USER MANUAL

Accessory 55E
Profibus Option

Universal Field Bus Adapter Network (UNET)

3A0-603485-PUxx

October 23, 16 2003

DELTA TAU
Data Systems, Inc.
NEW IDEAS IN MOTION ...

Single Source Machine Control *Power // Flexibility // Ease of Use*
21314 Lassen Street Chatsworth, CA 91311 // Tel. (818) 998-2095 Fax. (818) 998-7807 // www.deltatau.com

Copyright Information

To report errors or inconsistencies, call or email:

Delta Tau Data Systems, Inc. Technical Support
Phone: (818) 717-5656
Fax: (818) 998-7807
Email: support@deltatau.com
Website: http://www.deltatau.com

Operating Conditions

All Delta Tau Data Systems, Inc. motion controller products, accessories, and amplifiers contain static sensitive components that can be damaged by incorrect handling. When installing or handling Delta Tau Data Systems, Inc. products, avoid contact with highly insulated materials. Only qualified personnel should be allowed to handle this equipment.

In the case of industrial applications, we expect our products to be protected from hazardous or conductive materials and/or environments that could cause harm to the controller by damaging components or causing electrical shorts. When our products are used in an industrial environment, install them into an industrial electrical cabinet or industrial PC to protect them from excessive or corrosive moisture, abnormal ambient temperatures, and conductive materials. If Delta Tau Data Systems, Inc. products are directly exposed to hazardous or conductive materials and/or environments, we cannot guarantee their operation.

Table of Contents

INTRODUCTION

This manual was written to provide setup procedures for Delta Tau's Accessory 55E for Profibus. There are two chapters: Profibus Slave for UMAC Turbo and Profibus Master for UMAC Turbo. Although Profibus has specific baud rates you can set, the data updates to the network from UMAC are done in a compiled PLC on the controller. Typically, this translates into data updates every 8-15ms. This update time is dependent on the size of PLCs in the system.

The code written for the Accessory 55E requires turbo firmware 1.938 or higher and a version of PewinPro with Service Pack 3 or higher.

PROFIBUS SLAVE SETUP FOR UMAC TURBO

Following is a step-by-step setup procedure to configure Delta Tau's Accessory 55E to a UMAC Turbo system over the 3U bus. Delta Tau's Profibus slave module is very flexible; it has the auto baud rate detect feature, and can transfer large amounts of data for a single module. The module is capable of transferring data up to the Profibus maximum of 12Mbits/sec. The 55E setup for a Profibus slave can transfer 244 input bytes, 244 outputs, or a combined total not to exceed 400 total bytes. The Slave also has a maximum of 32 data modules that it can transfer. These specifications will be important when it comes time to configure your master and establish data transfer across the Profibus network.

To proceed in this chapter make sure you have all of the following files:

- ***UmacProfibusSlave.h*** — A header file that will include all of the Setup parameters for configuring the 55E to the Profibus Network. You will download this file to the controller to configure the system for the Accessory 55E data transfers.
- ***ProfibusSlaveServiceHeader.h*** — A file that contains definitions to support ProfibusSlaveServiceEvent.plc. Do not modify this file for any reason.
- ***ProfibusSlaveMvars.h*** — A file that contains suggested m-variables for the inputs and outputs in the system. The inputs to the Network start at M6000 and the outputs to the Network start at M7000. This file can be modified to change the m-variable numbers that are used, but the definitions may not be changed.
- ***ProfibusSlaveServiceEvent.plc*** — A file that contains the UMAC PLC code that services the Profibus Network. Do not modify this file.
- ***Ssti0870.gsd*** — A generic GSD file for the 55E/UCS Profibus Slave. It aids in the configuration of the slave to the master's scan list and minimizes configuration error at an early stage.

All of the above files are located on the Delta Tau Web Site in the Downloads section under UNET Software. If not, contact Delta Tau Technical Support.

Now that we have the files we need to finish the Profibus Slave Setup, let us outline the general procedure for configuring the slave:

- UNET Hardware and Address Configuration
- UMAC software parameter setup
- Installing the UNET parameters and UMAC software
- Putting the UNET card on and offline
- Interfacing the UNET slave to the master on the Profibus Network

UNET Hardware and Address Configuration

Configure the hardware settings of the 55E card through the switch on the top of the card labeled SW1. Make sure the address chosen does not conflict with any other I/O card address in the system. This address is important to the customization of the ***UmacProfibusSlave.h*** header file. See Address chart below:

Table 1. Accessory 55E SW1 Address Settings

	CS10 SW1-1 ON SW1-2 ON	CS12 SW1-1 OFF SW1-2 ON	CS14 SW1-1 ON SW1-2 OFF	CS16 SW1-1 OFF SW1-2 OFF	Definition
SW1-3 ON SW1-4 ON	$078C00	$078D00	$078E00	$078F00	UCS_DATAREG_LO
	$078C01	$078D01	$078E01	$078F01	UCS_DATAREG_HI
	$078C02	$078D02	$078E02	$078F02	UCS_DATAREG_ADDR
	$078C03	$078D03	$078E03	$078F03	UCS_DATAREG_STAT
SW1-3 OFF SW1-4 ON	$079C00	$079D00	$079E00	$079F00	UCS_DATAREG_LO
	$079C01	$079D01	$079E01	$079F01	UCS_DATAREG_HI
	$079C02	$079D02	$079E02	$079F02	UCS_DATAREG_ADDR
	$079C03	$079D03	$079E03	$079F03	UCS_DATAREG_STAT
SW1-3 ON SW1-4 OFF	$07AC00	$07AD00	$07AE00	$07AF00	UCS_DATAREG_LO
	$07AC01	$07AD01	$07AE01	$07AF01	UCS_DATAREG_HI
	$07AC02	$07AD02	$07AE02	$07AF02	UCS_DATAREG_ADDR
	$07AC03	$07AD03	$07AE03	$07AF03	UCS_DATAREG_STAT
SW1-3 OFF SW1-4 OFF	$07BC00	$07BD00	$07BE00	$07BF00	UCS_DATAREG_LO
	$07BC01	$07BD01	$07BE01	$07BF01	UCS_DATAREG_HI
	$07BC02	$07BD02	$07BE02	$07BF02	UCS_DATAREG_ADDR
	$07BC03	$07BD03	$07BE03	$07BF03	UCS_DATAREG_STAT

Note: SW1-5 and SW1-6 must be set to ON

UMAC Software Parameter Setup

Variable and Memory Usage

Next, edit ***UmacProfibusSlave.h*** to customize the UNET slave for your particular application. The main items to be set up include: node number, number of input bytes from the master, number of output bytes to the master, and the UNET card hardware address configured above. The UNET PLC utilizes the following variables, memory addresses, and PLCs for the servicing of the Profibus Network communications and data transfer:

- **Q-variables for CS1**

Q7000..7999	**Always used for data storage (used in PLCC for data transfers)**

- **M-variables**

M4000..5999	Used by default for Network Data Access (see ProfibusSlaveMvars.h) You can edit the file **ProfibusSlaveMvars.h** to eliminate unnecessary m-variable definitions and change the numbering scheme as well.
M8000..8031	**Always used for logic in ProfibusSlaveServiceEvent.plc**
M8050	Always used for Network Commands

- **L-variables**

L7998..7999	Always used for pointers to Network Data memory locations
L8000..8002	Always used for PLCC pointers to UCS memory
L8025..8027	Always used for PLCC logic
L8050	Always used for PLCC Network Commands

- **Scratch Memory Locations**

$10F0-$10FF	These registers are open user memory set to zero at power up. These registers are always used.

Make sure that your UMAC application is not also using these same variables or memory locations. Serious problems may occur due to the overlapping of memory usage if you do not assure proper variable assignments.

Note

The UNET PLC code utilizes the scratch memory locations at $10F0-$10FF. If additional scratch memory is needed, define a User Buffer to utilize a portion of user memory.

The entire header file titled ***UmacProfibusSlave.h*** is located at Appendix A for reference. The following discusses the portions of the file that must be configured specifically for your application.

Specifying a PLC Number

The UNET Service PLC is a PMAC PLC. You must define it by number. We do not recommend that it be defined as PLC 1. The first block of the template defines which number you are using for this service PLC. The default is PLC13, and probably it will not need to be changed. If you have more than one UNET interface card in your system, each card will require a separate PLC and therefore a different PLC_NUMBER. Upon downloading, a separate PLCC program will be utilized during the data transfer operation. The PLCC # will be the same number that is entered into the header file below.

```
//CHOOSE A PLC AND PLCC NUMBER THAT YOU CAN USE IN YOUR SYSTEM
#DEFINE         PLC_NUMBER              13
```

Specifying the Characteristics of the Module

You must tell the Service PLC the slaves' node number.

```
#DEFINE        UCS_NODE_NUMBER    5        ;MACID(PROFIBUS) OR STATION NUMBER(PFB)
```

This section defines the node as a slave with a network ID of 5. The actual number you select must not conflict with another node on the network. See the documentation for your network to determine the largest available number, and the numbers already assigned. We never recommend using a network ID of 0 or 1, or the last available ID available on the network. These are often reserved for special purposes.

Specifying the Number of Data Words and their Location

This section defines how many 8-bit data bytes the slave will produce and consume. Inputs are data bytes received by the UNET module from a master, outputs are data bytes to be sent to the master.

```
//INPUTS TO NETWORK FROM UCS
#DEFINE        SLAVE_INP_SIZE        16              ; Number of slave inputs in bytes

//OUTPUTS FROM NETWORK TO UCS
#DEFINE        SLAVE_OUT_SIZE        16              ; Number of slave outputs
```

The definitions in the default template define 16 input and 16 output 8-bit bytes that are mapped as pairs to P-Variables. As defined in ***ProfibusSlaveMvars.h***, the input data will start at M6000 and the output data will start at M7000. M6000 will contain the first two bytes of data received from the master, M6001, the next two, etc. Similarly, M7000 contains the first two bytes to be sent to the master, M7001 the next two, and so forth. Note that the architecture of the service PLC reserves extra M-variables for other UCS Networks, M6256-M6999 and M7256-M7999. You can use these for other reasons, but avoid doing so unless there is an absolute need. The less things that are changed the simpler configuring and troubleshooting becomes.

The configuration of these parameters is extremely important to your master device. You must be certain that:

- You understand how your master maps slave data to its internal memory space. It can be very confusing when the master and slave use different data sizes and names.
- You understand the limitations on numbers of slave inputs and outputs imposed by your master. This is especially true if your master is servicing multiple slaves. For example, your master may allow a maximum of 32 input bytes and 32 output bytes for all slaves. This may impact your final selection of a data provided by your UMAC. It is imperative that you carefully test and document the transfers before proceeding with development of detailed applications..

Specifying the Base Address for the UNET Card

When installing a UNET card you selected a base address using SW1 on the card. The address of the card must be specified in the UNET Service PLC template. This may require changing the four definitions shown here. These four definitions specify four contiguous addresses whose base address is $78D00. Table 1 provides the base addresses for other SW1 settings. The base addresses here are defined in the low 16 bits of the definitions located in ***ProfibusSlaveServiceHeader.h***.

```
//UMAC ADDRESSES BASED ON SW1 SETTINGS ON THE ACCESSORY 55E
#DEFINE UCS_DATALO_ADDR                 $78D00
#DEFINE UCS_DATAHI_ADDR         $78D01
#DEFINE UCS_ADDRESS_ADDR                $78D02
#DEFINE UCS_CONTROL_STATUS_ADDR $78D03
```

Installing the UNET Parameters and UMAC Software

After modifying the above parameters, there should be nothing else that needs to be changed in the UMAC program files. The next step is to download ***UmacProfibusSlave.h*** to the Delta Tau UMAC controller and enable it by setting i5=2 and issuing the online command "enable PLC 13". Since the PLC number is a variable in the ***UmacProfibusSlave.h*** you may have to change which PLC you enable on UMAC if you changed the setting from the default.

Notice that the last lines in the template are:

```
#include "ProfibusSlaveMvars.h"
#include "ProfibusSlaveServiceHeader.h"
#include "ProfibusSlaveServiceEvent.plc"
```

Do not download ***ProfibusSlaveServiceEvent.plc*** directly; download the template ***UmacProfibusSlave.h***. If you have Pewin32 version 2.36 or earlier, make sure that you have enabled the MACRO/PLCC option in the editor prior to downloading. If you experience problems during the download, check to make sure that you have enabled the MACRO/PLCC option and that you have not changed a macro name by mistake.

Putting the UNET Card On and Offline

Reset Command

Reset the UNET slave by setting m8050=1. This will force the PLC to reset the UNET interface card. During the reset the UCS Module Status Indicator LED will turn red and then green. If the status indicator is flashing green, this is acceptable. You can repeat the command as desired to verify the reset operation.

If the status indicator LED changes do not occur as described above, there are several possible problems:

- Your Service PLC was not downloaded or is not enabled. Check to verify these conditions.
- The base address specified for the UNET memory map (i.e., $78Dxx) does not match the setting of the card, or is in conflict with an existing device. Verify the setting of SW 1 and that the address is properly reflected in the template.
- If you changed the base addresses for the M variables in **ProfibusSlaveServiceHeader.h** (which you shouldn't have), your command variable is not properly set in M8050. Check and verify this condition.
- Make certain that changes in base M variable indices, CS timer, and scratch pad memory locations are consistent and do not interfere with other PLCs or motion programs you have written.

Online Command

After the green Status LED returns to a flashing green state, put the card online by setting m8050=2.

This should cause the UCS Network Status Indicator LED to become flashing or steady state green. M8050 will then be set to 4 to notify the user that Network is now transferring data. If the LED is off, flashing or steady state red, or has an amber color, verify that:

- Your network is properly configured, has power applied (DeviceNet), and is terminated. The best situation is one in which you have thoroughly configured and network using another SST network interface board, or have verified network operation using the tools supplied by your controller vendor (Allen Bradley, Siemens, et al.).
- The node number specified in the template ($0005 in the default template) is not in conflict with any other node on the network. If it is, make the appropriate change and download the PLC again.

- You have correctly specified a number of inputs and outputs that is consistent with the limitations of the network.
- You have selected the proper baud rate for the Network and it matches that of your master. There are no masters on the network attempting to connect to your UNET slave. You can verify this by disconnecting all other devices from the network.

Offline Command

During your verification of the module, you can take the card off-line by setting

M8050 = 3

You can repeatedly reset the card (M8050 = 1), go on line (M8050 = 2), and off-line (M8050 = 3) during testing and verification of your card, Service PLC, and network.

Advanced Testing of your Slave Configuration

Prior to the development of any PMAC motion or PLC programs to use the P-variable data, you should conduct further testing of your network and slave configuration. This will involve using the network tools you purchased from SST or those supplied with your master controller. To accomplish these tests you must:

1. Reset the UNET card by typing M8050 = 1
2. Place the UNET card on line by typing M8050 = 2
3. Verify that data transfers are set properly by checking that M8050 is equal to 4

The last step causes your UNET Service PLC to actually read and write the UCS module's memory map. In doing so, the data received from the network is copied into your Input variables (M6000 and up) and the data contained in your output variables (M7000 and up) is copied into UCS memory. If you do not enable transfers, an on-line UCS module may be functioning perfectly but you'd never know it because the data would not be accessible to the UMAC.

