

USING OPC MATH & LOGIC

The Math & Logic feature allows you to program mathematical and logical operations that the OPC server will perform on its data. These operations can range from simple averaging calculations or pass/fail decisions up to complex operations involving algebraic and trigonometric functions, string manipulation, data sorting and Boolean expressions. The OPC server will then provide the results of the calculations to OPC clients just as it would any other data item.

Can't I just do that in the client or the PLC?

You might not be able to, and even if you can, you might not want to.

Limitations in PLC and client software capabilities

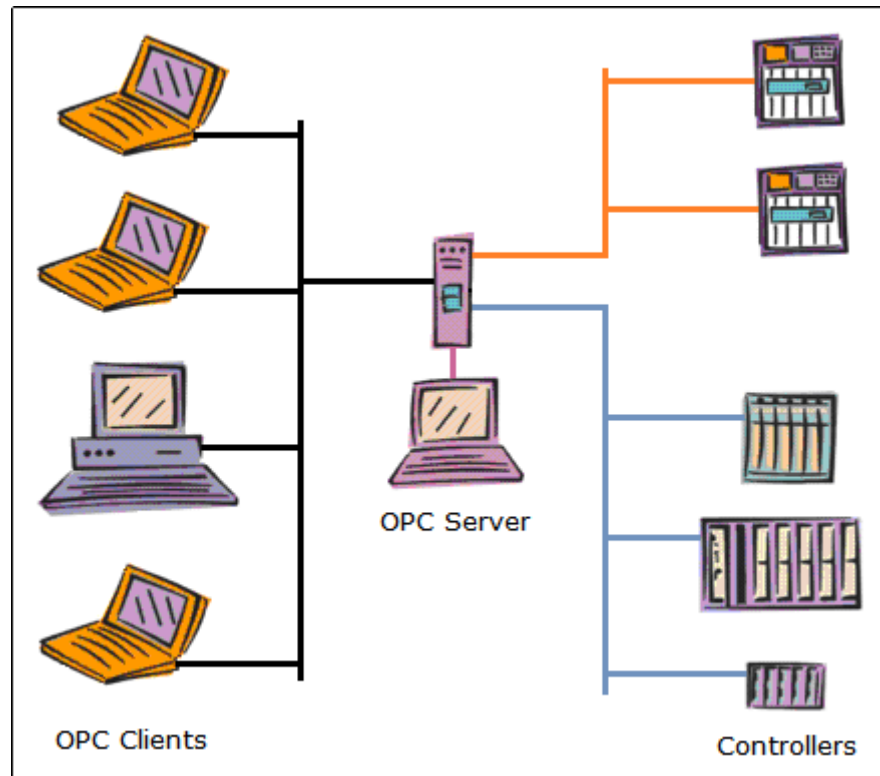
It is true that all PLCs can do logic, and most can do some simple math. But all of them are limited in these capabilities. Calculations that you can program quickly and easily in the OPC server may not be so simple in ladder logic. Most PLCs don't have built-in trig, log, exponential or sorting functions, so you would have to build those yourself. And some of your field devices may simply report data values, with no math or logic capability at all.

Some OPC client software packages have these kinds of functions available, but not all do. If you choose to do the math and logic in the client, you will limit yourself to using only the client software that has all of the math and logic capabilities you need.

Furthermore, Cyberlogic's Math & Logic includes pre-programmed functions, complex triggering capabilities and several debugging tools that are not available in PLCs or most client software.

The OPC server as "data central" for your system

If you have just one OPC client, you would be able to do all the programming there and have it all in one location. The same would be true if you have only one PLC. But if you have more, or think you may ever have more, then those approaches quickly lead to a lot of unnecessary, complex work.



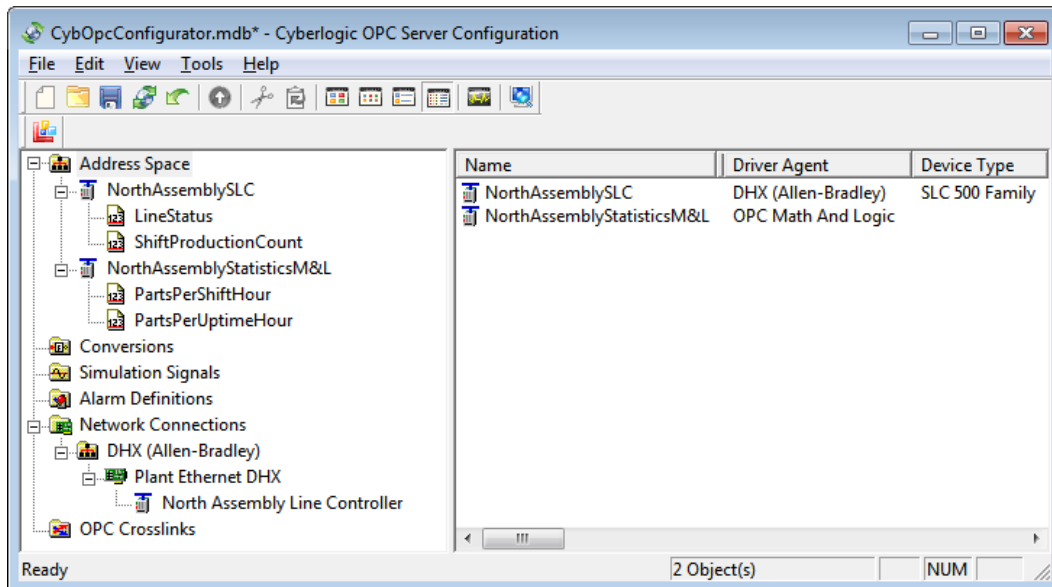
In the typical plant configuration shown above, many controllers pass data through the OPC server to many clients. If you want to do client-side calculations, you'll have to program the same functions in each of the clients. Similarly, if you want to program the calculations in the controllers, you'll have to do it in each of them. That means multiple copies to write and maintain. If the controllers or clients are not all identical, you'll need different versions of your programs, written in different languages. And you may have to purchase multiple copies of the software.

The OPC server is the central location in the system, making it the logical place to do the work. All the data flows through the OPC server, so all of it is available in one place, to let you do all of the programming one time, in one language, using one set of tools.

You may be able to reduce network traffic as well. If a calculation involves data from multiple PLCs, you won't need to pass those values back and forth among the controllers just for the sake of working the numbers. And if you can crunch the numbers down into a single result, you can pass that result to the clients instead of all the raw data they would need to do the calculations themselves.

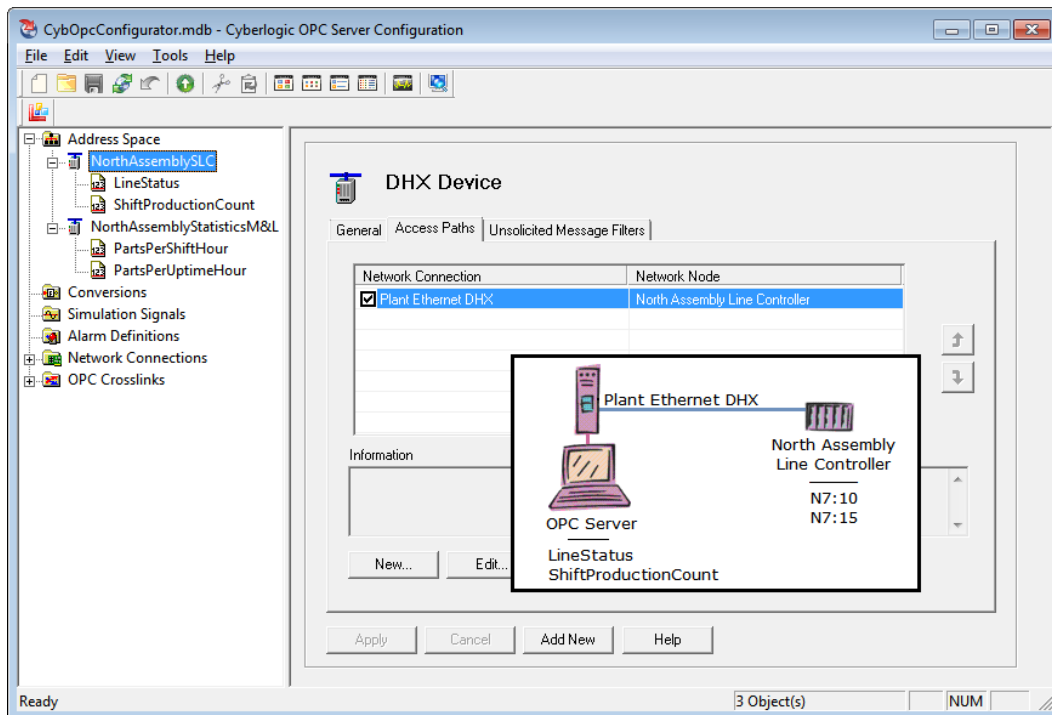
How does Math & Logic work in the OPC server?

In Cyberlogic OPC servers, you program Math & Logic as part of a data item. These data items exist in the OPC server's Address Space tree, within Math & Logic Devices.



