brackets definition of local and global variables (tags) basic syntax such as correct use of operators essential keywords are preset (BEGIN, END, PROC)
	Go to first error	Advances the cursor to the first line with an error in the file.
	Go to next error	Advances the cursor to the next line containing an error. Wraps to the top of the file if cursor is on or below the last error line.
	Clear all marks	Clears all bookmarks and error marks from the file.
	Font	Displays the Font dialog box to enable selection of font type, style and size
	Display the About information box.	Shows current Math and Logic software version plus the location of the FLAPP and FLINK directories

Mouse Functions

Custom mouse functions are described in [Table 1-8](#).

Table 1-8 Mouse Actions in the Math and Logic Editor

Mouse Action	Result
Left-button click over text	Inserts the cursor in the text at the position the mouse is clicked
Right-button click	Displays the Math and Logic context menu
Left-button down over selection, and drag	Moves the selected text
Control + Left-button down over selection, and drag	Copies the selected text
Left-Button click over left margin	Selects line
Left-Button click over left margin, and drag	Selects multiple lines
Alt + Left-Button down, and drag	Select columns of text
Left-Button double click over text	Select word under cursor

Table 1-8 Mouse Actions in the Math and Logic Editor

Mouse Action	Result
Spin IntelliMouse mouse wheel	Scroll the window vertically
Single click IntelliMouse mouse wheel	Select the word under the cursor
Double click IntelliMouse mouse wheel	Select the line under the cursor
Click and drag splitter bar	Split the window into multiple views or adjust the current splitter position
Double click splitter bar	Split the window in half into multiple views or combine the window if already split

Keyboard Shortcuts

For users who prefer keyboard commands, the Math and Logic editor provides an abundance of keyboard shortcuts. See [Table 1-9](#) for a complete list of the actions and keycodes arranged alphabetically by keyword.

Table 1-9 Keyboard Shortcuts (Sheet 1 of 4)

Keyword	Action	Keycode
BookmarkNext	Go to next bookmark	F2
BookmarkPrev	Previous bookmark	Shift + F2
BookmarkToggle	Set/delete bookmark on current line	Control + F2
CharLeftExtend	Select character(s) left and extend	Shift + Left Arrow
CharRightExtend	Select character(s) right and extend	Shift + Right Arrow
Copy	Copy	Control + C
	Copy	Control + Insert
Cut	Cut selection	Shift + Delete
		Control + x
		Control + Alt + w
Delete	Delete forward	Delete
	Delete back	Backspace
DocumentEnd	Go to end of document	Control + End

- **CONFIGURING MATH AND LOGIC**
- *The Math and Logic Editor Reference Pages*
-
-

Table 1-9 Keyboard Shortcuts (Sheet 2 of 4)

Keyword	Action	Keycode
DocumentEndExtend	Select from cursor location to end of document	Control + Shift + End
DocumentStart	Go to start of document	Control + Home
DocumentStartExtend	Select from cursor location to start of document	Control + Shift + Home
Find	Open find dialog box	Alt + F3
Find		Control + F
FindNext	Find next match	F3
FindPrev	Find previous word	Shift + F3
FindReplace	Open find and replace dialog box	Control + Alt + F3
GoToLine	Open go-to-line dialog box	Control + G
GoToMatchBrace	Go to matching brace	Control +]
Home	Go to start of line	Home
HomeExtend	Select from cursor back to start of line	Shift + Home
LineCut	Cut current line	Control + Y
LineDownNextend	Select from cursor to line below cursor	Shift + Down
LineEnd	Go to end of line	End
LineEndExtend	Select from cursor to end of line	Shift + End
LineOpenAbove	Add line above current line	Control + Shift + N
LineUpExtend	Select from cursor to line below cursor	Shift + Up
LowercaseSelection	Change selection to lower case	Control + U
PageDownExtend	Select from cursor to top of next page	Shift + Page Down
PageUpExtend	Select from cursor to bottom of previous page	Shift + Page Up
Paste	Paste	Control + V
		Shift + Insert
Properties	Display Window Properties dialog box	Alt + Enter

Table 1-9 Keyboard Shortcuts (Sheet 3 of 4)