Interfacing the UNET Slave to the Master on the Profibus Network

Now that the slave is ready and online, it is time to configure the master on the Profibus Network. Master configuration software is highly vendor-specific and it is up to you to know and understand this procedure for setting up slaves on the network. Usually, the baud rate is configured within the master setup. The 55E-slave module will detect automatically the baud rate you choose. Many master systems will need to load the GSD file for the UNET Profibus slave. When you do this, you will tell the master what the slave node number is and what types of I/O modules it is going to pass. The modules need to be configured in such a manner that they correspond with how the slave sends the data. The UCS card firmware will pass data across the Profibus Network in 4 byte consistent modules when the data size (input bytes + output bytes) is less than 128 bytes (32 modules * 4 byte consistency). This is defined in ***UmacProfiSlave.h***. The word consistent is a definition in the Profibus world that translates into data that holds its integrity no matter when it is polled. This is because it is all transferred at once in 32 bit transfers. This protects against the case in which you query a chunk of data that is in the middle of a read and you get jumbled data back. On the master side of things you must define all of the master input modules first and then the master output modules or it will not communicate.

Since the Profibus slave is capable of transferring data sizes up to 400 bytes, the UCS firmware adjusts the size of data per module when the defined amount is greater than 128 bytes. This is based on the definition setup in the header file ***UmacProfiSlave.h***. If the data size is greater than 128 bytes (input bytes + output bytes) then the UCS processor will send the data in a form that is not consistent and 16 bytes wide. The data is not considered consistent because it can only assure consistency within 4 bytes. This consistency still holds for every 4 bytes even though the module structure has changed. It is important to note that the data is sent in 16-byte non-consistent modules. If the data size for either inputs

or outputs is not a factor of 16, then the last module passed of the inputs and/or the outputs will be however many bytes are left over after passing as many 16-byte not-consistent modules as possible. On the master side of things you must define all of the master input modules first and then the master output modules, or the slave will not communicate.

PROFIBUS MASTER SETUP FOR UMAC TURBO

Following is a setup procedure to configure Delta Tau's Accessory 55E as a Profibus Master to a UMAC Turbo system over the 3U bus. Delta Tau's Profibus master module is very flexible; it can be configured to any Profibus baud rate up to 12Mbits/sec, and can transfer large amounts of data for a single module. The 55E setup for a Profibus master can transfer 244 input bytes, 244 outputs, for a combined total of 488 bytes per slave. The Profibus Master can also handle a total of 125 slaves on the network. These specifications are important for configuring your master and establishing data transfer across the Profibus network.

Overview of the Different Configuration Methods

There are two different methods for configuring the network setup. The difference between them lies mainly in the configuration of the slave node properties. The methods are –

- Using the ***SST Scanner Module 5136-PFB-PCM-ST*** and the Delta Tau UNET Configuration Host. You may configure the system by connecting to the network and allowing the scanner card to pull information from the slaves on the network. This method involves the purchase of a scanner card from SST, which costs about $1800. It is a nice luxury to be able to scan the network and test the slave hardware individually, and worth the cost if you are planning on doing a lot of work with Profibus in the future. It is not necessary to purchase one to configure your system.
- Using EDS files supplied from a given manufacturer. This method does not require the purchase of any additional hardware, and provides less data entry than the manual method mentioned below.

Necessary Components

Delta Tau Hardware

- ***Accessory 55E with Profibus Master option*** -3U accessory card that will serve as the master on the Profibus Network.

Download Hardware

You must have one of the following:

• ***SST Download adapter UCS-DA-1 (INTL)***	Download adapter necessary for downloading personality files to the master UCS modules.
• ***Accessory 55E On-board Serial Port***	On-board hardware to supply a communication port to the SST UCS card from the PC configuration software (Lucien-does not exist yet but is coming).

Software

• ***Delta Tau UNET Configuration Host***	Application software necessary to generate specific personality files which contain the scan list for the master UCS module. This application also generates a file that includes address tags for each set of inputs and outputs on the Profibus Network.
• ***UmacProfibusMaster.h***	A header file that includes all of the setup parameters for configuring the 55E to the Profibus Network.
• ***ProfibusMasterServiceHeader.h***	A file that contains definitions to support *ProfibusMasterServiceEvent.plc*. Do not modify this filefor any reason.
• ***ProfibusMasterMvars.h***	A file that contains suggested m-variables for the inputs and outputs in the system. The inputs to the Network start at M6000 and the outputs to the Network start at M7000. This file can be modified to change the m-variable numbers that are used, but the definitions may not be changed.
• ***ProfibusMasterServiceEvent.plc***	A file that contains the UMAC PLC code that services the Profibus Network. This file should not be modified for any reason.

Optional Hardware Components

• ***SST Scanner Module 5136-PFB-PCM-ST***	SST scanner module that aids in troubleshooting and allows a scanning feature to be enabled within the Delta Tau UNET Configuration Host application.

If for some reason you do not have the Delta Tau UNET CD, all of the above files and software applications are located on the Delta Tau Web Site in the Downloads section under UNET Software. For information on purchasing hardware components contact Delta Tau. You may contact SST directly only for the ***SST Scanner Module 5136-PFB-PCM-ST*** and ***SST Download adapter UCS-DA-1 (INTL)***.

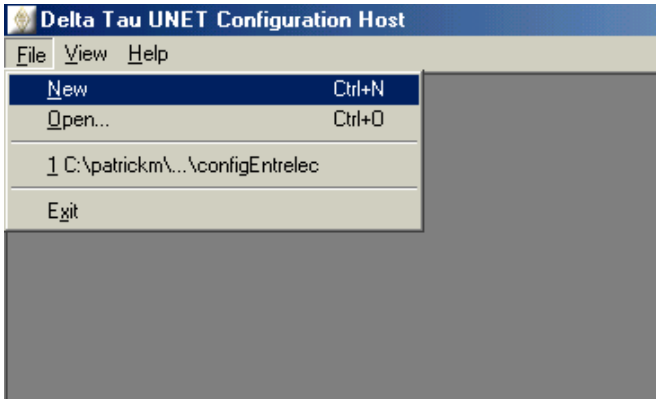
If you have purchased ***SST Scanner Module 5136-PFB-PCM-ST*** from SST then you received a CD with all of the software and drivers to support the various software tools needed in this section. After you install the SST software, insert the Delta Tau UNET CD and install the software needed to configure the system. All of the files needed for the setup are located in the Program Files/Delta Tau/UNET directory.

If you do not have the ***SST Scanner Module 5136-PFB-PCM-ST*** then load the UNET CD; all of the necessary files will be loaded into the Program Files/Delta Tau/UNET directory.

Using the Delta Tau Configuration Host to Set Up the Network Parameters

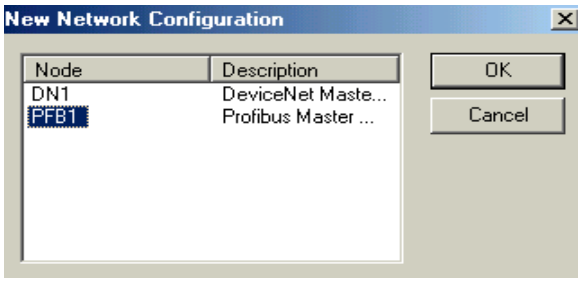
Configuring the Network Master Properties

Within the UNET directory, open the application titled “Delta Tau UNET Configuration Host”. Select ***New*** from the file menu. See the figure below:



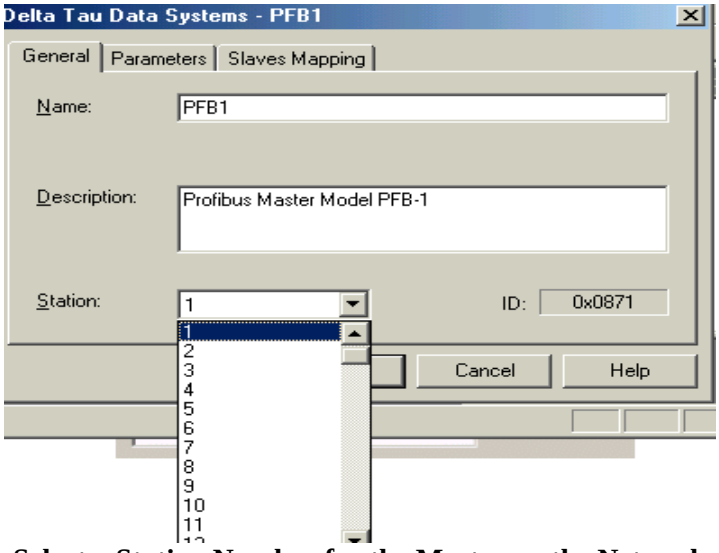
Opening New File from Configuration Host

When you select ***New***, a dialog box will open asking you whether you wish to create a master configuration for DeviceNet (**DN1**) or Profibus (**PFB1**). Choose ***PFB1*** for Profibus, as shown below.



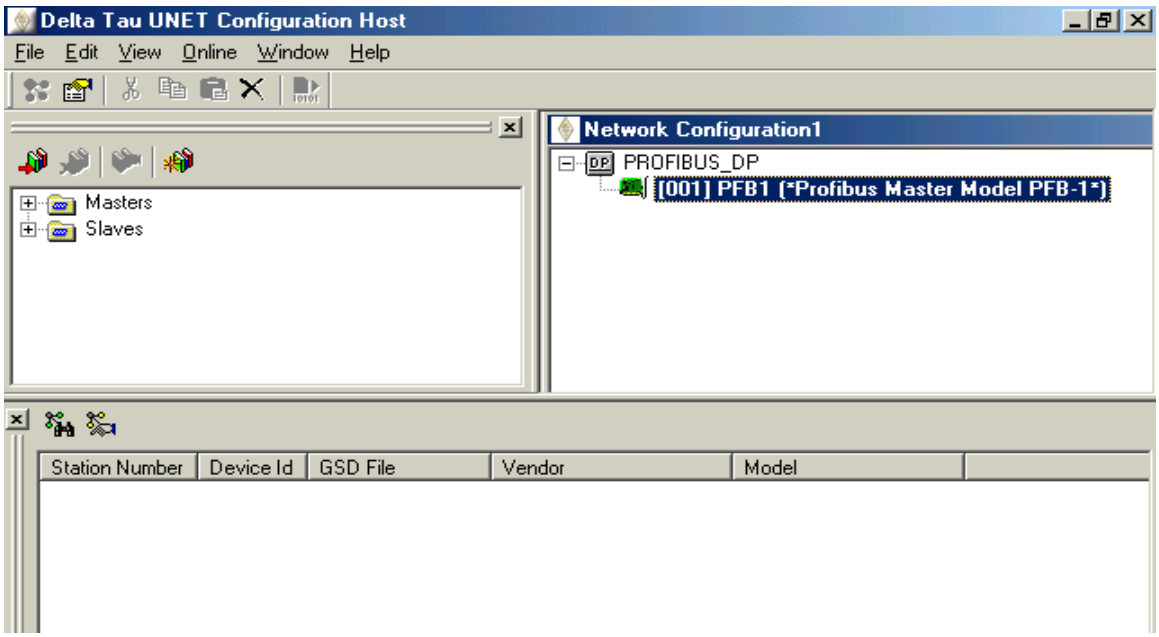
Selecting Profibus for the Configuration

After opening a new Profibus master configuration, your screen will appear similar to the one shown below.



Select a Station Number for the Master on the Network

The above screen is the master properties display, in which the main item to complete is the ***Station*** number. The ***Station*** number drop down menu selects the address for the master on the Profibus fieldbus. It is important to keep track of what node is addressed to what ***Station*** number. Make sure to choose a station number that will not interfere with another node on the Network. The ***Parameters*** and ***Slaves Mapping*** tabs usually do not need to be changed. After selecting a station number click ***OK*** and the application will appear as in the figure below.



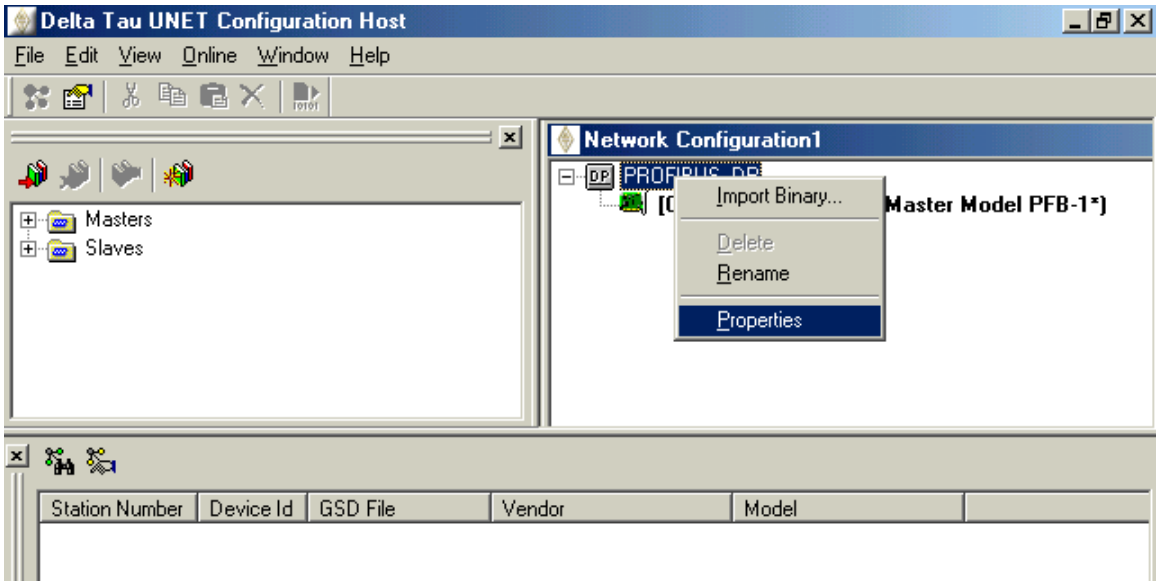
Main Form

The application is divided into 3 separate forms. The form in the upper right is titled Network Configuration1 (title will change once the application is saved) as is used to show all of the components within the network configuration. You can access the properties of each item within the configuration by pressing the right mouse button.

The form in the upper left is a list of the EDS files that are opened for the particular configuration. These are used more often when configuring a system manually, without the use of the SST Scanner Card. This is where you load the EDS file for a particular slave module that you are going to add to the configuration. When using the browse tool, the EDS files are uploaded from the particular slave online.