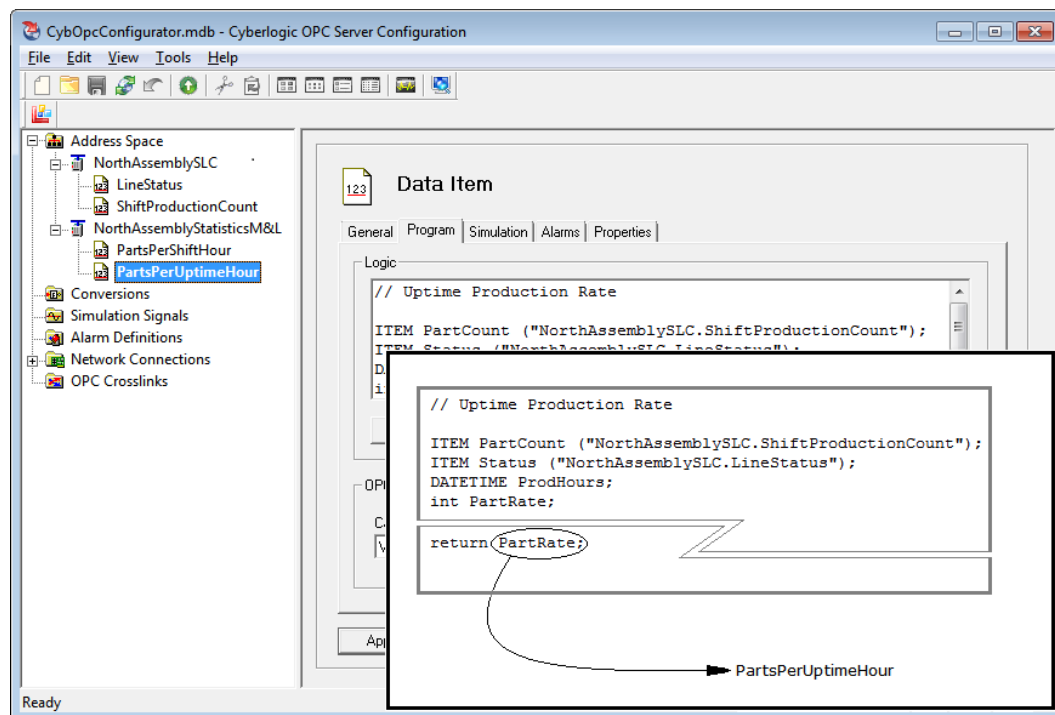
In the OPC server shown above, NorthAssemblySLC is an ordinary device associated with a SLC-500 controller. It contains data items LineStatus and ShiftProductionCount.

NorthAssemblyStatisticsM&L is a Math & Logic device, and contains Math & Logic data items PartsPerShiftHour and PartsPerUptimeHour.



When you configure a conventional data item, you place it in a device with an access path to a PLC, as shown in the figure above. The value of the data item simply reflects the value of a register you specify in that controller.

The device NorthAssemblySLC has an access path that uses the Plant Ethernet DHX network connection to communicate with the network node called North Assembly Line Controller. That node is a SLC-500 with registers N7:10 and N7:15. These registers provide the values for the OPC data items LineStatus and ShiftProductionCount.



When you configure a Math & Logic data item, you instead provide a program that produces a value as its result. This value then becomes the value of the Math & Logic data item.

The Math & Logic device NorthAssemblyStatisticsM&L contains Math & Logic data item PartsPerUptimeHour. You can then write a program to calculate the value for this data item. The program shown uses values from the NorthAssemblySLC device, along with some internal variables, to calculate a value called PartRate. When the program runs, it returns the value of PartRate, placing it in the data item PartsPerUptimeHour.

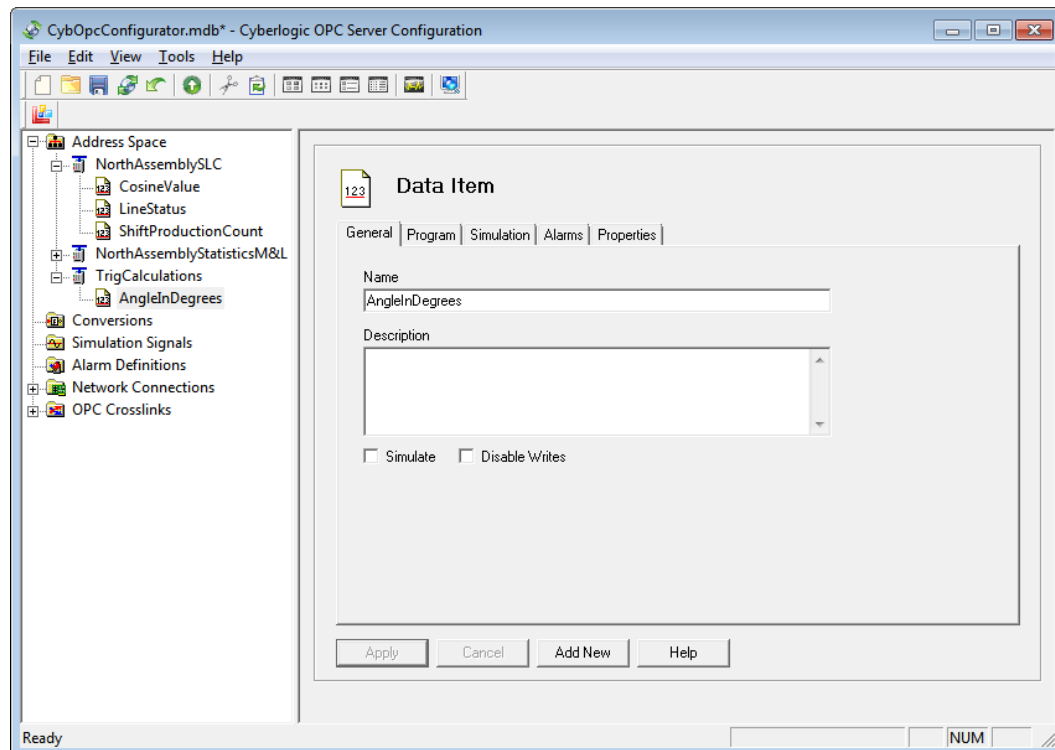
A Math & Logic program may write to additional data items, so it is possible for the program to have more than one output value. All of these data items can be used by clients, by other Math & Logic programs, and by crosslinks, just like any other data item.

The input values for the Math & Logic program may be conventional data items, other Math & Logic data items, or constant values.

How Do I Write Math & Logic Programs?

Let's create a simple program to illustrate the process. We'll take a value from a programmable controller, find the arc cosine of the value, convert that to degrees and return the result for the client applications to use. The result will be between 0 and 180 degrees.

However, the cosine of an angle must be between 1 and -1, so it is possible for the controller to give us an invalid value. In that case, we will return a value of -1 for the angle.



In the figure above, you can see that we've added the data item `CosineValue` to the `NorthAssemblySLC` device. This gets its data from the SLC, and holds the cosine value that we want to convert to an angle.

We have also created a new Math & Logic device called `TrigCalculations`. Within that device is Math & Logic data item `AngleInDegrees`. This data item contains the program that calculates the angle, and the value of the data item reflects the result of the calculation.

Our task now is to write the program for `AngleInDegrees`, and configure the `TrigCalculations` device to enable and run the program as needed. We could simply type the program in any text editor, copy the code and then paste it into the data item. However, the Math & Logic Editor is particularly suited to creating C-Logic programs quickly and efficiently, so we will use that editor.

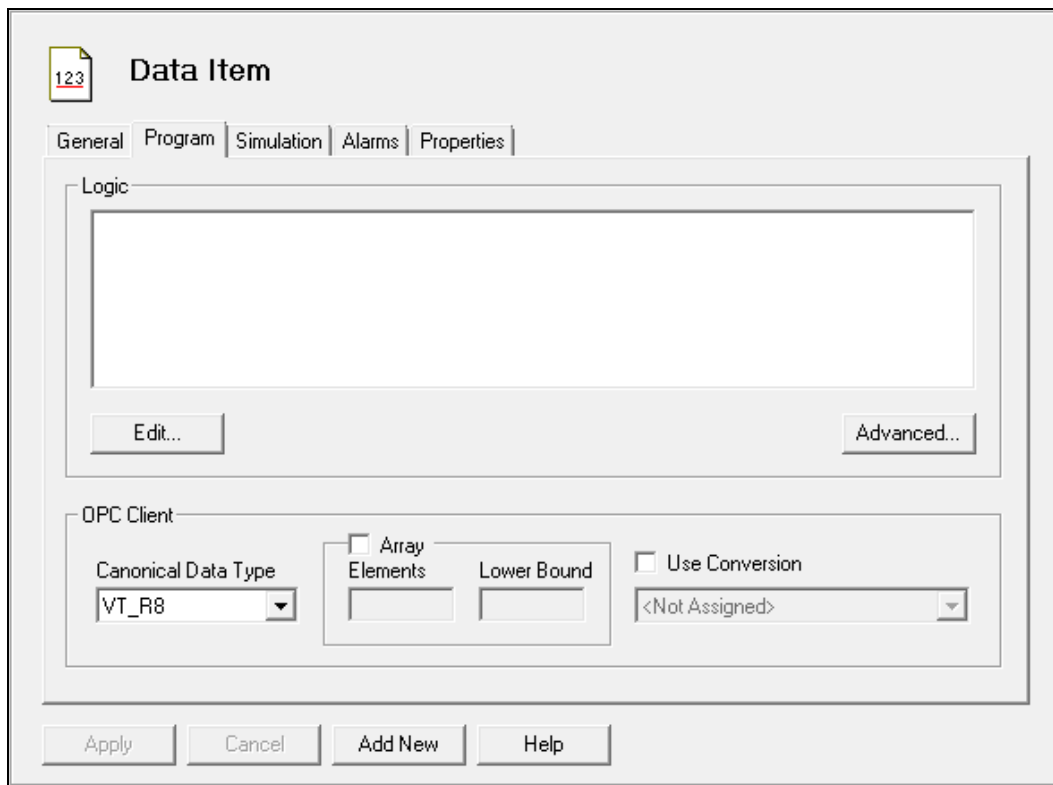
Here is the program we will use:

```
//This program reads the value of the CosineValue data item
//from the North Assembly SLC, and calculates the angle in
//degrees that corresponds to that value. The returned angle
//is in the range of 0-180 degrees. If the input value is
//invalid, the program returns -1.
//
//This is version 1.0.
//
ITEM CosVal ("NorthAssemblySLC.CosineValue");
VAR varAngle;

if((CosVal < -1) || (CosVal > 1))
{
    varAngle = -1;
}
else
{
    varAngle = Acos(CosVal) * 180 / pi;
}

return varAngle;
```