Keyword	Action	Keycode
RecordMacro	Record a new macro	Control + Shift + R
SelectLine	Select line	Control + Alt + F8
SentenceCut	Cut current sentence and all sentences above up to first occurring blank line	Control + Alt + K
SentenceLeft	Go to beginning of previous line after line break (previous sentence)	Control + Alt + Left
SentenceRight	Go to beginning of next blank line (next sentence)	Control + Alt + Right
SetRepeatCount	Set repeat counter (use this in conjunction with macros to repeat the macro)	Control + R
TabInsertSelection	Change every four spaces in selection to tabs	Control + Shift + T
TabRemoveSelection	Change tabs in selection to four spaces per tab	Control + Shift + Space
ToggleOvertyp	Toggle between overtype and insert	Insert
ToggleWhitespaceDisplay	Toggle between white space and no white space display	Control + Alt + T
Undo	Undo last edit(s) in reverse sequence they were made. The undo function buffers the last XX edits to undo.	Control + Z
		Alt + Backspace
UnindentSelection	Unindent Selection (doesn't work - only moves tab to right)	Shift + Tab
UppercaseSelection	Change selection to uppercase	Control + Shift + U
WindowScrollDown	Cursor remains in current line but file scrolls down	Control + Up Arrow
WindowScrollLeft	Window scrolls to the left one character at a time	Control + PageUp
WindowScrollRight	Window scrolls to the right one character at a time	Control + PageDown
WindowScrollUp	Cursor remains in current line but window scrolls up	Control + Down Arrow

- **CONFIGURING MATH AND LOGIC**
- *The Math and Logic Editor Reference Pages*
-
-

Table 1-9 Keyboard Shortcuts (Sheet 4 of 4)

Keyword	Action	Keycode
WordDeleteToEnd	Delete the word or symbol to the right	Control + Delete
WordDeleteToStart	Delete the word or symbol to the left	Control + Backspace
WordLeft	Move the cursor one word to the left	Control + Left Arrow
WordLeftExtend	Select word(s) to the left	Control + Shift + Left Arrow
WordRight	Move the cursor one word to the right	Control + Right Arrow
WordRightExtend	Select word(s) to the right	Control + Shift + Right Arrow

Chromocoding

Color helps to define the purpose of the text at a glance. See [Table 1-10](#) for a description of the colors and the functions they describe.

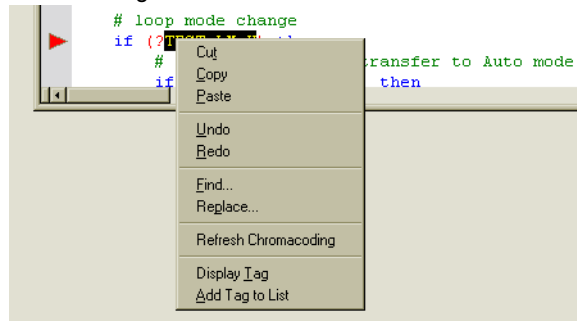
Table 1-10 Color Codes for the Math and Logic Editor

Color	Function
Red	Punctuation symbols
Blue	Reserved words and globally defined variables
Purple	Strings
Black	Text not globally defined
Green	Comments
Aquamarine	Numerics

Menu Commands

Right-click the mouse with the cursor positioned in the edit buffer to display a context menu (see [Figure 1-23](#)).

Figure 1-23 Math and Logic Context Menu



See [Table 1-11](#). for a list of menu commands and their corresponding actions.

Table 1-11 Right Click Menu Commands

Command	Action	Comment
Cut	Remove selected test	Standard cut function
Copy	Copy selected text	Standard copy function
Paste	Paste test from the clipboard	Standard paste function
Undo	Remove last edit	Remove the last edit(s) in reverse sequence they were made. The undo function buffers the last XX edits to undo.
Redo	Read last removed edit	Read last edit(s) in reverse sequence they were removed. The redo function buffers the last XX undo functions.
Find	Search for whole word, fragment and match case	Standard find function which provides the addition of setting bookmarks at each find location to facilitate editing.
Replace	Search for above and substitute new text	Standard replace function providing the ability to find and replace the entire document or just the selected text.
Refresh Chromocoding	Check syntax/function color settings	Standard window refresh function
Display Tag	Displays the FactoryLink TagEditor dialog box if the tag is not previously defined	Use this box to configure or modify the global tag and add it automatically to the Tag list and the Math and Logic Variables table.

- **CONFIGURING MATH AND LOGIC**
- *The Math and Logic Editor Reference Pages*
-
-

Table 1-11 Right Click Menu Commands

Command	Action	Comment
Add to Tag List	Displays the FactoryLink TagEditor dialog box if the tag is not previously defined	Use this box to configure or modify the global tag and add it automatically to the Tag list and the Math and Logic Variables table.