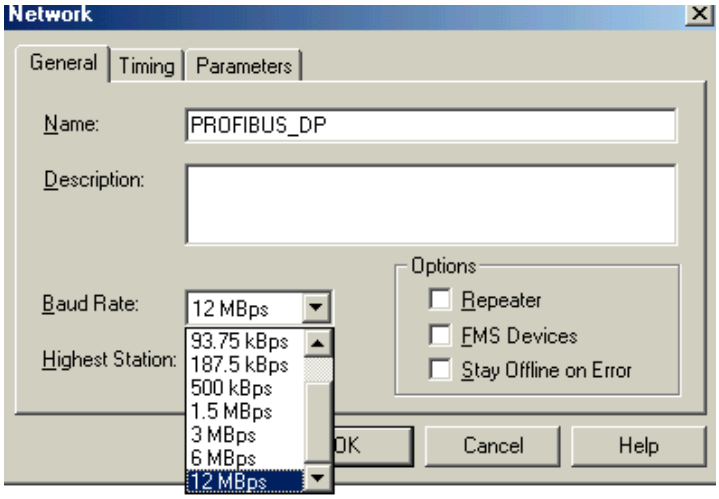
The form at the bottom of the application is a special form that is used with optional network browse feature. The SST scanner card (5136-PFB-PCM-ST) is required to utilize these features.

Delta Tau's UNET Configuration Host supplies a configuration tool that allows your master's properties to be tailored for a specific network configuration. The first in this process is to set up the master and network properties. The master you are configuring from the UNET Configuration Tool is not the same as the UCS master that you will be using in your final configuration. We are going to use this tool to simulate the network properties and then, based on those properties, download a personality file to the UCS master module with the proper network settings. To set up the network properties, right-click on the icon labeled ***PROFIBUS_DP*** and select the ***Properties*** option. See the figure below:



Choosing the Network Properties Option

The following network properties window will appear.



Configuring the Network Properties

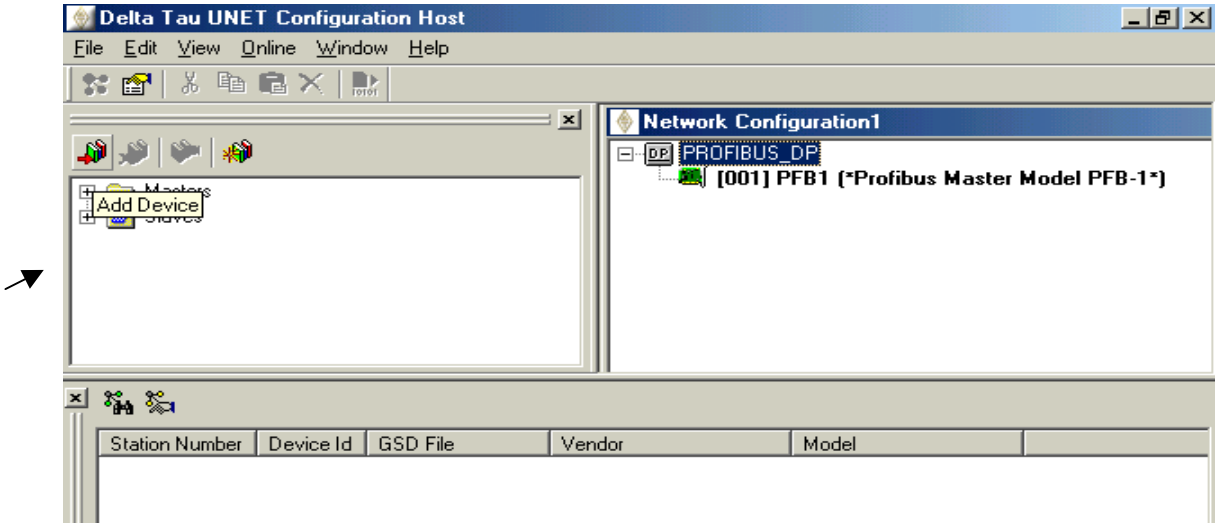
There are three tabs located on the form shown above: ***General***, ***Timing***, and ***Parameters***. Typically, the only tab that needs to be modified is the ***General*** tab. Under the ***General*** tab, the ***Name*** and ***Description*** text boxes may be changed for application customization if desired. The ***Baud Rate*** text box must be set appropriately for your network. It is important to choose a baud rate that all of your slave nodes can support on the network. The ***Highest Station*** typically does not need to be changed from default, which is the maximum per Profibus Specification.

Once you have selected the master and network properties, click ***OK*** and return to the "Main Form" screen. If you succeeded in setting up the master properties, proceed to the next section and configure the slave properties.

Using the Delta Tau UNET Configuration Host to Configure the Network Slaves Properties

Manually Setting up the Slave Properties for the Network Configuration by Using an EDS File Supplied By the Manufacturer of the Slave Node.

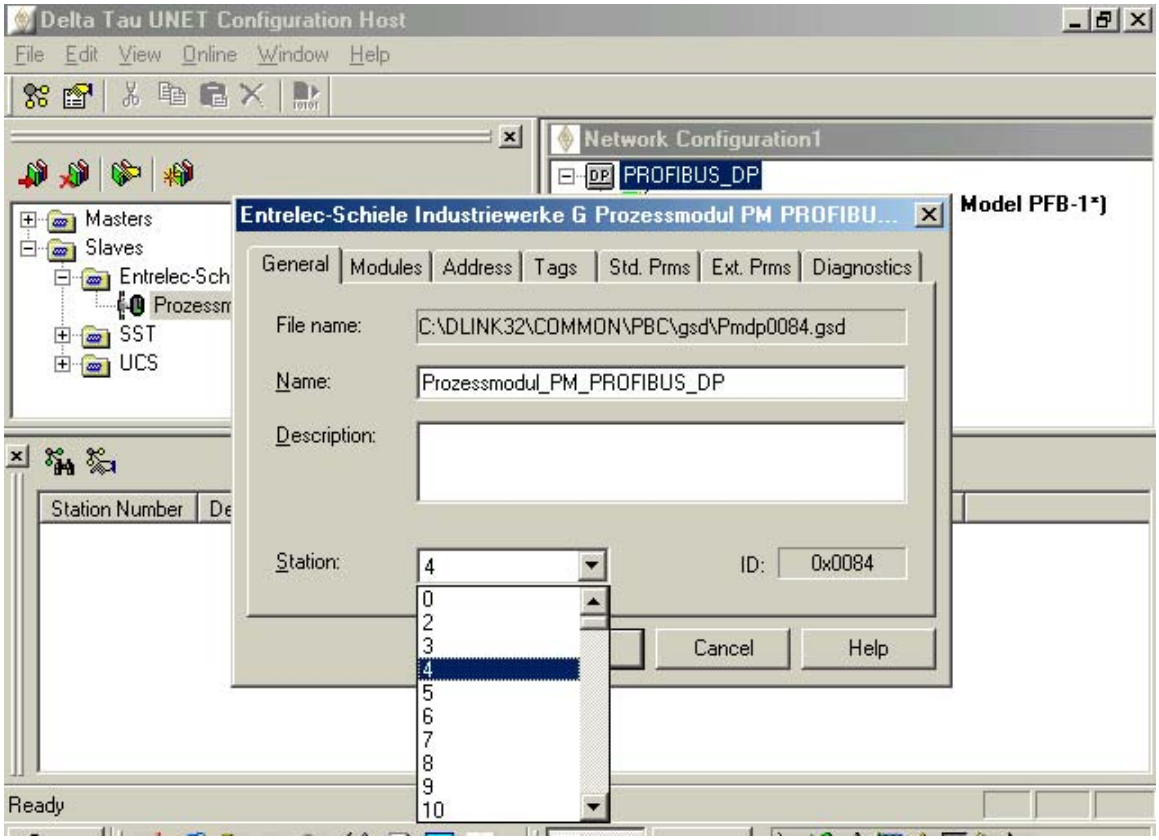
When manually setting up the nodes on the network, you can add a new node through the software, or load a specific node EDS file into the library. To add a new EDS file to the library, click on the icon above the EDS tree on the left side of the application as shown below:



Adding an EDS file to the Library

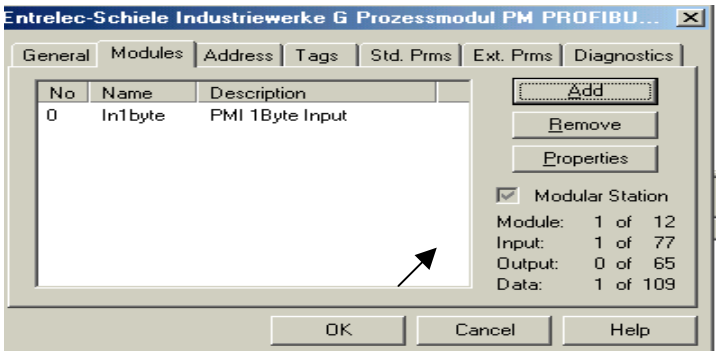
Once you have browsed your machine and added the EDS file to the library you should see the file located in the library. Next, drag the icon of the node's EDS file and place it under the master in the Network to the right. When you drop the EDS file under the master icon, the slave properties window appears automatically. There are 7 tabs within the slave properties window: ***General***, ***Modules***, ***Address***, ***Tags***, ***Standard Parameters***, ***Extra Parameters*** and ***Diagnostics***.

The key entry in the ***General*** tab is the ***Station*** number. Choose an open address on the Profibus network. Make sure not to use the same ***Station*** number as the master. The ***Name*** and ***Description*** text boxes are for you to add comments and descriptions for reference. They are not critical to the network configuration.



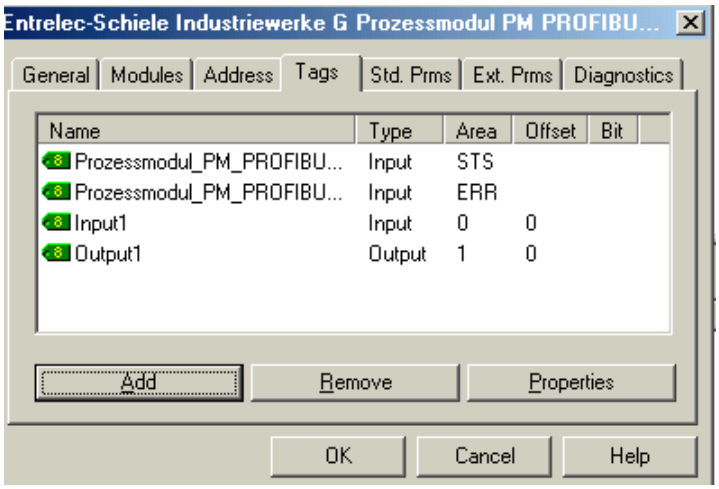
Setting the Properties of the Slave Node

The ***Modules*** tab is where you configure the size and number of modules for each slave on the network. The size and number of modules for a slave is specific to the particular node that you have chosen. When the ***Add*** button is pressed, the possible choices for sizes will be listed based on the particular nodes' EDS file. Select the appropriate size and number of modules for this node and move onto the next tab.



Identity Tab of the Slave Properties

The next tab located in the slave properties window is the ***Address*** tab. Usually, this tab does not need to be changed. The ***Tags*** tab does need to be configured for every network configuration. By setting up the tags we informed the application what data will be on the Network and then Configuration host can set up the proper addressing in the output tag file and the personality file (scan list) of the master.



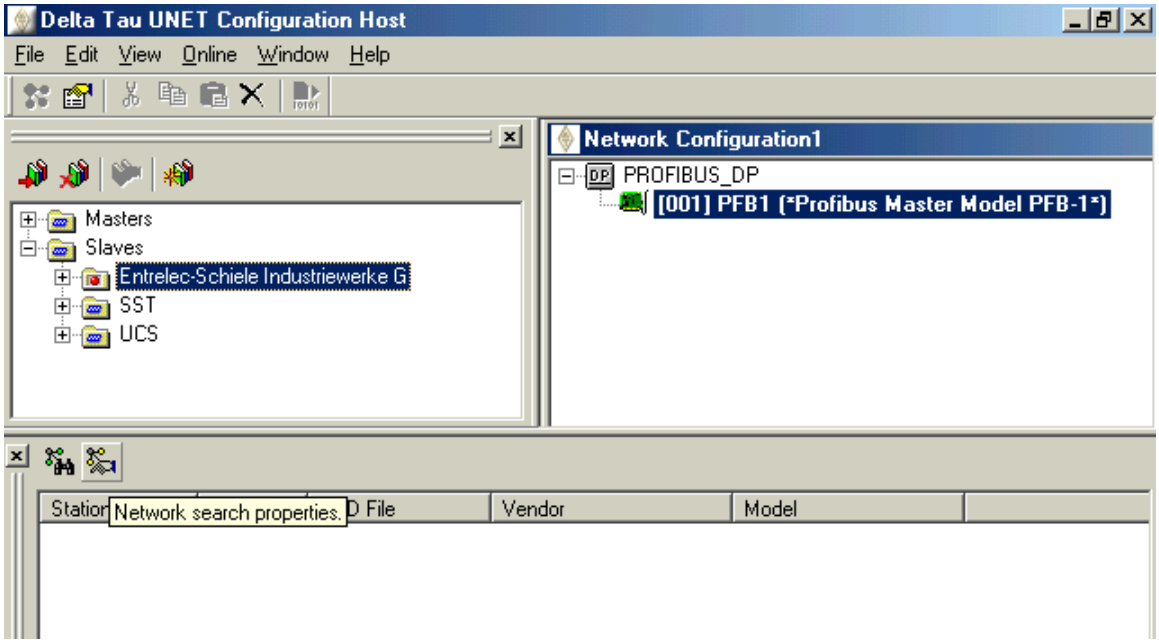
The Tags Tab for the Slave Node's Properties

Select ***Add*** from the Tags tab and the available tags will appear based on the modules that have been set up. Since the modules were configured to make the node's data available to you on the network, there should be little configuring in the ***Tags*** tab. You should just add all of the available tags. Select ***Add*** and add all of the tags to the network configuration.

When you are finished with setting up the tags click ***OK*** to return to the main form. The next step is to export the newly set up tags. This step exports an address file for the inputs and outputs on the Network. At this point all of the properties of the master and the slaves should have been configured.