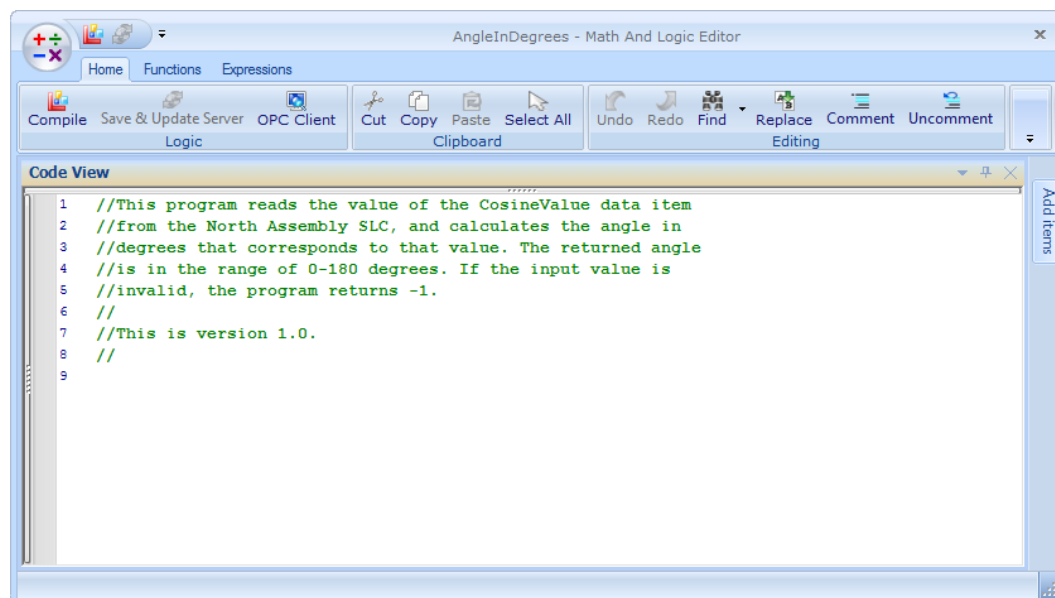
To begin, select **AngleInDegrees** in the **Address Space** tree, and then select its **Program** tab.



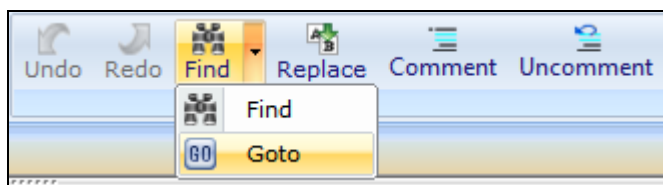
Click the **Edit...** button to open the editor.

Adding Comments

It is a good practice to comment your programs, so we'll begin with a header that explains the purpose of the program and includes a version number. You start each comment line with a double-slash, and then type whatever text you wish. Comments are ignored by the compiler, so they don't affect the execution or performance of the program.



Another way to create comments is to type just the desired text, select it, and click the **Comment** button in the ribbon's **Editing** group. The Comment tool places double-slashes at the start of every line in the selection. It is typically used to comment-out code during debugging.



Notice that the same group includes an Uncomment button. This tool removes the double-slash at the start of every line in the selection. Again, this is primarily used in debugging.

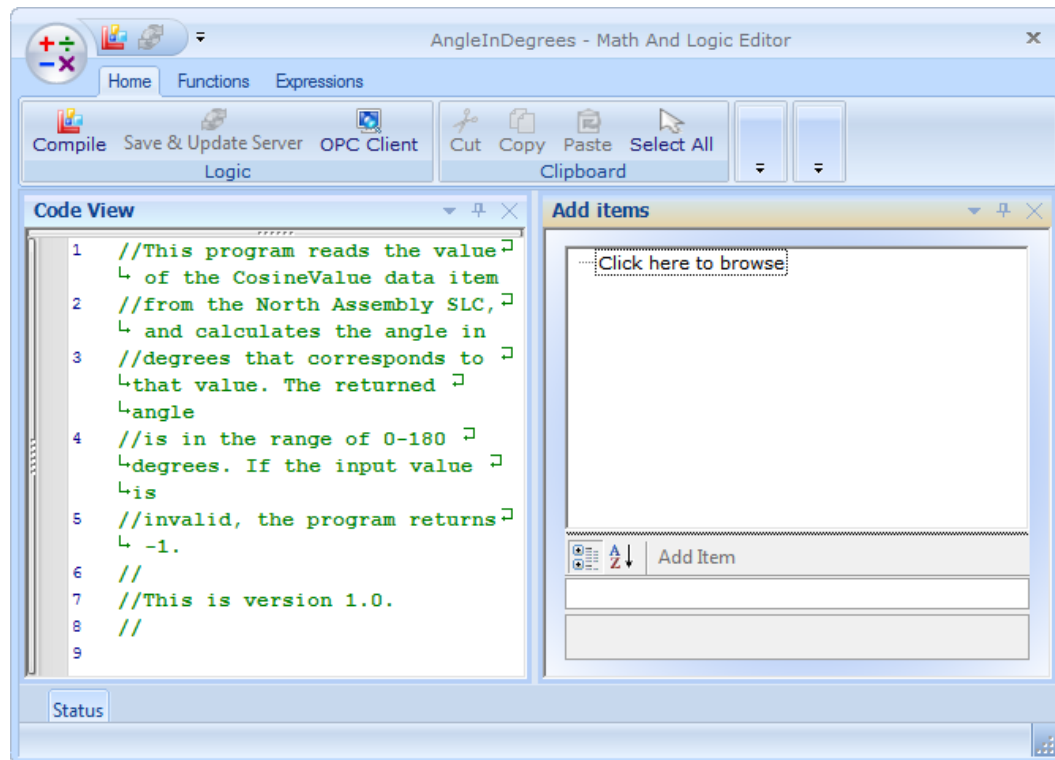
The Editing group also includes buttons for Undo, Redo, Find and Replace, which provide the functions common to most editors. The Find button can search for any text you enter, and can also go to a line number you specify.

Declaring Variables

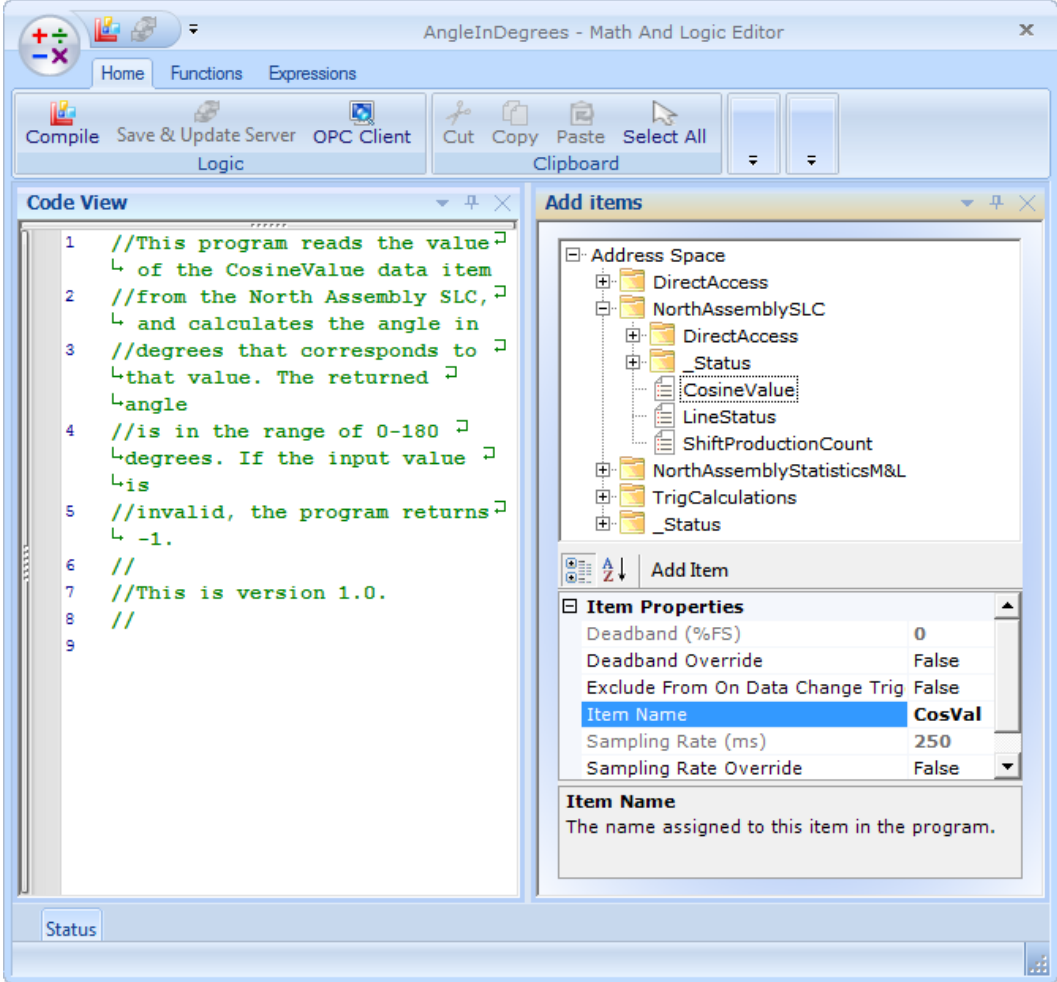
You must declare each variable in your program before you can use it. You can do this by simply typing a declaration statement, but the logic editor provides tools to make the job easier. The first of these that we will look at is the Add Items window.