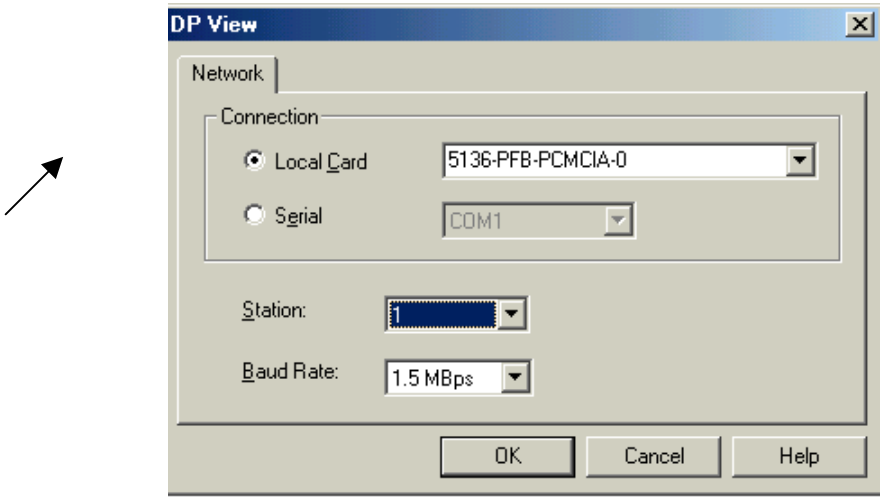
Using the SST Scanner Module 5136-PFB-PCM-ST to Browse the Network and Configure the Slave Nodes

In order to browse the network and poll the individual slaves for their information, you must first make sure that the proper software components have been installed on your laptop. The ***SST Scanner Module 5136-PFB-PCM-ST*** will ship with a CD with many network troubleshooting utilities. When installing these components the drivers necessary for browsing the network through the Delta Tau Configuration Host will also be loaded to your computer. After the components have been installed, launch the Delta Tau Configuration Host. This executable will be located in the Profibus Master Folder of the UNET directory. You should have already set up the master properties (above) and you are now ready to configure the browser properties. Click on the icon shown below:



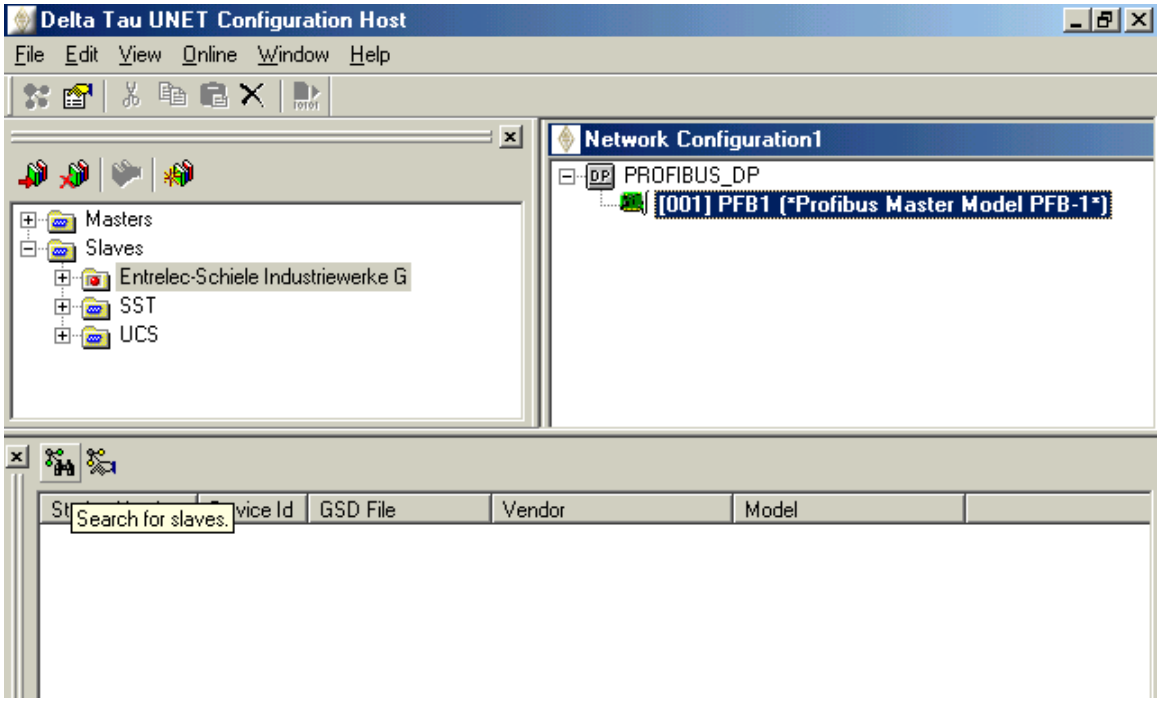
Configuring the Network Parameters of the Browse Feature

A window will appear on your screen similar to that shown below. There will be three fields contained within the window: ***Local Card***, ***Station***, and ***Baud Rate***. The system will sense that the scanner card is in the system and display the ***5136-PFB-PCMCIA-0*** within the ***Local Card*** drop down box. Configure the ***Baud Rate*** for the browse, making sure that all of the slave nodes can handle the selection. Choose the same ***Station Number*** as you chose earlier for your master. Select ***OK*** when you are finished.



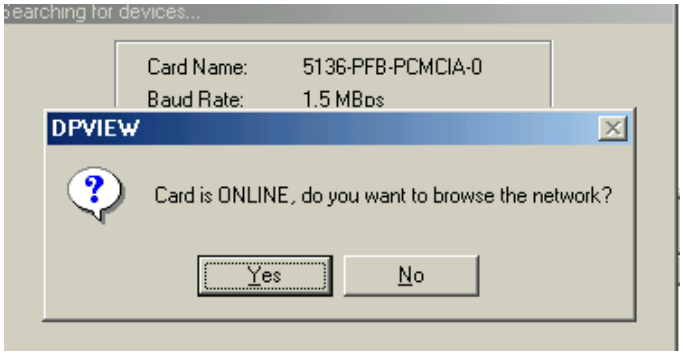
Online Browser Properties

When you are finished entering these parameters, click on the network browse icon shown below.



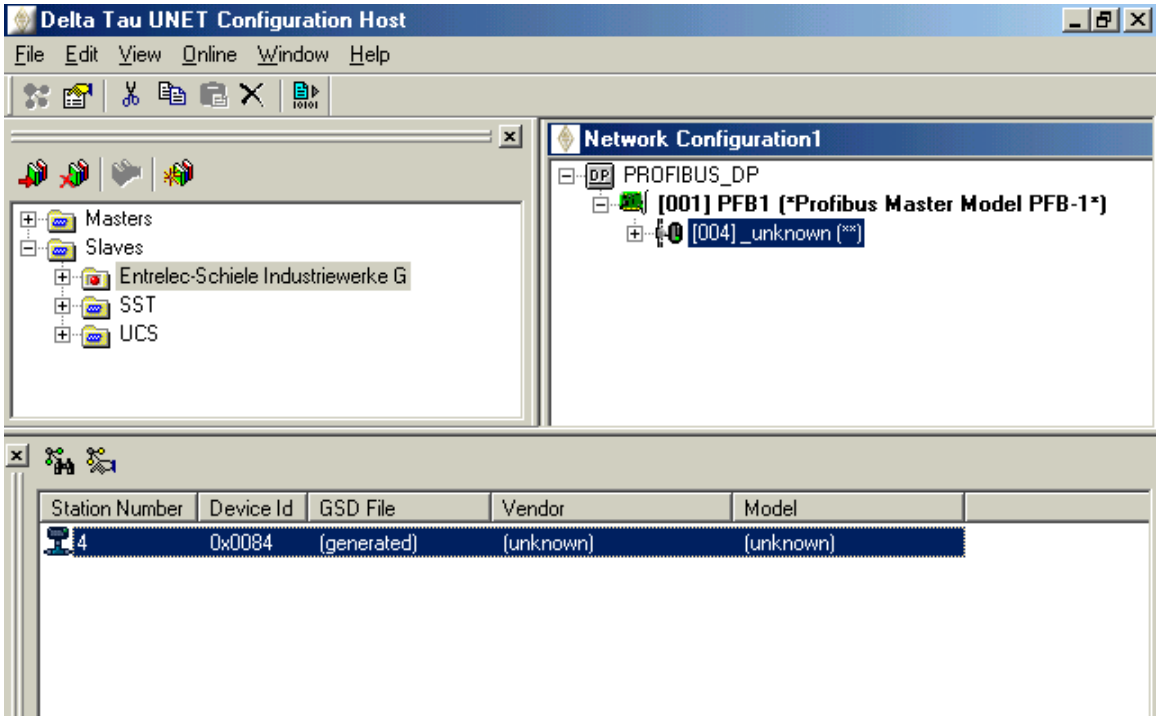
Launch the Browse Feature

After launching the browse feature you will see a screen similar to the following.



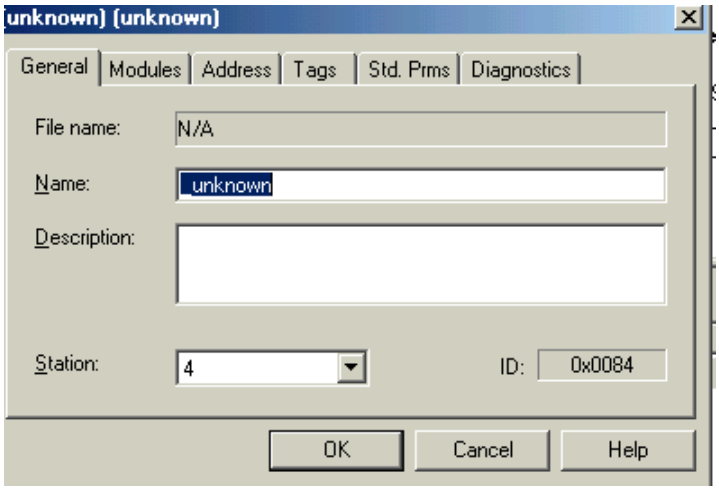
Searching for Devices on the Network

Select ***Yes*** at the prompt. Once the browser has finished you will have a list of the slave devices on the network. For this example there is one slave node at Station Number 4. The device will be listed in the form at the bottom of the application. Drag the slave node and drop it on top of the master icon in the Network Configuration Window. Your screen should then look as follows:



Results of the Network Browse

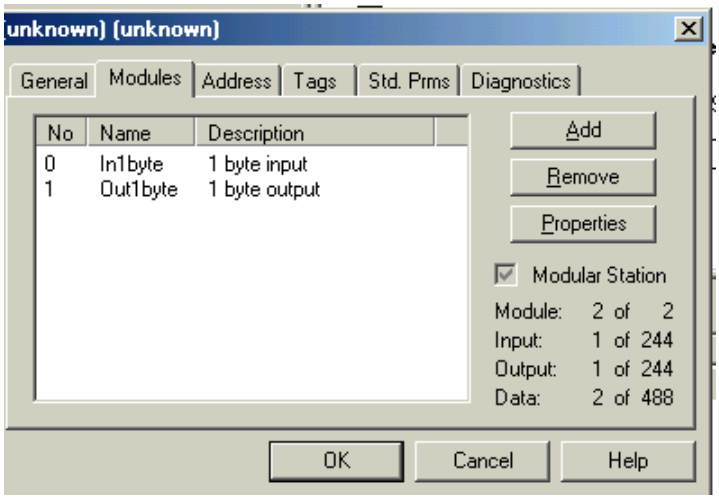
You are now ready to configure the slave node properties. Right click on the slave node icon within the Network Configuration Window in the upper right portion of the application. Select properties.



Slave Node Properties

As you can see from the figure below, there are seven tabs within the slave properties window: ***General***, ***Modules***, ***Address***, ***Tags***, ***Standard Parameters***, ***Extra Parameters*** and ***Diagnostics***. Many of the fields will be completed already, as a convenience of the browse feature. When browsing, the software essentially uploads an EDS file stored on a particular slave module.

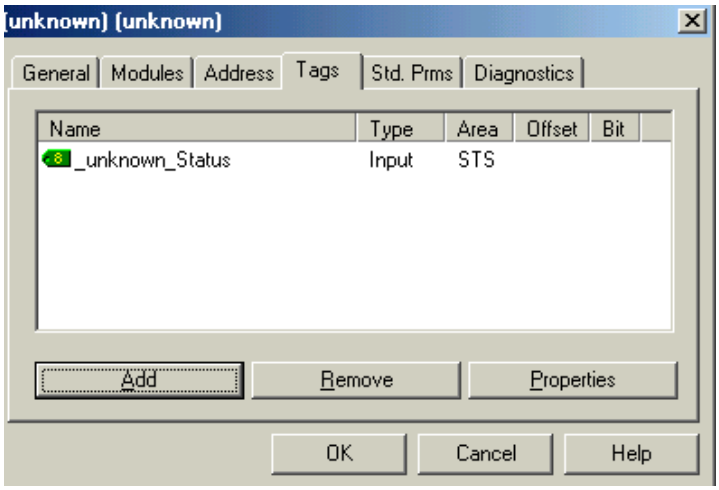
The key entry in the ***General*** tab is the ***Station*** number. Choose an open address on the Profibus network. Make sure not to use the same ***Station*** number as the master. The ***Name*** and ***Description*** text boxes are for adding comments and descriptions for reference. They are not critical to the network configuration.



Modules Setting Tab

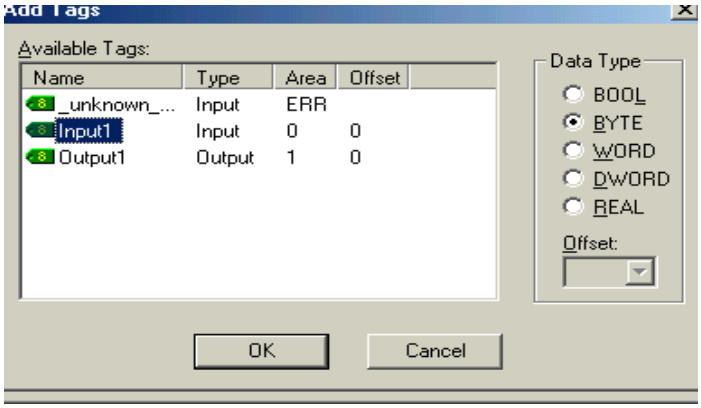
The ***Modules*** tab is where you configure the size and number of modules for each slave on the network. The size and number of modules for a slave is specific to the particular node that you have chosen. When the ***Add*** button is pressed, the possible choices for sizes will be listed based on the particular nodes' EDS file. Select the appropriate size and number of modules for this node and move onto the next tab.

The ***Tags*** tab needs to be configured for every network configuration. By setting up the tags we informed the application what data will be on the Network, and then Configuration host can set up the proper addressing in the output tag file and the personality file (scan list) of the master.

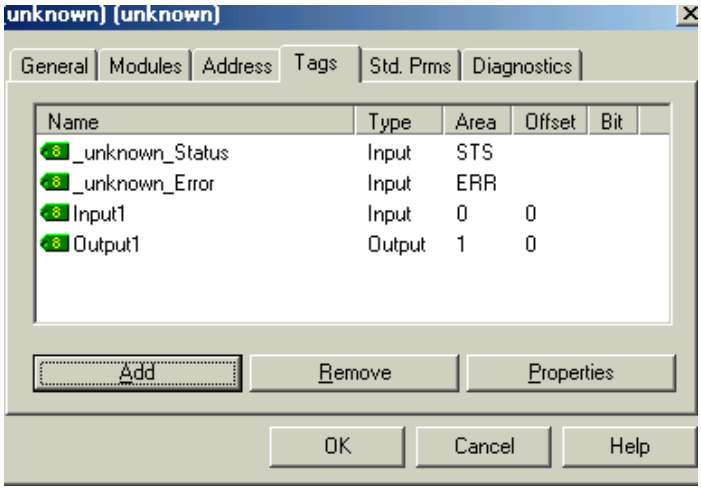


The Tags Tab for the Slave Node's Properties

Select ***Add*** from the Tags tab and the available tags will appear based on the modules that have been set up. Since the modules were configured to make the node's data available to you on the network, there should be little configuring in the ***Tags*** tab. You should just add all of the available tags. Select ***Add*** and add all of the tags to the network configuration.

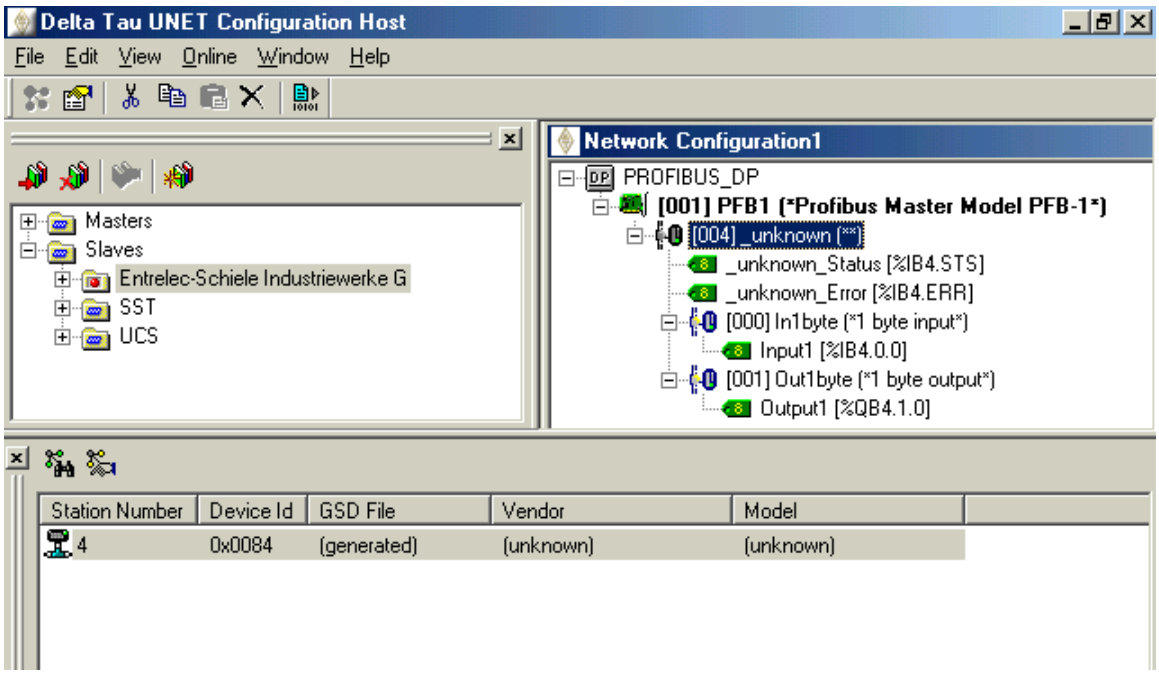


Adding Tags to the Network Configuration



Completing the Tags Setup

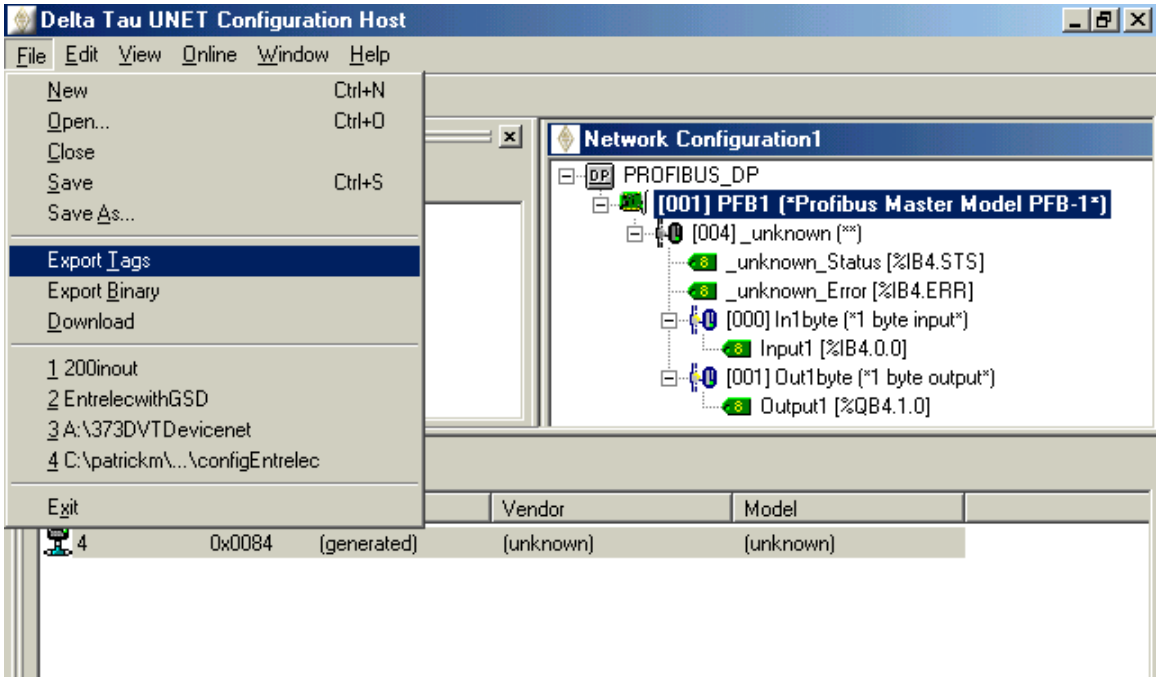
When you have finished setting up the tags, click ***OK*** to return to the main form. The next step is to export the newly set up tags. This step exports an address file for the inputs and outputs on the Network. At this point all of the properties of the master and the slaves should have been configured and your screen will look similar to the one below.



Naming the Output Tag

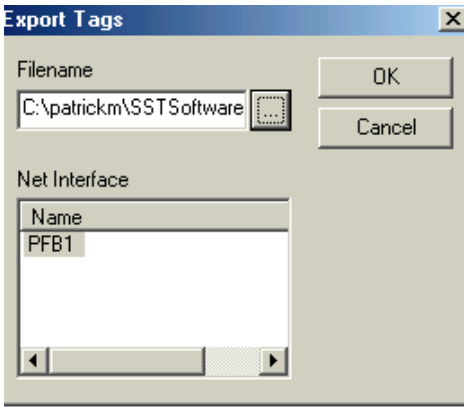
Creating and Exporting a Tag File for Setting Up the Master Scan List

To export the tags, click on ***File Menu*** and select ***Export Tags***.



Selecting Export Tags

The following window appears. Browse to the folder location where you would like to save the exported Tag File.



Export Tags Window

Once you have specified the location to save the file, select ***OK*** and the Tag address will be created. We will use the file later, when setting up the UNET PLC inside the controller. When you finish setting up and downloading the tags, then you are done configuring the personality of the master.

It is a good idea to save the Network Configuration through the File menu. Select ***Save*** or ***Save As*** from the file menu and store your network configuration on the hard drive. You can then reopen it later from the same application.

The ASCII "exported symbolic tag file" provides a one-line entry for each slave tag handled by the master. The format for each item is:

Name = Reserved, 1, Address, Byte Offset, Type

For the sample personality created in this chapter the contents of the file are:

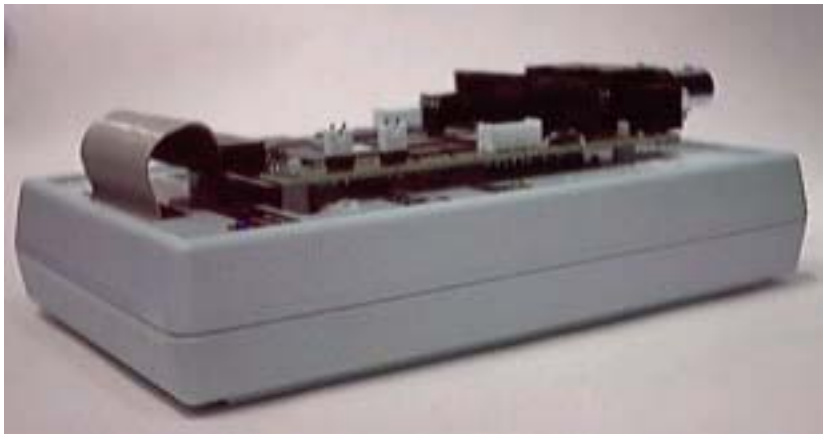
```
[ID]
Configuration = 0000037a01000001
[SYMBOLS]
_unknown_Status = 17,1,0x8C00,0x0,5
_unknown_Error = 17,1,0x8C01,0x0,5
Input1 = 17,1,0xC00,0x0,5
Output1 = 17,1,0x4C00,0x0,2
```

The addresses specified here are the actual UCS memory map addresses of the data. There is one issue you must be aware of. The Profibus tags file specifies the addresses in 32 words and then byte offsets within the words. The Profibus addresses are in bytes. When you use these addresses for the master UCS Service PLC, you will need to divide these by 4 and keep track of the byte offset yourself. This will be discussed in a later portion of this manual when it is time to input these addresses into the ***UmacProfibusMaster.h*** file.

Downloading the Personality File to the Master UCS Module.

There are essentially two different ways to download a personality file to the master UCS module.

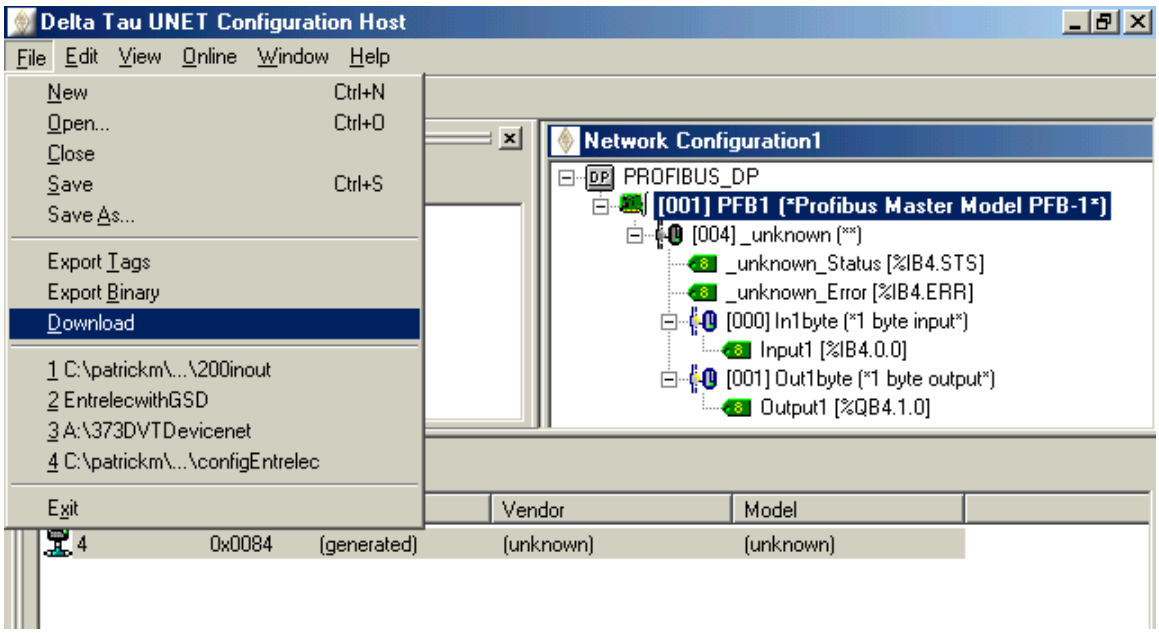
- **First method:** Use a download adapter module connected to your computer's serial port. The ***SST Download adapter UCS-DA-1 (INTL)*** is the module you will plug the UCS module into when downloading the personality file through the Delta Tau UNET Configuration Host Software. You will have to remove the UCS module from the Accessory 55E or other Delta Tau UNET product. Once you have the UCS module in the download adapter it should look like the following picture.



UCS Module plugged into the Download Adapter Properly

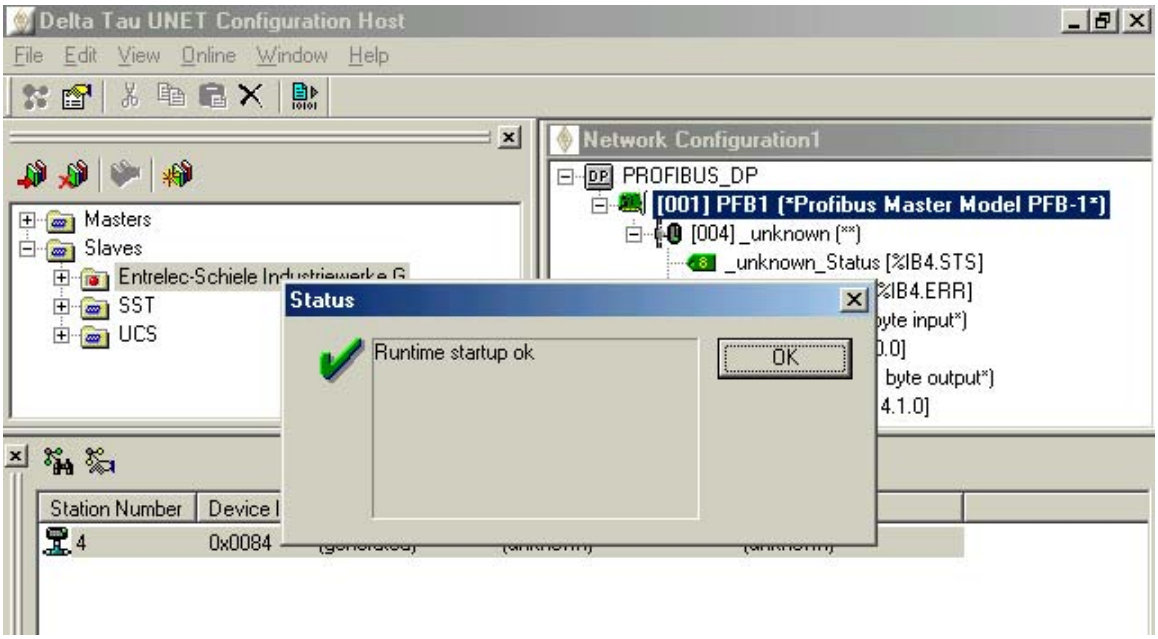
- **Second Method:** If you have the onboard Serial Port to the 55E or Qmac module, then you can connect directly to that serial port from your computer's serial communications port.

  After the proper connections have been made to the hardware and the previous steps above have been executed, you are now ready to download the personality. Right click on the Master icon from within the Network configuration window.



Downloading the Personality Data to the Master Module

When you select download, a window appears informing you that the software is in the downloading process. Once the download has been completed you will see the following screen appear. Click ***OK*** and you have finished configuring your master hardware.



Successful download to the UCS Module

If you had a problem downloading to your master module, recheck the connections and test your communications port on your computer. If you continue to experience problems, contact Delta Tau Technical Support.

After a successful download you are ready to configure the UMAC software for your specific network needs. Proceed to "Unet Hardware and Address Configuration," below.