Using the Add Items Window

The Add Items window helps you to declare ITEM variables that take the value of data items in the OPC server's Address Space.

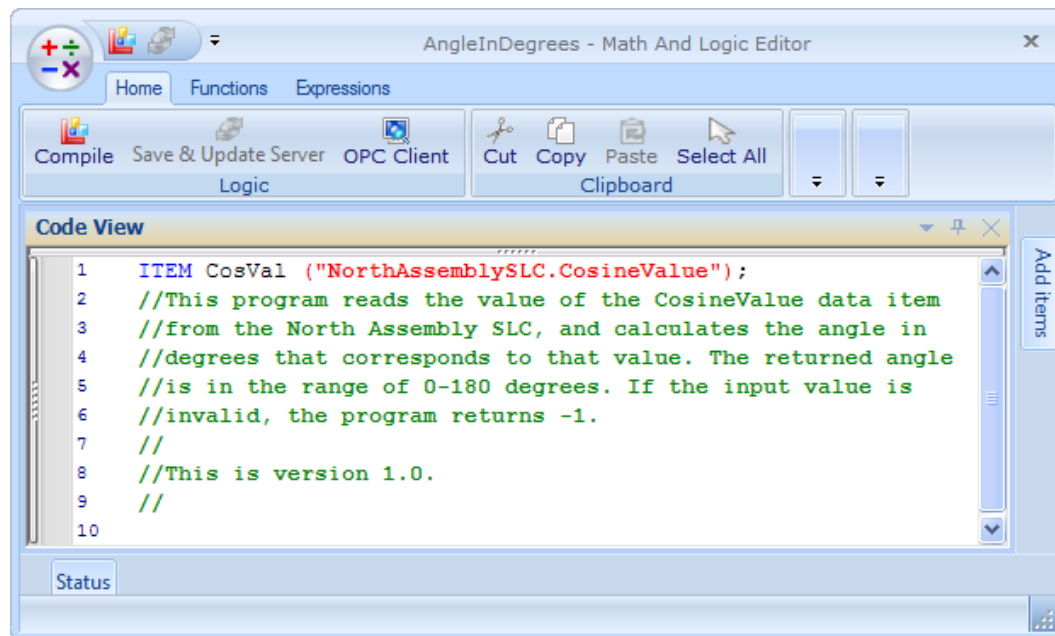


Click the **Add items** tab at the right side of the editing window to open the Add items window. You can then click in the **Add items** window to browse the OPC server for the data item you want to add.

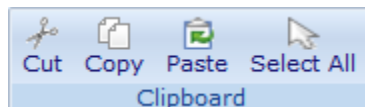


Here we have selected the **CosineValue** register in the **NorthAssemblySLC** device. After you select it, you can give it a name that the Math & Logic program will use, **CosVal** in this case. You can also modify other properties, as shown in the editing pane.

When you have named the item and finished any other edits, click **Add Item**, just above the editing pane.

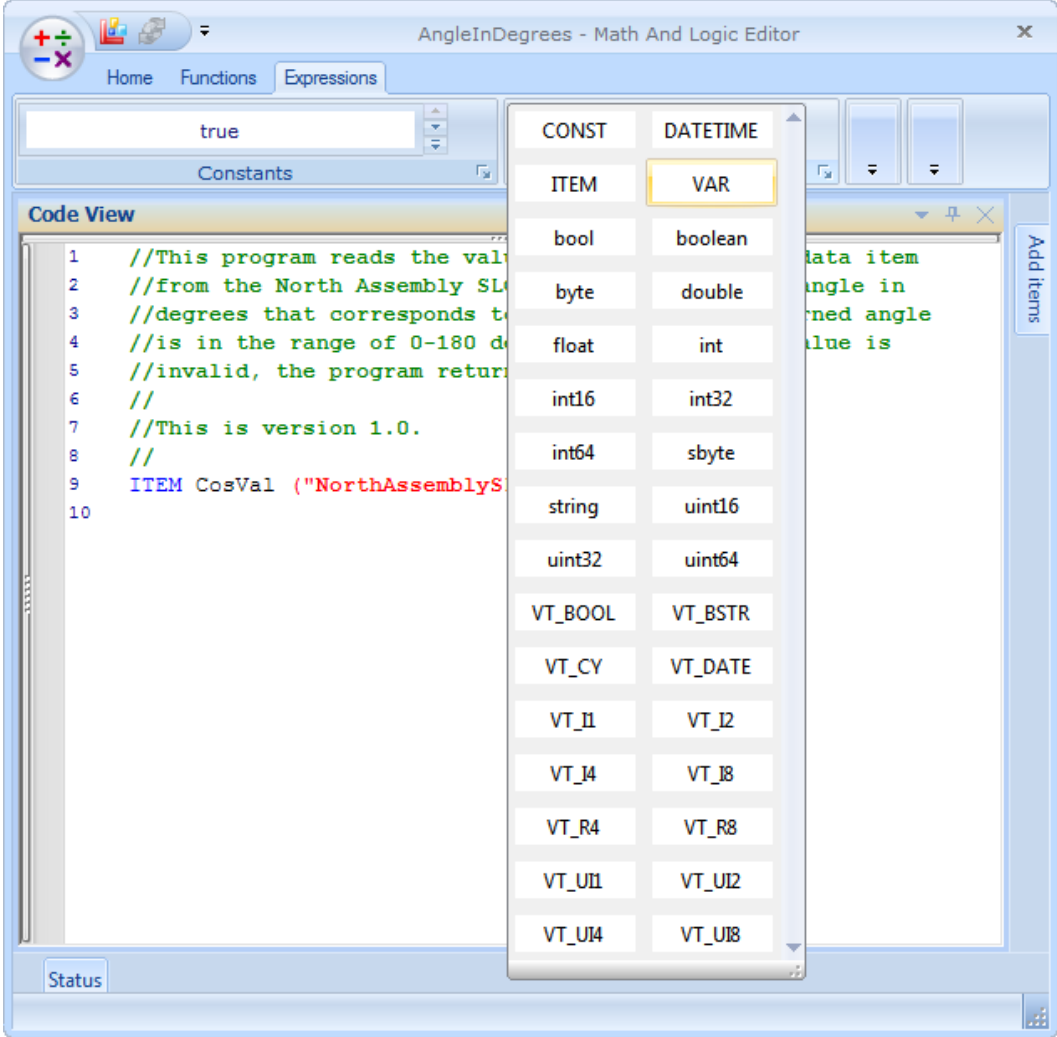



The editor always places the new item at the top of the program. It does this because all variable declarations must come before any executable statements. However, in this case you will want the declaration to come after the heading comments.

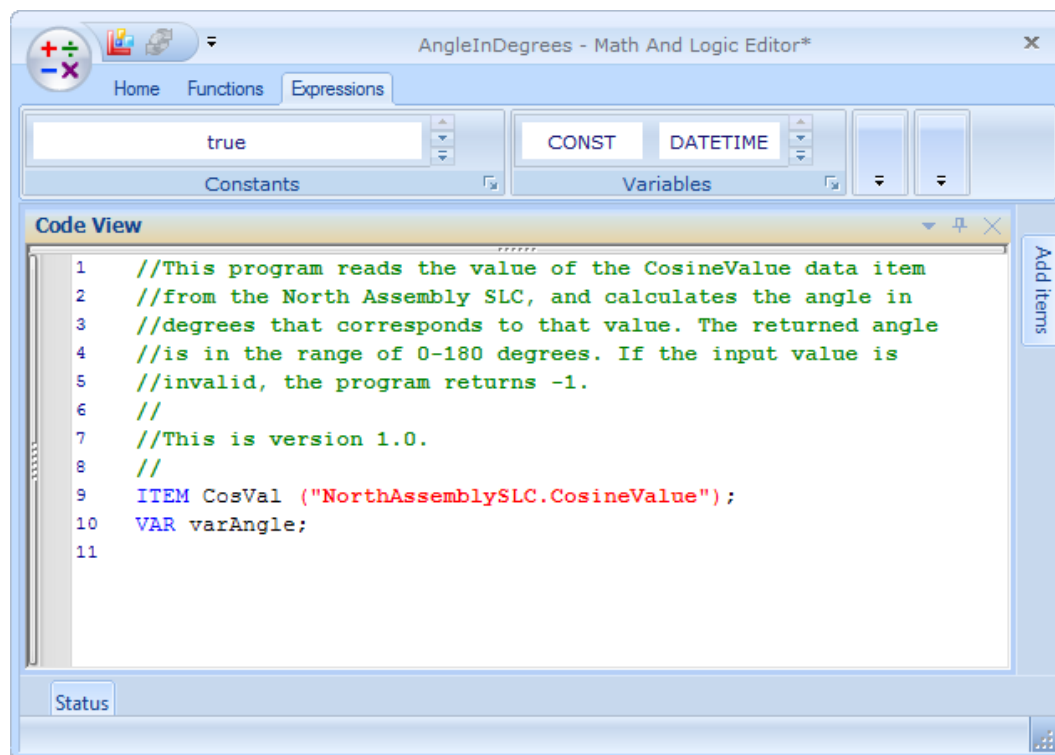


Use the tools in the Clipboard group to cut the item declaration and paste it below the header. You can also use the standard keyboard shortcuts, Ctl-X (cut), Ctl-C (copy), Ctl-V (paste) and Ctl-A (select all).