Unet Hardware and Address Configuration

Configure the hardware settings of the 55E card through the switch on the top of the card labeled SW1. Make sure the address chosen does not conflict with any other I/O card address in the system. This address is important to the customization of the ***UmacProfibusMaster.h*** header file. See Address chart below:

Accessory 55E SW1 Address Settings

	CS10 SW1-1 ON SW1-2 ON	**CS12 SW1-1 OFF SW1-2 ON**	**CS14 SW1-1 ON SW1-2 OFF**	**CS16 SW1-1 OFF SW1-2 OFF**	**Definition**
SW1-3 ON SW1-4 ON	$078C00	$078D00	$078E00	$078F00	UCS_DATAREG_LO
	$078C01	$078D01	$078E01	$078F01	UCS_DATAREG_HI
	$078C02	$078D02	$078E02	$078F02	UCS_DATAREG_ADDR
	$078C03	$078D03	$078E03	$078F03	UCS_DATAREG_STAT
SW1-3 OFF SW1-4 ON	$079C00	$079D00	$079E00	$079F00	UCS_DATAREG_LO
	$079C01	$079D01	$079E01	$079F01	UCS_DATAREG_HI
	$079C02	$079D02	$079E02	$079F02	UCS_DATAREG_ADDR
	$079C03	$079D03	$079E03	$079F03	UCS_DATAREG_STAT
SW1-3 ON SW1-4 OFF	$07AC00	$07AD00	$07AE00	$07AF00	UCS_DATAREG_LO
	$07AC01	$07AD01	$07AE01	$07AF01	UCS_DATAREG_HI
	$07AC02	$07AD02	$07AE02	$07AF02	UCS_DATAREG_ADDR
	$07AC03	$07AD03	$07AE03	$07AF03	UCS_DATAREG_STAT
SW1-3 OFF SW1-4 OFF	$07BC00	$07BD00	$07BE00	$07BF00	UCS_DATAREG_LO
	$07BC01	$07BD01	$07BE01	$07BF01	UCS_DATAREG_HI
	$07BC02	$07BD02	$07BE02	$07BF02	UCS_DATAREG_ADDR
	$07BC03	$07BD03	$07BE03	$07BF03	UCS_DATAREG_STAT

Note: SW1-5 and SW1-6 must be set to ON.

UCS Software Parameter Setup

The following discussion explains how to configure the setup parameters for the Network as well as the controller resources that are used. You will need to edit ***UmacProfibusMaster.h*** to customize the UNET master for your particular application. None of the other files that service the Network transfers need to be modified. The main items to be set up include: node number, number of input bytes from the master, number of output bytes to the master, and the UNET card hardware address configured above.

The entire header file titled ***UmacProfibusMaster.h*** is located in Appendix B for reference. The following highlights the portions of the file that must be configured specifically for your application.

Variable and Memory Usage

The UNET PLC utilizes the following variables, memory addresses, and PLCs for the servicing of the Profibus Network communications and data transfer:

- **Q-variables for CS1**

Q7000..7999	**Always used for data storage (used in PLCC for data transfers)**

- **M-variables**

M4000..5999	Used by default for Network Data Access (see ProfibusMasterMvars.h) You can edit the file **ProfibusMasterMvars.h** to eliminate unnecessary Variables and change variable assignments as well.
M8000..8031	Always used for logic in ProfibusMasterServiceEvent.plc
M8050	**Always used for Network Commands**

- **L-variables**

L7998..7999	Always used for pointers to Network Data memory locations
L8000..8002	Always used for PLCC pointers to UCS memory
L8025..8027	Always used for PLCC logic
L8050	Always used for PLCC Network Commands

- **Scratch Memory Locations**

$10F0-$10FF	**These registers are open user memory set to zero at power up. These registers are always used.**

Make sure that your UMAC application is not using these same variables or memory locations. Serious problems may occur due to the overlapping of memory usage if you do not assure proper variable assignments.

The entire header file titled ***UmacProfibusMaster.h*** is at Appendix B for reference. The following highlights the portions of the file that will need to be configured specifically for your application.

Specifying a PLC Number

The UNET Service PLC is a PMAC PLC. You must define it by number. We do not recommend that it be defined as PLC 1. The first block of the template simply defines which number you are using for this service PLC. The default is PLC13 and most likely will not need to be changed. If you have more than one UNET interface card in your system, each card will require a separate PLC and therefore a different PLC_NUMBER. Upon downloading a separate PLCC program will be utilized during the data transfer operation. The PLCC # will be the same number that is entered into the header file below.

```
//CHOOSE A PLC AND PLCC NUMBER THAT YOU CAN USE IN YOUR SYSTEM
#DEFINE         PLC_NUMBER              13
```

Specifying the Characteristics of the Module

You must tell the Service PLC whether the module is a master or slave, and its node number. And, you can provide a control word that enables more advanced operational features. The second block of the template provides these definitions.

```
#DEFINE         UCS_NODE_NUMBER   5         ;STATION NUMBER(PFB)
```

This section defines the node as a slave with a network ID of 5. The actual number you select must not conflict with another node on the network. See the documentation for your network to determine the largest available number, and the numbers already assigned. We never recommend using a network ID of 0 or 1, or the last available ID available on the network. These are often reserved for special purposes.

Specifying the Number of Data Words and their Location

This section defines how many 8-bit data bytes the slave will produce and consume. Inputs are data bytes received by the UNET module from a master, outputs are data bytes to be sent to the master.

```
//INPUTS TO NETWORK FROM UCS
#DEFINE         MASTER_INP_SIZE      16
#DEFINE         MASTER_INPUT_STARTADDRESS       $4C00

//OUTPUTS FROM NETWORK TO UCS
#DEFINE         MASTER_OUT_SIZE      16
#DEFINE         MASTER_OUTPUT_STARTADDRESS     $C00
```

The definitions in the default template define 16 input and 16 output 8-bit bytes that are mapped as pairs to P-Variables. As defined in ***ProfibusMasterMvars.h***, the input data will start at M6000 and the output data will start at M7000. M6000 will contain the first two bytes of data received from the master, M6001, the next two, and so forth. Similarly, M7000 contains the first two bytes to be sent to the master, M7001 the next two, and so forth. Note that the architecture of the service PLC reserves extra M-variables for other UCS Networks, M6256-M6999 and M7256-M7999. Use these if you wish for other reasons, but avoid doing so unless there is an absolute need. The less things are changed, the simpler configuring and troubleshooting becomes.

The addresses are located from the tags file that was exported above.We are concerned only with the addresses for the inputs and outputs. The Unet Service PLC was written to transfer the data and must be modified to include other tags from various slaves. If this other information is vital to your system, please contact Delta Tau Technical Support with help in configuring this feature.

The configuration of these parameters is extremely important to your master device. You must be certain that:

- You understand how your master maps slave data to its internal memory space. It can be very confusing when the master and slave use different data sizes and names.
- You understand the limitations on numbers of slave inputs and outputs imposed by your master. This is especially true if your master is servicing multiple slaves. For example, your master may allow a maximum of 32 input bytes and 32 output bytes for all slaves. This may impact your final selection of a data provided by your UMAC. It is imperative that you carefully test and document the transfers before proceeding with development of detailed applications.

Specifying the Base Address for the UNET Card

When installing a UNET card you selected a base address using SW1 on the card. The address of the card must be specified in the UNET Service PLC template. This may require changing the four definitions shown here. These four definitions specify four contiguous addresses whose base address is $78D00. The "Accessory 55E Sw1 Address Settings" table (above) provides the base addresses for other SW1 settings. The base addresses here are defined in the low 16 bits of the definitions located in ***ProfibusMasterServiceHeader.h***.

```
//UMAC ADDRESSES BASED ON SW1 SETTINGS ON THE ACCESSORY 55E
#DEFINE UCS_DATALO_ADDR                 $78D00
#DEFINE UCS_DATAHI_ADDR         $78D01
#DEFINE UCS_ADDRESS_ADDR                $78D02
#DEFINE UCS_CONTROL_STATUS_ADDR $78D03
```

Installing the UNET Parameters and UMAC Software

After modifying the above parameters, there should be nothing else that needs to be changed in the UMAC program files. The next step is to download ***UmacProfibusMaster.h*** to the Delta Tau UMAC controller and enable it by setting i5=2 and issuing the online command “enable plc 13”. Since the PLC number is a variable in the ***UmacProfibusMaster.h*** you may have to change which PLC you enable on UMAC if you changed the setting from the default.

Notice that the last line in the template is:

```
#include "ProfibusMasterMvars.h"
#include "ProfibusMasterServiceHeader.h"
#include "ProfibusMasterServiceEvent.plc"
```

Do not download ***ProfibusMasterServiceEvent.plc*** directly; download the template. If you have Pewin32 version 2.36 or earlier, make sure that you have enabled the MACRO/PLCC option in the editor prior to downloading. If you experience problems during the download, check to make sure that you have enabled the MACRO/PLCC option and that you have not changed a macro name by mistake.

Putting the UNET Card On and Offline

Reset Command

Reset the UNET slave by setting m8050=1. This will force the PLC to reset the UNET interface card. During the reset the UCS Module Status Indicator LED will turn red, then green, then off. If the status indicator is flashing green, this is acceptable. You can repeat the command as desired to verify the reset operation.

If the Status Indicator LED changes do not occur as described above, there are several possible problems:

- Your Service PLC was not downloaded or is not enabled. Check to verify these conditions.
- The base address specified for the UNET memory map (i.e., $78Dxx) does not match the setting of the card, or is in conflict with an existing device. Verify the setting of SW 1 and that the address is properly reflected in the template.
- If you changed the base addresses for the M variables in **ProfibusMasterServiceHeader.h** (which you shouldn’t have), your command variable is not properly set in M8050. Check and verify this condition.
- Make certain that changes in base M variable indices, CS timer, and scratch pad memory locations are consistent and do not interfere with other PLCs or motion programs you have written.

Online Command

After the green Status LED returns to a flashing green state, put the card online by setting m8050=2.

This should cause the UCS Network Status Indicator LED to become flashing or steady state green. M8050 will then be set to 4 to notify the user that Network is now transferring data. If the LED is off, flashing or steady state red, or has an amber color, verify that:

- Your network is properly configured, has power applied, and is terminated. The best situation is one in which you have thoroughly configured and network using another SST network interface board, or have verified network operation using the tools supplied by your controller vendor (Allen Bradley, Siemens, et al.).
- The node number specified in the template ($0005 in the default template) is not in conflict with any other node on the network. If it is, make the appropriate change and download the PLC again.
- You have correctly specified a number of inputs and outputs that is consistent with the limitations of the network.
- You have selected the proper baud rate for the Network and it matches that of your master.
- There are no masters on the network attempting to connect to your UNET slave. You can verify this by disconnecting all other devices from the network.

Offline Command

During your verification of the module, you can take the card off-line by setting

M8050 = 3

You can repeatedly reset the card (M8050 = 1), go on line (M8050 = 2), and off-line (M8050 = 3) during testing and verification of your card, Service PLC, and network.

Advanced Testing of your Master Configuration

Prior to the development of any PMAC motion or PLC programs to use the P-variable data, you should conduct further testing of your network and slave configuration. This will involve using the network tools you purchased from SST or those supplied with your master controller. To accomplish these tests you must:

1. Reset the UNET card by typing M8050 = 1
2. Place the UNET card on line by typing M8050 = 2
3. Verify data transfers by checking M8050 is equal to 4

The last step causes your UNET Service PLC to actually read and write the UCS module's memory map. In doing so, the data received from the network is copied into your Input variables (M6000 and up) and the data contained in your output variables (M7000 and up) is copied into UCS memory. If you do not enable transfers an on-line UCS module may be functioning perfectly but you'd never know it because the data would not be accessible to the PMAC.

Interfacing the UNET Slave to the Master on the Profibus Network

Now that the master is ready and online it is time to test it on the Profibus Network. Make sure that the Profibus Network is powered and that there is a terminating resistor on the last node in the Network. After this try putting the card online and test the mapping of the P-variables to the data on each of the slaves

APPENDIX A- PROFIBUS SLAVE FOR UMAC

UmacProfibusSlave.h

```
CLOSE
DELETE GATHER

//CHOOSE A PLC AND PLCC NUMBER THAT YOU CAN USE IN YOUR SYSTEM
#DEFINE PLC_NUMBER            13

//
#DEFINE UCS_NODE_NUMBER       0       ;STATION NUMBER(PFB)

//INPUTS TO NETWORK FROM UCS
#DEFINE SLAVE_INP_SIZE 200            ; Number of slave inputs in bytes

//OUTPUTS FROM NETWORK TO UCS
#DEFINE SLAVE_OUT_SIZE 200            ; Number of slave outputs

//UMAC ADDRESSES BASED ON SW1 SETTINGS ON THE ACCESSORY 55E
#DEFINE UCS_DATALO_ADDR               $78D00
#DEFINE UCS_DATAHI_ADDR               $78D01
#DEFINE UCS_ADDRESS_ADDR              $78D02
#DEFINE UCS_CONTROL_STATUS_ADDR       $78D03


#include "ProfibusSlaveMvars.h"
#include "ProfibusSlaveServiceHeader.h"
#include "ProfibusSlaveServiceEvent.plc"
```

ProfibusSlaveServiceHeader.h

```
#DEFINE SLAVE                 1             ;SET TO 1 SINCE THIS IS A SLAVE
#DEFINE DEVICENET             0             ;SET TO ONE IF THIS IS DEVICENET
#DEFINE UCS_BAUDRATE          125           //NOT USED BUT NEED FOR UMAC SERVICE EVENT

;       The command variable

#DEFINE IDLE                  $0000
#define RESET                 $0001         //PERFORMS A SOFT RESET
#define UCS_OPEN              $0002         //SENDS OPEN TRIGGER AND STARTS TRANSFERS
#define UCS_CLOSE             $0003         //SENDS CLOSE TRIGGER
#DEFINE TRANSFER              $0004         //READ ONLY FOR USER
#define LATCH_ERR             $0005         //NETWORK ERROR CARD NEEDS A SOFT RESET

#define UCS_CMD_IDBLOCK       $0100   //VARIOUS DEBUGGING CAPABILITIES-TAKE OUT OF USER PLC
#define UCS_CMD_CSBLOCK       $0101
#define UCS_CMD_ERRORLOG      $0102
#define UCS_CMD_SENDTRIGGER   $1000   //USED FOR TRIGGER HANDLER STARTER

//FIRST BYTE OF UCS_STATUS_CONTROL IS STATUS FROM UCS AND 2ND BYTE IS CONTROL TO UCS
#DEFINE UCS_BUSY      UCS_STATUS&1                ;BIT 0 OF STATUS IS BUSY FLAG
#DEFINE UCS_BUSY_L    UCS_STATUS_L&1              ;BIT 0 OF STATUS IS BUSY FLAG

#DEFINE UCS_IRQ       UCS_STATUS&2                ;BIT 1 OF STATUS IS IRQ FROM UCS
#DEFINE UCS_CMD UCS_COMMAND&$F
#DEFINE UCS_CMD_L     UCS_COMMAND_L&$F

//THESE MEMORY LOCATIONS ARE SET AND ARE CONSISIENT WITH PROFIMASTER DEFAULTS IN CFGHOST
#define SLAVE_INP_IRAM        $300          ; IRAM location of slave inputs-SELF DEFINED
#define SLAVE_OUT_IRAM        $1300         ; IRAM location of slave outputs-SELF DEFINED
#DEFINE IRQ_CLEAR             $2
#DEFINE SEND_INT              $2
#DEFINE OPEN_EVENT            $1
#DEFINE TRUE                  1
#DEFINE ON                    1
#DEFINE FALSE                 0
#DEFINE OFF                   0
#DEFINE RUN_MODE              3
```

```
;READ IN MODULE STATUS WORD
#DEFINE MS_CONVERT            8388608/I10
#DEFINE MODULE_STATUS_ADDR    $0
#DEFINE UCS_STATUS_ADDR       $21
#DEFINE EVENT_IN_ADDR         $26
#DEFINE EVENT_OUT_ADDR        $27
#DEFINE TRIGGER_IN_ADDR       $28
#DEFINE TRIGGER_OUT_ADDR      $29
#DEFINE UCS_READ              $8000

;*************************************************************************
//
#DEFINE TIMER                 I5511                //USES TIMER FROM CS5

#DEFINE UCS_DATALO            M8000
#DEFINE UCS_DATAHI            M8001
#DEFINE UCS_ADDRESS           M8002
#DEFINE UCS_DATALO_L          L8000
#DEFINE UCS_DATAHI_L          L8001
#DEFINE UCS_ADDRESS_L         L8002
#DEFINE UCS_CONTROL_STATUS    M8003
#DEFINE UCS_CONTROL           M8004
#DEFINE UCS_STATUS            M8005
#DEFINE UCS_STATUS_L          L8005
#DEFINE MODULE_STATUS_LO      M8010
#DEFINE MODULE_STATUS_HI      M8011
#DEFINE UCS_STATUS_LO         M8012
#DEFINE UCS_STATUS_HI         M8013
#DEFINE TRIGGERIN_POINTER_LO  M8014
#DEFINE TRIGGERIN_POINTER_HI  M8015
#DEFINE TRIGGERIN_POINTER     M8016
#DEFINE EVENTIN_POINTER_LO    M8017
#DEFINE EVENTIN_POINTER_HI    M8018
#DEFINE EVENTOUT_POINTER_LO   M8019
#DEFINE EVENTOUT_POINTER_HI   M8020
#DEFINE EVENT_LO              M8021
#DEFINE EVENT_HI              M8022
#DEFINE BAUD                  M8023
#DEFINE NODE_NUMBER           M8024
#DEFINE COUNTER               M8025
#DEFINE COUNTER_L             L8025
#DEFINE INPUT_LATCH           L8026
#DEFINE OUTPUT_LATCH          L8027

#DEFINE I_O_UPDATE_TIME             M8028
#DEFINE SERVO_COUNTER         M8029
#DEFINE TEMP_UPDATE1          M8030
#DEFINE TEMP_UPDATE2          M8031
#DEFINE UCS_COMMAND           M8050
#DEFINE UCS_COMMAND_L         L8050

; -- PMAC Specific Memory Addresses

; -- PMAC Specific Memory Addresses M-VAR DEFS
UCS_DATALO->X:UCS_DATALO_ADDR,24                    //M8000
UCS_DATAHI->X:UCS_DATAHI_ADDR,24                    //M8001
UCS_ADDRESS->X:UCS_ADDRESS_ADDR,24                  //M8002
UCS_DATALO_L->X:UCS_DATALO_ADDR,24                  //L8000
UCS_DATAHI_L->X:UCS_DATAHI_ADDR,24                  //L8001
UCS_ADDRESS_L->X:UCS_ADDRESS_ADDR,24                //L8002
UCS_CONTROL_STATUS->X:UCS_CONTROL_STATUS_ADDR,24    //M8003
UCS_CONTROL->X:UCS_CONTROL_STATUS_ADDR,8,8          //M8004
UCS_STATUS->X:UCS_CONTROL_STATUS_ADDR,0,8           //M8005
UCS_STATUS_L->X:UCS_CONTROL_STATUS_ADDR,0,8         //L8005

UCS_COMMAND->Y:$10F0,0,24
UCS_COMMAND_L->Y:$10F0,0,24

MODULE_STATUS_LO->Y:$10F1,0,24                            //M8010
MODULE_STATUS_HI->X:$10F1,0,24                            //M8011
```

```
UCS_STATUS_LO->Y:$10F2,0,24                          //M8012
UCS_STATUS_HI->X:$10F2,0,24                          //M8013
TRIGGERIN_POINTER_LO->Y:$10F3,0,24                   //M8014
TRIGGERIN_POINTER_HI->X:$10F3,0,24                   //M8015
TRIGGERIN_POINTER->Y:$10F4,0,24                             //M8016
EVENTIN_POINTER_LO->Y:$10F5,0,24                            //M8017
EVENTIN_POINTER_HI->X:$10F5,0,24                            //M8018
EVENTOUT_POINTER_LO->Y:$10F6,0,24                    //M8019
EVENTOUT_POINTER_HI->X:$10F6,0,24                    //M8020
EVENT_LO->Y:$10F7,0,24                               //M8021
EVENT_HI->X:$10F7,0,24                               //M8022
BAUD->Y:$10F8,0,24                                   //M8023
NODE_NUMBER->X:$10F8,0,24                            //M8024
COUNTER_L->Y:$10F9,0,24                                     //ML025
INPUT_LATCH->X:$10F9,0,24                                   //L8026
OUTPUT_LATCH->X:$10FA,0,24                           //L8027
I_O_UPDATE_TIME->Y:$10FA,0,24,S                             //M8028
SERVO_COUNTER->X:$0,0,24                                    //M8029
TEMP_UPDATE1->X:$10FB,0,24                           //M8030
TEMP_UPDATE2->Y:$10FB,0,24                           //M8031

L7998->Y:$7B58[1024]                ;MUST BE A FACTOR OF 2
L7999->X:$7B58[1024]                ;MUST BE A FACTOR OF 2
```