Next, we need a local variable to hold the value of the calculated angle.



Go to the **Variables** group in the **Expressions** ribbon. There, you can click the  button to access a list of the available types that you can declare. In this case, we'll choose **VAR** to create an untyped variable.




The editor inserts the declaration expression, and you need only type in the name. As with all C-Logic expressions, you can simply type the declaration yourself, if you wish.

Getting Help

The editor includes an extensive help system, and there are several ways you can access it.



First, you can click the button at the upper left corner of the editor window: . This will open a menu that includes a Help selection.

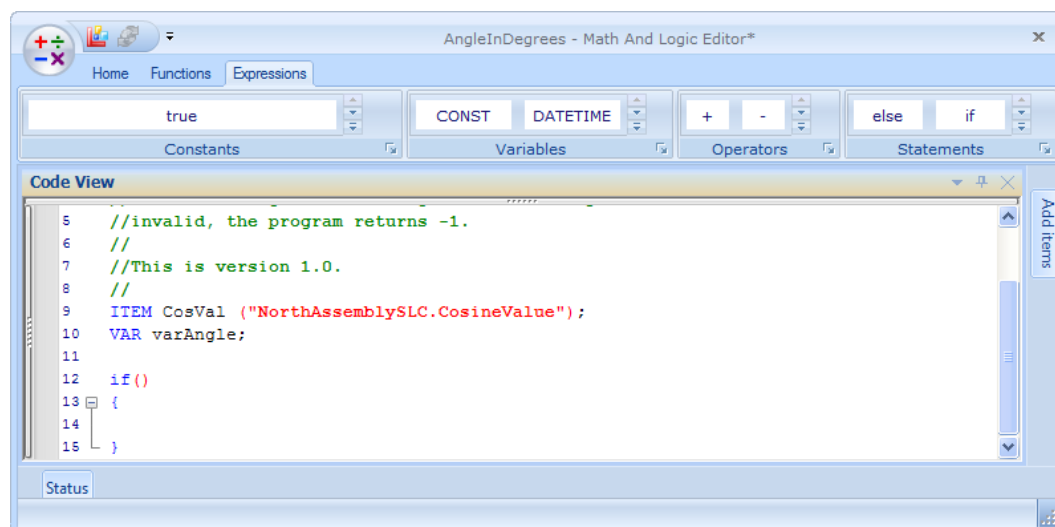
You can select any keyword in your program and press F1 to open the help file directly to the section for that function or expression.

<u>Function</u>	<u>Description</u>	<u>Example</u>
GetBitField	Returns a bit field value	x = GetBitField(y, 2, 3)
Sar	Arithmetic shift right	x=sar(y, 4)
Shl	Bit shift left	x=shl(y, 4)
Shr	Bit shift right	x=shr(y, 8)

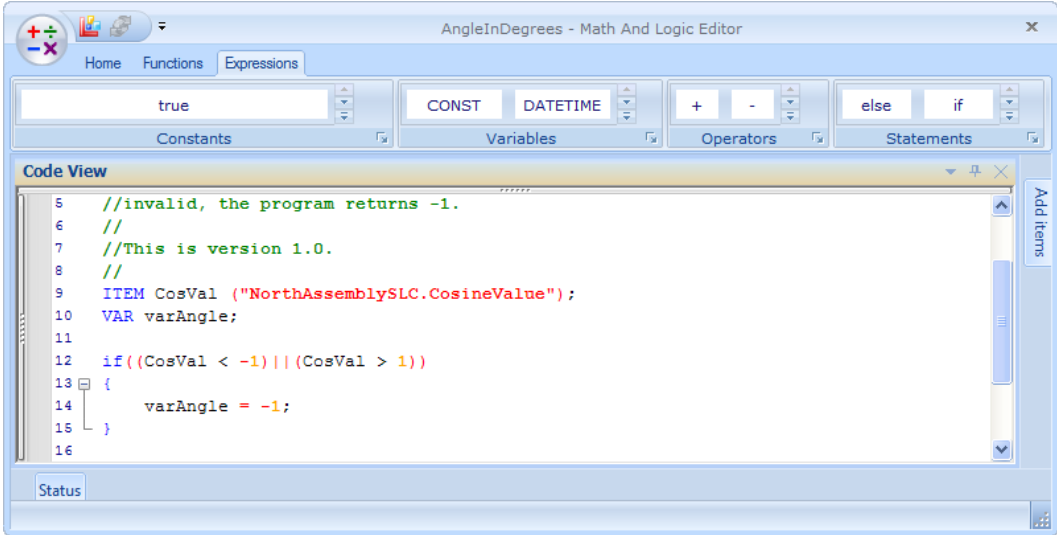
Yet another way is to click the dialog box launcher at the lower right corner of a ribbon group to get a description and example for each item in that group. The figure above shows the help screen for the Bitwise group on the Functions ribbon. Notice that the function names are hyperlinks. Click on the name of the function to go directly to a detailed explanation in the help file.

Statements, Operators and Functions

The first step in the program is to determine if CosVal is in the valid range for a cosine. To do this, we need an "if" statement.

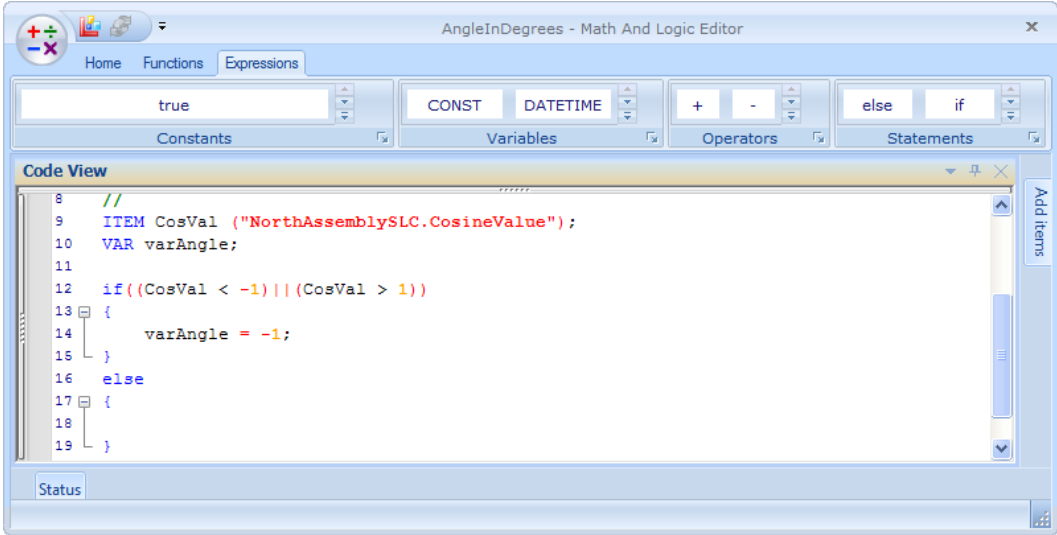


Open the **Expressions** ribbon and go to the **Statements** group. Click **if**. The editor will insert the "if" statement, and you must now write the condition and the code to be executed.

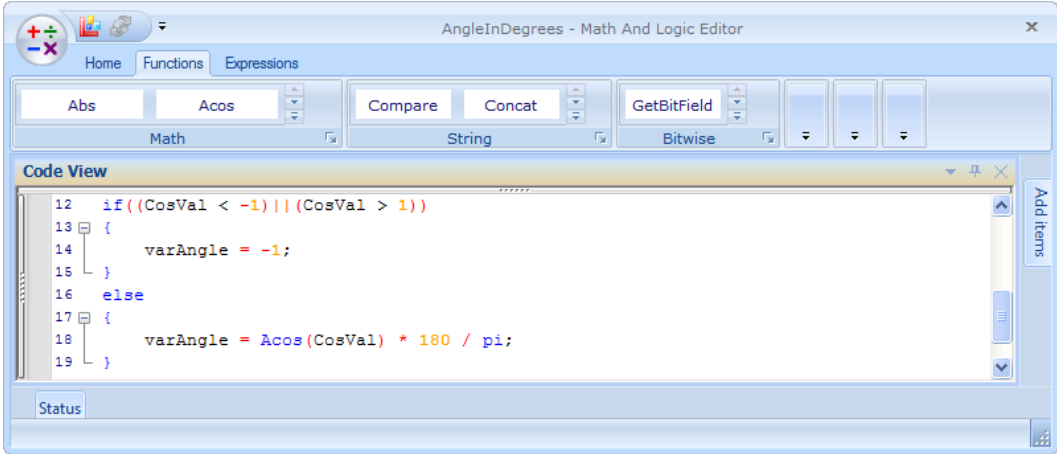


The easiest way is to simply type in the expressions. Open the **Operators** group to view the list of available math, logical and bitwise operators.

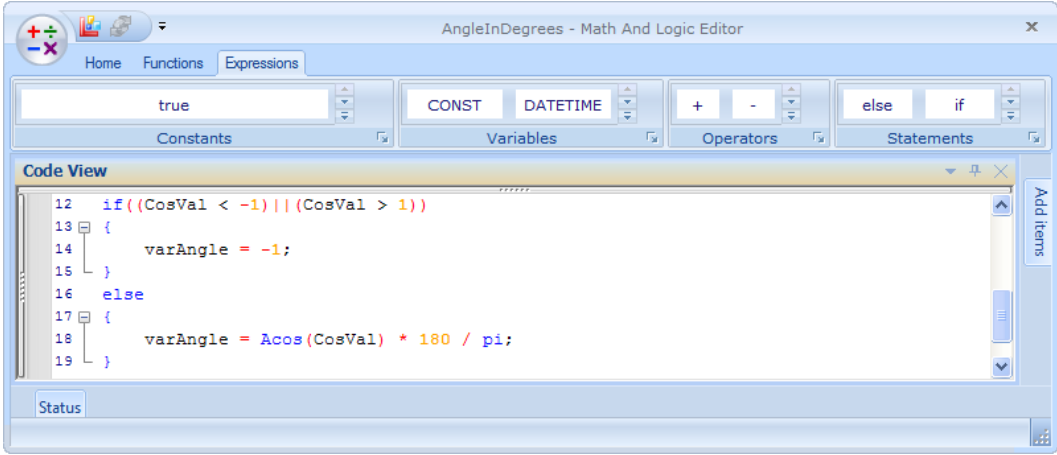
Here, the "if" statement checks to see if CosVal is less than -1 or greater than 1. If it is, then varAngle is set to -1, as required in the specification for the program.



Click **else** in the **Statements** group. You can now type in the procedure to follow if CosVal has a valid value.

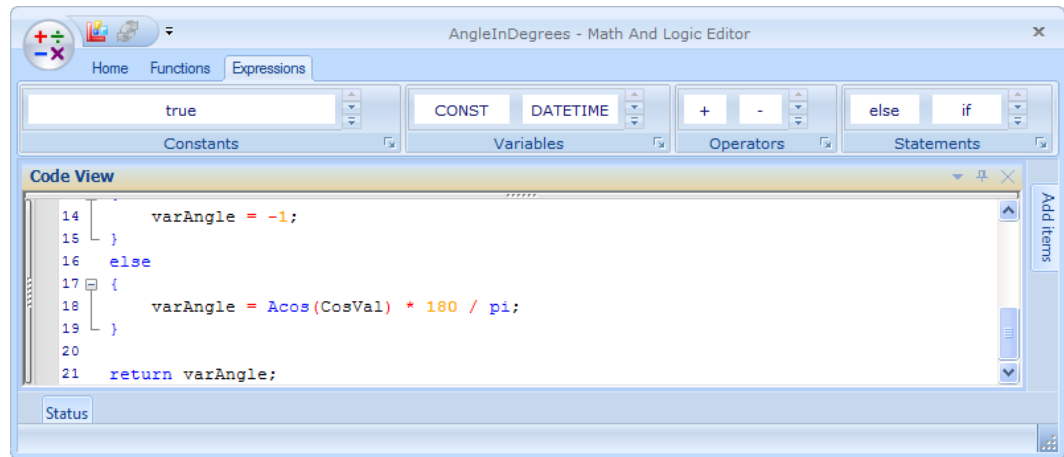


You can type in the expression to calculate the arc cosine of CosVal, placing the result in varAngle. Open the **Math** group on the **Functions** ribbon to find the **Acos** function that you'll need here.



Acos returns a value in radians, but the spec calls for degrees, so we must make the conversion. The value for pi is a pre-defined constant that you can enter by selecting it from the **Constants** group on the **Expressions** ribbon, or by simply typing "pi".

This completes the calculation of varAngle.



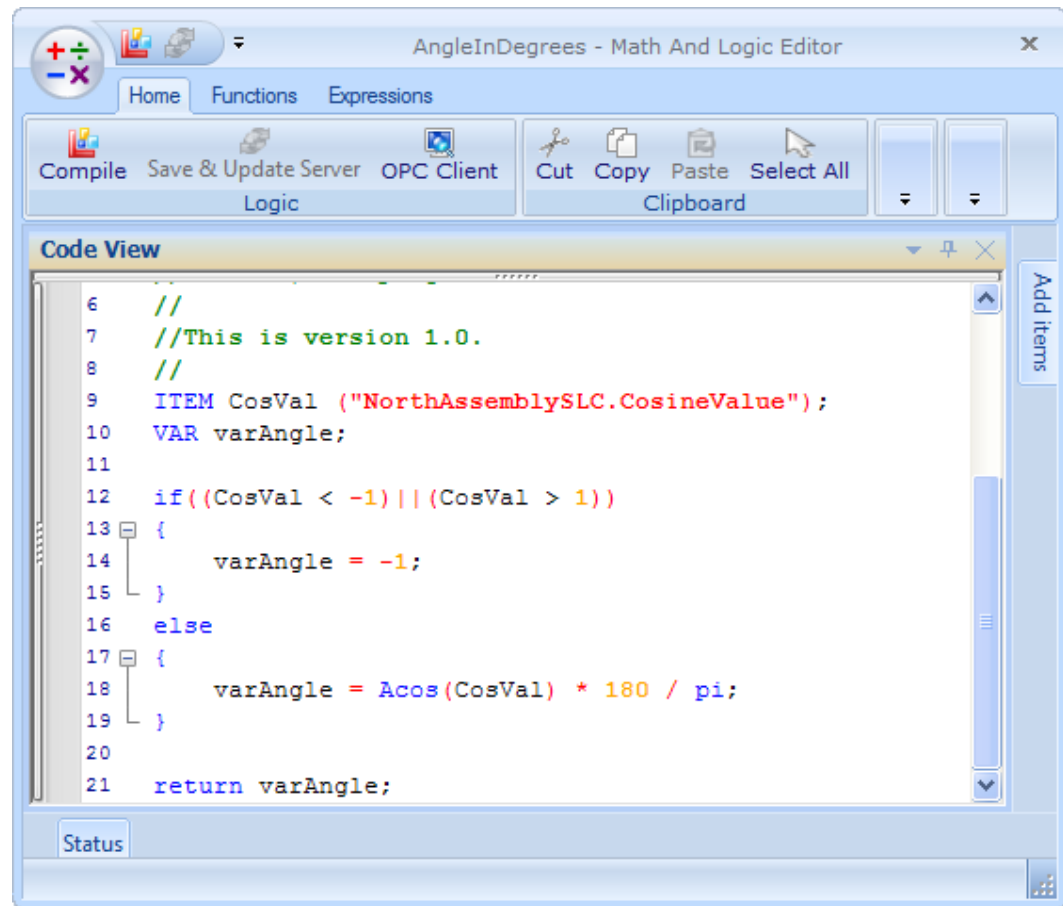
Go to the **Statements** group on the **Expressions** ribbon to select **return**. Include varAngle in the statement to specify that its value should be placed in the data item AngleInDegrees.

The program is now complete.

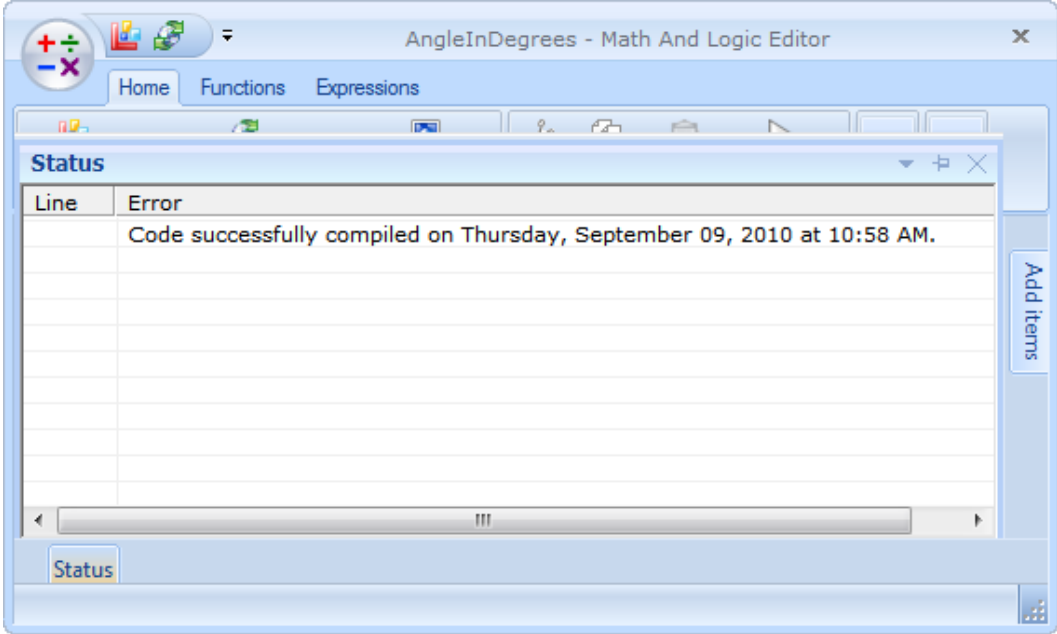
Compiling and Debugging

C-Logic is a compiled language, allowing it to execute much more efficiently than if it were interpreted at runtime. When you finish editing a program, you must compile it before the OPC server can execute it.

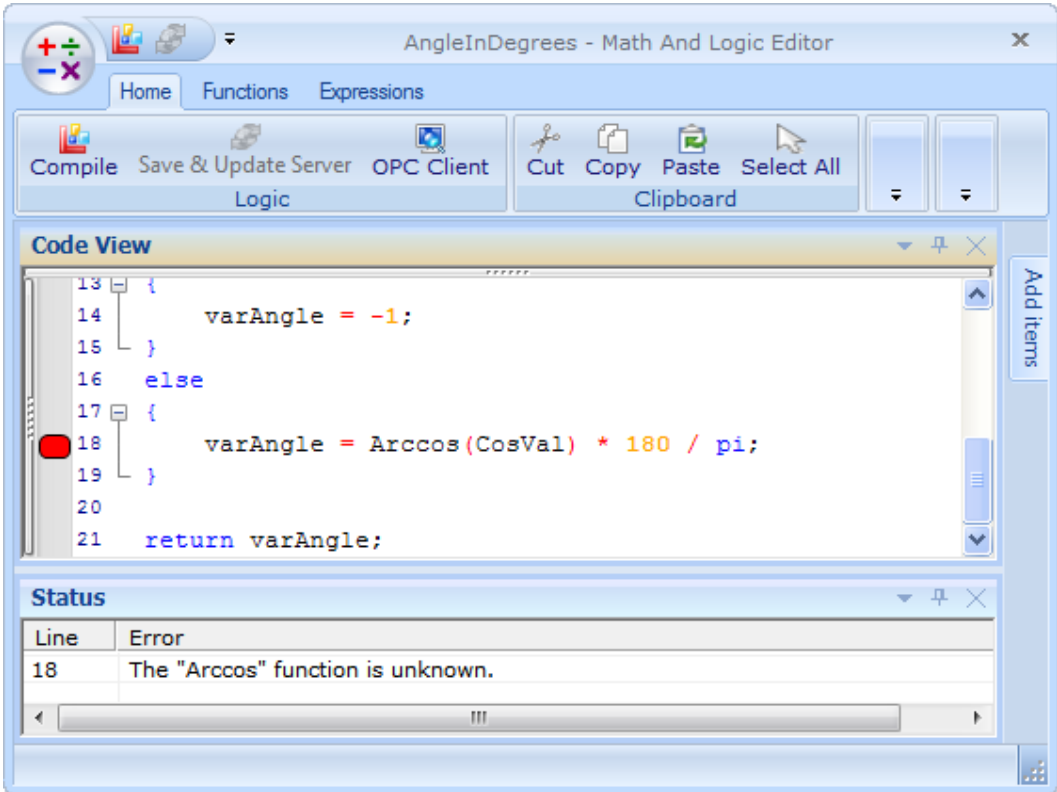
The use of a compiler has an additional benefit. The compiler can detect many types of syntax errors in the code and give you information on how to correct them. We'll look at these capabilities next.



To compile the program, go to the **Home** ribbon's **Logic** group and click **Compile**.



Open the **Status** tab at the bottom of the window to confirm that the program compiled properly. You can now close the C-Logic editor.



If the compile fails, the Status window will provide information on the failure.

In the example above, we've incorrectly entered the "Acos" function as "Arccos". The Status pane tells us that there is an error in line 18, and describes the problem that the

compiler detected. The red dot to the left of line 18 in the Code View pane helps you to locate the error quickly.



The Home ribbon's Logic group contains two additional buttons that will help you to debug your program. Click **Save & Update Server** to load your program into the running OPC server. Click **OPC Client** to open Cyberlogic's client application, allowing you to observe and test the program as it runs.

But first, you must specify when the program should run.

When Do the Programs Run?

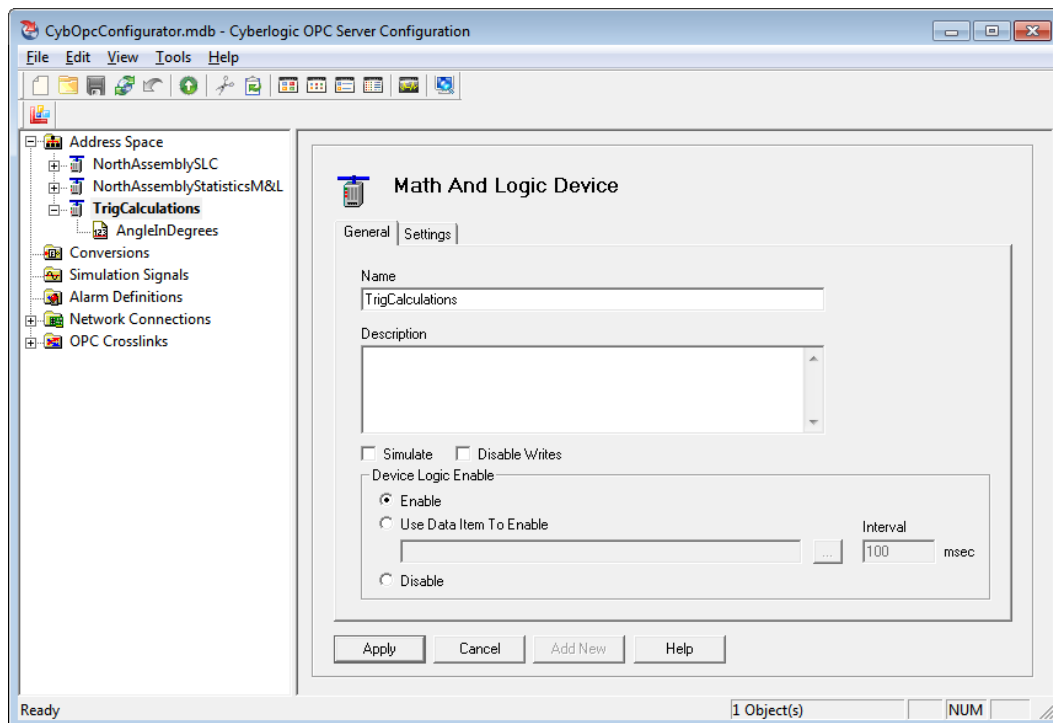
The programs run according to enable and run criteria you configure. The program must be enabled, and its trigger criteria must be met. You set these in the Math & Logic device. This allows you to group the programs that should run at the same time, so you can program their criteria in one place. However, these are default settings, which you can override for individual programs, if you wish.

For example, you might want to group programs that should run:

- Only at the beginning or end of a shift
- Only when you are producing a specific part type
- Only when the machine is down for maintenance
- Every minute, every machine cycle, or at some other fixed interval
- Whenever the controller detects a bad-quality part
- When the operator requests the programs to run

Enabling the Logic

Go to the Math & Logic device's **General** tab to set the enable criteria.

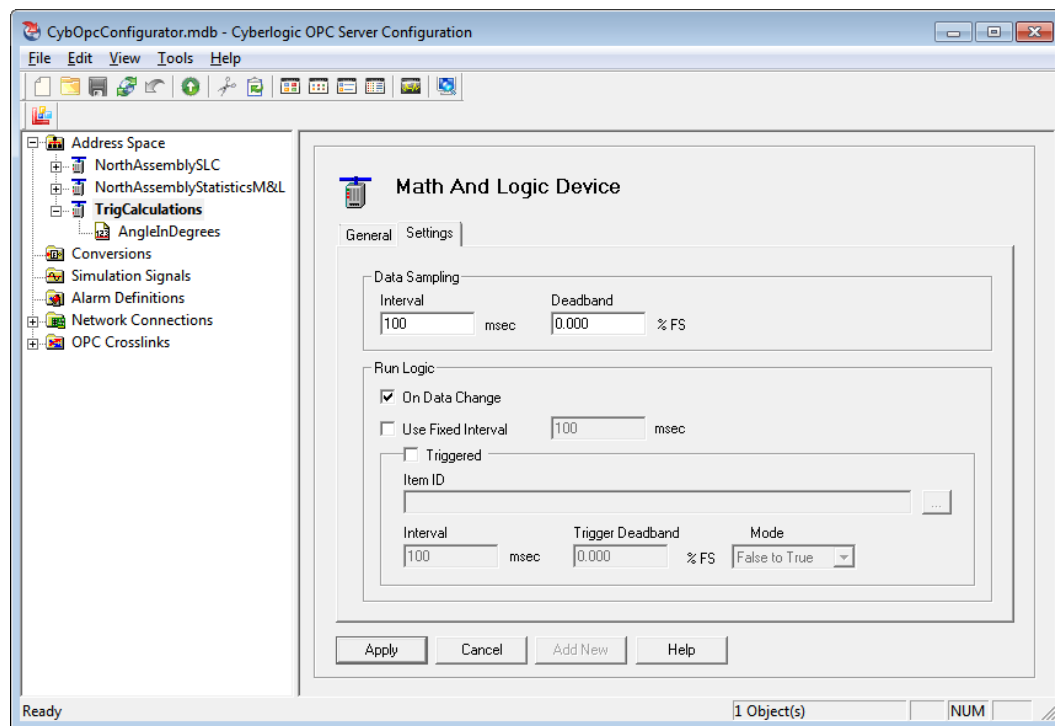


Pick one of the three selections in the **Device Logic Enable** section.

- **Enable** keeps the program always enabled. It will run whenever the trigger criteria are met.
- **Use Data Item To Enable** lets you specify a data item that will control whether or not the program is enabled. If the data item is true, the program will run when the trigger criteria are met. If the data item is false, the program will not run. You must also specify the interval at which the OPC server will check the enable data item.
- **Disable** prevents the program from running. You would typically use this setting during machine debugging.

Running the Logic

If the program is enabled, it will run when it is triggered. Go to the **Settings** tab to set the trigger criteria.

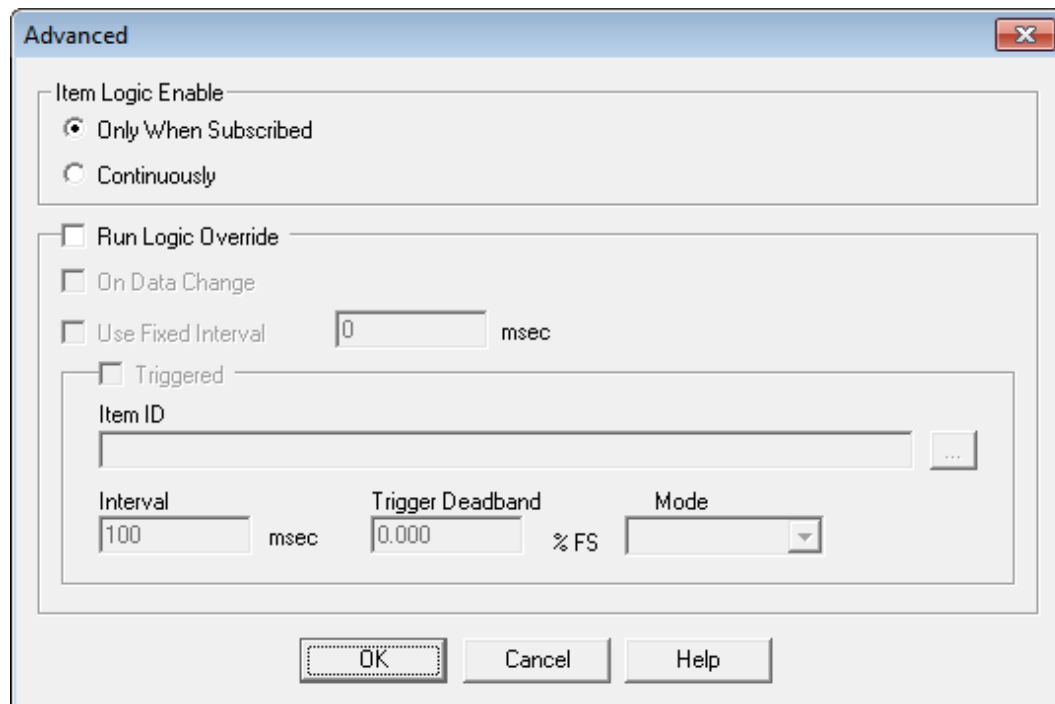


The Run Logic section is where you specify trigger criteria. There are three choices, and you may use any combination of them.

- **On Data Change** will run the program when any of the program's input data values change. If you select this option, go to the **Data Sampling** section to specify the default **Interval** that the OPC server will use to check for changes in the programs input values. You can also specify the default **Deadband** it should use.
- **Use Fixed Interval** will run the logic at an interval you specify.
- **Triggered** lets you indicate that the program should run when a specified data item changes value. You must choose the **Item ID** for the data item, an **Interval** that the OPC server will sample at, and a **Deadband**. You can also choose to trigger on a true-to-false change, false-to-true change, or any change.

Advanced Users: Creating Exceptions to the Rules

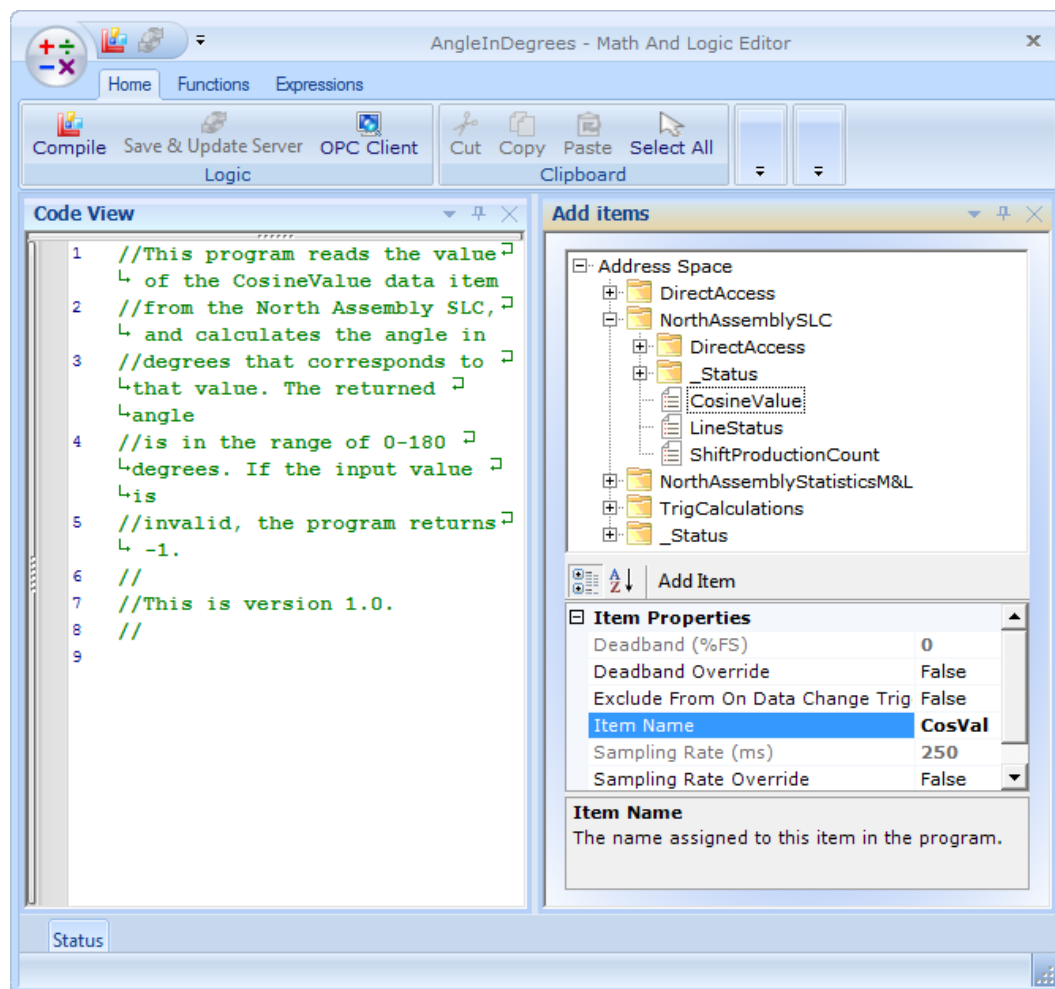
The configuration techniques we've already described will be sufficient for nearly all programming needs. However, to accommodate the unusual needs of advanced users, the editor provides two ways to create exceptions to the configured rules.



Go to the **Program** tab of a Math & Logic data item, and click the **Advanced...** button. The Advanced editing window will open. This window allows you to create special cases for the data item's enable and trigger logic.

In the Item Logic Enable group, you can choose to allow the logic to be enabled only when a client subscribes to the data item, or continuously, that is, even when no client has subscribed. In either case, the enable criteria set in the Math & Logic Device must still be met. The default is Only When Subscribed.

The Run Logic Override group allows you to configure a different set of criteria for when to run the program. The criteria here are configured in the same way as in the Math & Logic device.



Another place you can override some general settings is in the C-Logic program itself. When you add a data item declaration to your program, you can override the deadband and sampling rate. You can also override the deadband and sampling rate when declaring public variables. If your program uses On Data Change triggering, you can exclude the data item from consideration for that criterion. The same can be done in the public variable declaration. A program can also override the Fixed Interval at which the program executes by calling the SetFixedInterval function.

Getting More Information

The OPC server help files provide extensive information about programming, debugging and using the Math & Logic feature. In addition, Cyberlogic's sample configuration files provide many examples and programming suggestions to help get you started.

If you have any questions about OPC Math & Logic, please contact Cyberlogic's technical support group by emailing techsupport@cyberlogic.com or by calling 248-631-2288.

For quotations or other sales-related inquiries, contact Cyberlogic's sales department by emailing sales@cyberlogic.com or by calling 248-631-2200.