ProfibusSlaveServiceEvent.plc

```
//SERVICE PLC FOR THE DATA TRANSFERS AND ERROR HANDLING
//SEE FLOW CHART ON UNET CD FOR LOGIC TRAIN

//IN=NETWORK IN FROM SLAVE
//OUT=NETWORK OUT TO SLAVE


OPEN PLC PLC_NUMBER CLEAR

;Check IRQ Status (INT FROM UCS)
If (UCS_IRQ != FALSE)         //CLEAR IRQ
UCS_STATUS=IRQ_CLEAR                 //TELL UCS I SAW IRQ AND CLEAR IT

;*****************************************************************************
;STORE MODULE STATUS AND CHECK IT FOR RUN MODE
While (UCS_BUSY = TRUE)
  ENDWHILE                                  //WAIT UNTIL UCS IS NOT BUSY (1-8uSEC TYPICAL)
  UCS_ADDRESS = UCS_READ + MODULE_STATUS_ADDR       ;IRAM address for MODULE_STATUS
  While (UCS_BUSY = TRUE)
    ENDWHILE                                //WAIT UNTIL UCS IS NOT BUSY (1-8uSEC TYPICAL)
    MODULE_STATUS_LO = UCS_DATALO
       //STORES MODULE STATUS AND CONTROL
    MODULE_STATUS_HI= UCS_DATAHI
    //*************************************************************************
    ;IF UCS IS NOT IS RUN MODE
    If (MODULE_STATUS_LO&$FF != RUN_MODE)           ;MODULE STATUS MODE IS LOW BYTE OF IRAM 0
    While (UCS_BUSY = TRUE)
      ENDWHILE
                                    //WAIT UNTIL UCS IS NOT BUSY (1-8uSEC TYPICAL)
      UCS_ADDRESS = UCS_READ + UCS_STATUS_ADDR      ; IRAM address
      While (UCS_BUSY = TRUE)
        ENDWHILE
                                    //WAIT UNTIL UCS IS NOT BUSY (1-8uSEC TYPICAL)
        UCS_STATUS_LO = UCS_DATALO                                  //STORES UCS_STATUS BITS
        UCS_STATUS_HI = UCS_DATAHI

        UCS_COMMAND = LATCH_ERR
      Else
        //UCS_COMMAND= IDLE
      EndIf
      ;*************************************************************************
      //PROCESS FOR EVENT HANDLER
      ///READ IN EVENT OUT POINTER/////////////////
      While (UCS_BUSY = TRUE)
      EndWhile
```

```
      UCS_ADDRESS = UCS_READ + EVENT_OUT_ADDR               ; Read EVENT_OUT
      While (UCS_BUSY = TRUE)
      EndWhile
      EVENTOUT_POINTER_LO = UCS_DATALO
      EVENTOUT_POINTER_HI = UCS_DATAHI

      ///READ IN EVENT IN POINTER/////////////////
      While (UCS_BUSY = TRUE)
      EndWhile
      UCS_ADDRESS = UCS_READ + EVENT_IN_ADDR         ; Read EVENT_IN
      While (UCS_BUSY = TRUE)
      EndWhile
      EVENTIN_POINTER_LO = UCS_DATALO
      EVENTIN_POINTER_HI = UCS_DATAHI

      //Compare EVENT_IN to EVENT_OUT//////////////

      If (EVENTIN_POINTER_LO > EVENTOUT_POINTER_LO)
        While (UCS_BUSY = TRUE)
        EndWhile
        UCS_ADDRESS = UCS_READ + EVENTOUT_POINTER_LO
        While (UCS_BUSY = TRUE)
        EndWhile
        EVENT_LO = UCS_DATALO
        EVENT_HI = UCS_DATAHI
        While (UCS_COMMAND = UCS_OPEN And EVENT_LO&$FF != OPEN_EVENT)
          While (UCS_BUSY = TRUE)
          EndWhile
          UCS_ADDRESS = UCS_READ + EVENTOUT_POINTER_LO
          While (UCS_BUSY = TRUE)
          EndWhile
          EVENT_LO = UCS_DATALO
          EVENT_HI = UCS_DATAHI
        EndWhile

        EVENTOUT_POINTER_LO = EVENTOUT_POINTER_LO + 1
        If (EVENTOUT_POINTER_LO = $200)
          EVENTOUT_POINTER_LO=$100
        EndIf

        While (UCS_BUSY = TRUE)
        EndWhile
        UCS_ADDRESS = EVENT_OUT_ADDR
        While (UCS_BUSY = TRUE)
        EndWhile
        UCS_DATALO = EVENTOUT_POINTER_LO
        UCS_DATAHI = 0
        While (UCS_BUSY = TRUE)
        EndWhile

       If ((EVENT_LO&$FF) != OPEN_EVENT)
          UCS_COMMAND=LATCH_ERR
       Else
          ENABLE PLCC13
          UCS_COMMAND = TRANSFER
          DISABLE PLC13
        EndIf
       EndIf
      //

      ;*************************************************************************

      ELSE                    //IRQ NOT PRESENT
      ;*************************************************************************
      ;COMMAND RESET
      If (UCS_CMD = RESET)
        UCS_CONTROL=RESET     // toggle lsb of control byte for reset
        TIMER=10*8388608/I10                                //NEED TO WAIT MIN OF 50usec
according to client.pdf
        While (TIMER > 0)
        EndWhile       //THIS VALUE IS A CONSERVATIVE 2 SERVO CYCLES
```

```
        UCS_STATUS=$0E //INTO STATUS CLEAR ALL FAULTS--WHY SEPARATE???
        UCS_CONTROL=0         //SET CONTROL TO ZERO
        While (UCS_IRQ = FALSE)
        EndWhile
        UCS_STATUS=IRQ_CLEAR
        UCS_COMMAND=IDLE
        TRIGGERIN_POINTER=0
      EndIf
      ;*************************************************************************
; Send a CLOSE Trigger

If (UCS_CMD = UCS_CLOSE)
  //READ TRIGGER IN POINTER
  While (UCS_BUSY = TRUE)
  EndWhile
  UCS_ADDRESS = UCS_READ + TRIGGER_IN_ADDR          ; Fetch TRIGGER_IN pointer
  While (UCS_BUSY = TRUE)
  EndWhile
  TRIGGERIN_POINTER_LO = UCS_DATALO
  TRIGGERIN_POINTER_HI = UCS_DATAHI
  While (TRIGGERIN_POINTER_LO < TRIGGERIN_POINTER)
    While (UCS_BUSY = TRUE)
    EndWhile
    UCS_ADDRESS = UCS_READ + TRIGGER_IN_ADDR        ; Fetch TRIGGER_IN pointer
    While (UCS_BUSY = TRUE)
    EndWhile
    TRIGGERIN_POINTER_LO = UCS_DATALO
    TRIGGERIN_POINTER_HI = UCS_DATAHI
  EndWhile
       //STUFF IN A CLOSE TRIGGER
       While (UCS_BUSY = TRUE)
       EndWhile
       //
       UCS_ADDRESS = TRIGGERIN_POINTER_LO ; Setup address to write to current TRIGGER_IN
       While (UCS_BUSY = TRUE)
       EndWhile
       UCS_DATALO = $0104                           ; Trigger command = CLOSE + OPERATOR
SHUTDOWN SEE CLIENT FOR DETAILS
       UCS_DATAHI = $0000
       TRIGGERIN_POINTER=TRIGGERIN_POINTER_LO+1
       If (TRIGGERIN_POINTER = $300)
              TRIGGERIN_POINTER=$200
       EndIf
       While (UCS_BUSY = TRUE)
       EndWhile
       UCS_ADDRESS = TRIGGER_IN_ADDR
       While (UCS_BUSY = TRUE)
       EndWhile
       UCS_DATALO     =      TRIGGERIN_POINTER
       UCS_DATAHI     =      $0
       While (UCS_BUSY = TRUE)
       EndWhile
       UCS_CONTROL = SEND_INT               ;SEND INTERUPT TO PROCESS TRIGGER DATA
       While (UCS_IRQ = FALSE)
       EndWhile
       UCS_COMMAND = IDLE     ; Enable post TRIGGER Handler
       //SET AN INT TO TELL UCS TO PROCESS TRIGGER
       EndIf
;*************************************************************************
; Send an OPEN Trigger

If (UCS_CMD = UCS_OPEN)
  If (SLAVE = TRUE)
    ;FIRST WRITE TO INPUT DEFINITION-IF IT IS A MASTER PERSONALITY WILL OVERWRITE IT
    While (UCS_BUSY = TRUE)
    EndWhile
    UCS_ADDRESS = $0022                                                 ;ADDRESS FOR INPUT DEF
    While (UCS_BUSY = TRUE)
    EndWhile
    UCS_DATALO = SLAVE_INP_SIZE
    UCS_DATAHI = SLAVE_INP_IRAM
```

```
    ;NEXT WRITE TO OUTPUT SIZE AND LOCATION-MASTER'S PERSONALITY WILL OVERWRITE IT
    While (UCS_BUSY = TRUE)
    EndWhile
    UCS_ADDRESS = $0023                                                    ;ADDRESS FOR OUTPUT
DEF
    While (UCS_BUSY = TRUE)
    EndWhile
    UCS_DATALO = SLAVE_OUT_SIZE
    UCS_DATAHI = SLAVE_OUT_IRAM
  EndIf

  ; Sending open trigger and handle event queue
  While (UCS_BUSY = TRUE)
  EndWhile
  UCS_ADDRESS = UCS_READ + TRIGGER_IN_ADDR                          ;READ + ADDRESS FOR TRIGGER
IN POINTER
  While (UCS_BUSY = TRUE)
  EndWhile
  TRIGGERIN_POINTER_LO = UCS_DATALO
  TRIGGERIN_POINTER_HI = UCS_DATAHI
  While (TRIGGERIN_POINTER_LO < TRIGGERIN_POINTER)
    While (UCS_BUSY = TRUE)
    EndWhile
    UCS_ADDRESS = UCS_READ + TRIGGER_IN_ADDR        ; Fetch TRIGGER_IN pointer
    While (UCS_BUSY = TRUE)
    EndWhile
    TRIGGERIN_POINTER_LO = UCS_DATALO
    TRIGGERIN_POINTER_HI = UCS_DATAHI
  EndWhile
  //increment trigger in pointer
  While (UCS_BUSY = TRUE)
  EndWhile
  //write the trigger
  UCS_ADDRESS = TRIGGERIN_POINTER_LO ; Setup address to write to current TRIGGER_IN
  While (UCS_BUSY = TRUE)
  EndWhile
        If (SLAVE = TRUE)
        UCS_DATALO = $0701            ;7->DISABLES C.O.S; ENABLES PLUG AND PLAY; AND SWITCH DATA
SUPPLIED(BAUD RATE ADN NODE NUMBER)
        Else
        UCS_DATALO = $0401
        EndIf
  If (DEVICENET = TRUE And SLAVE = TRUE)
    If (UCS_BAUDRATE = 125)
      BAUD=$00
    EndIf
    If (UCS_BAUDRATE = 250)
      BAUD=$40
    EndIf
    If (UCS_BAUDRATE = 500)
      BAUD=$80
    EndIf
  Else
    BAUD=$00
  EndIf

  UCS_DATAHI = BAUD + (UCS_NODE_NUMBER & $3F)
  TRIGGERIN_POINTER=TRIGGERIN_POINTER_LO+1
  If (TRIGGERIN_POINTER = $300)
    TRIGGERIN_POINTER=$200
  EndIf

  While (UCS_BUSY = TRUE)
  EndWhile
  UCS_ADDRESS = TRIGGER_IN_ADDR
  While (UCS_BUSY = TRUE)
  EndWhile
  UCS_DATALO = TRIGGERIN_POINTER
  UCS_DATAHI = 0
  While (UCS_BUSY = TRUE)
  EndWhile
```

```
  //send INT TO UCS
  UCS_CONTROL = SEND_INT                    ;SEND INTERUPT TO PROCESS TRIGGER DATA
  TIMER=2500*8388608/I10
  While (TIMER > 0)
  EndWhile
  While (UCS_IRQ = FALSE)
  EndWhile
  //
  EndIf               ;END OF OPEN TRIGGER
  ;*************************************************************************
    If (UCS_STATUS&$8 = $8)
    UCS_COMMAND=RESET
    EndIf
EndIf
close


;*****************************************************************************
//PLCC code for transfers
OPEN PLCC PLC_NUMBER CLEAR
If (UCS_CMD_L = TRANSFER And UCS_STATUS_L&8 != 8)// And(UCS_IRQ = FALSE)
  ;*************************************************************************
  ; -- Output the data to NETWORK from PMAC
  COUNTER_L = 0
  While (UCS_BUSY_L = TRUE)
  EndWhile
  UCS_ADDRESS_L = SLAVE_INP_IRAM
  While (COUNTER_L < SLAVE_INP_SIZE+1)
    //WE DON'T WANT TO SKIP OUT OF HERE SO WE USE MULTIPLE IF STATEMENTS TO MAX BANDWIDTH
    INPUT_LATCH=FALSE
    //TRIAL 1 FOR BUSY FLAG
    If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
      UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
      UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
      UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
      COUNTER_L = COUNTER_L + 2
      INPUT_LATCH=INPUT_LATCH+1
    EndIf
    //TRIAL 2 FOR BUSY FLAG
    If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
      UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
      UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
      UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
      COUNTER_L = COUNTER_L + 2
      INPUT_LATCH=INPUT_LATCH+1
    EndIf
    //TRIAL 3 FOR BUSY FLAG
    If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
      UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
      UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
      UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
      COUNTER_L = COUNTER_L + 2
      INPUT_LATCH=INPUT_LATCH+1
    EndIf
    //TRIAL 4 FOR BUSY FLAG
    If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
      UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
      UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
      UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
      COUNTER_L = COUNTER_L + 2
      INPUT_LATCH=INPUT_LATCH+1
    EndIf
    //TRIAL 5 FOR BUSY FLAG
    If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
      UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
      UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
      UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
      COUNTER_L = COUNTER_L + 2
      INPUT_LATCH=INPUT_LATCH+1
    EndIf
    //6
```

```
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
  UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 7 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
  UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 8 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
  UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//9
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
  UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 10 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
  UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 11 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
  UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 12 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
  UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 13 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
  UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 14 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
  UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
```

```
//TRIAL 15 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
  UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 16 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
  UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 17 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
  UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 18 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
  UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//19
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
  UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 20 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
  UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 21 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
  UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//22
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
  UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 23 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
  UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
```

```
  EndIf
  //TRIAL 24 FOR BUSY FLAG
  If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
    UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
    UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
    UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
    COUNTER_L = COUNTER_L + 2
    INPUT_LATCH=INPUT_LATCH+1
  EndIf
  //TRIAL 25 FOR BUSY FLAG
  If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_INP_SIZE+1)
    UCS_ADDRESS_L = SLAVE_INP_IRAM+COUNTER_L/2
    UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
    UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
    COUNTER_L = COUNTER_L + 2
    INPUT_LATCH=INPUT_LATCH+1
  EndIf

EndWhile
//*********************************************************************************************
//Read data from UCS into PMAC

COUNTER_L = 0
While (UCS_BUSY_L = TRUE)
EndWhile
UCS_ADDRESS_L = UCS_READ + SLAVE_OUT_IRAM
While (COUNTER_L < SLAVE_OUT_SIZE+1)
  OUTPUT_LATCH=FALSE
  //WE DON'T WANT TO SKIP OUT OF HERE SO WE USE MULTIPLE IF STATEMENTS TO MAX BANDWIDTH
  //TRIAL 1 FOR BUSY FLAG
  If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
    L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
    L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
    COUNTER_L = COUNTER_L + 2
    OUTPUT_LATCH=OUTPUT_LATCH+1
  EndIf
  //TRIAL 2 FOR BUSY FLAG
  If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
    L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
    L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
    COUNTER_L = COUNTER_L + 2
    OUTPUT_LATCH=OUTPUT_LATCH+1
  EndIf
  //TRIAL 3 FOR BUSY FLAG
  If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
    L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
    L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
    COUNTER_L = COUNTER_L + 2
    OUTPUT_LATCH=OUTPUT_LATCH+1
  EndIf
  //TRIAL 4 FOR BUSY FLAG
  If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
    L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
    L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
    COUNTER_L = COUNTER_L + 2
    OUTPUT_LATCH=OUTPUT_LATCH+1
  EndIf
  //TRIAL 5 FOR BUSY FLAG
  If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
    L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
    L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
    COUNTER_L = COUNTER_L + 2
    OUTPUT_LATCH=OUTPUT_LATCH+1
  EndIf
  //TRIAL 6 FOR BUSY FLAG
  If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
    L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
    L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
    COUNTER_L = COUNTER_L + 2
    OUTPUT_LATCH=OUTPUT_LATCH+1
  EndIf
```

```
//7
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 8 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 9 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 10 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 11 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 12 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 13 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 14 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 15 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 16 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 17 FOR BUSY FLAG
```

```
    If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
      L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
      L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
      COUNTER_L = COUNTER_L + 2
      OUTPUT_LATCH=OUTPUT_LATCH+1
    EndIf
    //TRIAL 18 FOR BUSY FLAG
    If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
      L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
      L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
      COUNTER_L = COUNTER_L + 2
      OUTPUT_LATCH=OUTPUT_LATCH+1
    EndIf
    //19
    If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
      L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
      L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
      COUNTER_L = COUNTER_L + 2
      OUTPUT_LATCH=OUTPUT_LATCH+1
    EndIf
    //TRIAL 20 FOR BUSY FLAG
    If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
      L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
      L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
      COUNTER_L = COUNTER_L + 2
      OUTPUT_LATCH=OUTPUT_LATCH+1
    EndIf
    //TRIAL 21 FOR BUSY FLAG
    If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
      L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
      L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
      COUNTER_L = COUNTER_L + 2
      OUTPUT_LATCH=OUTPUT_LATCH+1
    EndIf
    //TRIAL 22 FOR BUSY FLAG
    If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
      L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
      L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
      COUNTER_L = COUNTER_L + 2
      OUTPUT_LATCH=OUTPUT_LATCH+1
    EndIf
    //TRIAL 23 FOR BUSY FLAG
    If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
      L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
      L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
      COUNTER_L = COUNTER_L + 2
      OUTPUT_LATCH=OUTPUT_LATCH+1
    EndIf
    //TRIAL 24 FOR BUSY FLAG
    If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
      L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
      L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
      COUNTER_L = COUNTER_L + 2
      OUTPUT_LATCH=OUTPUT_LATCH+1
    EndIf
    //TRIAL 25 FOR BUSY FLAG
    If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
      L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
      L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
      COUNTER_L = COUNTER_L + 2
      OUTPUT_LATCH=OUTPUT_LATCH+1
    EndIf

  EndWhile
  TEMP_UPDATE2=TEMP_UPDATE1
  TEMP_UPDATE1=SERVO_COUNTER
  I_O_UPDATE_TIME=ABS(TEMP_UPDATE2-TEMP_UPDATE1)*(I10/8388608)
  ;*************************************************************************
Else
  ENABLE PLC13
  DISABLE PLCC13
```

```
EndIf


CLOSE
;*****************************************************************************

ena plc PLC_NUMBER
```

APPENDIX B – PROFIBUS MASTER FOR UMAC

UmacProfibusMaster.h

```
CLOSE
DELETE GATHER

; -- Define UNet Service PLC

#DEFINE PLC_NUMBER            13

; -- Definitions for Slave/Master IO Blocks

#DEFINE UCS_NODE_NUMBER       5              ;STATION NUMBER(PFB)

//INPUTS TO NETWORK FROM UCS
#DEFINE MASTER_INP_SIZE              16             ; Number of master inputs in bytes
#DEFINE MASTER_INPUT_STARTADDRESS $4C00

//OUTPUTS FROM NETWORK TO UCS
#DEFINE MASTER_OUT_SIZE       16             ; NUMBER OF MASTER OUTPUTS
#DEFINE MASTER_OUTPUT_STARTADDRESS $C00

//UMAC ADDRESSES DIFFERENT FOR MACRO
#DEFINE UCS_DATALO_ADDR               $78D00
#DEFINE UCS_DATAHI_ADDR               $78D01
#DEFINE UCS_ADDRESS_ADDR              $78D02
#DEFINE UCS_CONTROL_STATUS_ADDR       $78D03


#include "ProfibusMasterMvars.h"
#include "ProfibusMasterServiceHeader.h"
#include "ProfibusMasterServiceEvent.plc"
```

ProfibusMasterServiceHeader.h

```
#DEFINE SLAVE                 0              ;SET TO 1 IF IT IS A SLAVE
#DEFINE DEVICENET             0
#DEFINE UCS_BAUDRATE          125
#DEFINE SLAVE_INP_SIZE        200            ; Number of slave inputs in bytes
#DEFINE SLAVE_OUT_SIZE        200            ; Number of slave outputs

;       The command variable

#DEFINE IDLE                  $0000
#define RESET                 $0001          //PERFORMS A SOFT RESET
#define UCS_OPEN              $0002          //SENDS OPEN TRIGGER AND STARTS TRANSFERS
#define UCS_CLOSE             $0003          //SENDS CLOSE TRIGGER
#DEFINE TRANSFER              $0004          //READ ONLY FOR USER
#define LATCH_ERR             $0005
#define UCS_CMD_IDBLOCK       $0100
#define UCS_CMD_CSBLOCK       $0101
#define UCS_CMD_ERRORLOG      $0102
#define UCS_CMD_SENDTRIGGER   $1000          //USED FOR TRIGGER HANDLER STARTER

//FIRST BYTE OF UCS_STATUS_CONTROL IS STATUS FROM UCS AND 2ND BYTE IS CONTROL TO UCS
#DEFINE UCS_BUSY      UCS_STATUS&1                  ;BIT 0 OF STATUS IS BUSY FLAG
#DEFINE UCS_BUSY_L    UCS_STATUS_L&1                ;BIT 0 OF STATUS IS BUSY FLAG

#DEFINE UCS_IRQ       UCS_STATUS&2                  ;BIT 1 OF STATUS IS IRQ FROM UCS
#DEFINE UCS_CMD UCS_COMMAND&$F
#DEFINE UCS_CMD_L     UCS_COMMAND_L&$F

//THESE MEMORY LOCATIONS ARE SET AND ARE CONSISIENT WITH PROFIMASTER DEFAULTS IN CFGHOST
#define SLAVE_INP_IRAM        $300                  ; IRAM location of slave inputs-SELF DEFINED
#define SLAVE_OUT_IRAM        $1300                 ; IRAM location of slave outputs-SELF
DEFINED
#DEFINE IRQ_CLEAR             $2
#DEFINE SEND_INT              $2
```

```
#DEFINE OPEN_EVENT            $1
#DEFINE TRUE                  1
#DEFINE ON                    1
#DEFINE FALSE                 0
#DEFINE OFF                   0
#DEFINE RUN_MODE              3
        ;READ IN MODULE STATUS WORD
#DEFINE MS_CONVERT            8388608/I10
#DEFINE MODULE_STATUS_ADDR    $0
#DEFINE UCS_STATUS_ADDR       $21
#DEFINE EVENT_IN_ADDR         $26
#DEFINE EVENT_OUT_ADDR        $27
#DEFINE TRIGGER_IN_ADDR       $28
#DEFINE TRIGGER_OUT_ADDR      $29
#DEFINE UCS_READ              $8000

;*************************************************************************
//
#DEFINE TIMER                 I5511                 //USES TIMER FROM CS5

#DEFINE UCS_DATALO            M8000
#DEFINE UCS_DATAHI            M8001
#DEFINE UCS_ADDRESS           M8002
#DEFINE UCS_DATALO_L          L8000
#DEFINE UCS_DATAHI_L          L8001
#DEFINE UCS_ADDRESS_L         L8002
#DEFINE UCS_CONTROL_STATUS    M8003
#DEFINE UCS_CONTROL           M8004
#DEFINE UCS_STATUS            M8005
#DEFINE UCS_STATUS_L          L8005

#DEFINE MODULE_STATUS_LO      M8010
#DEFINE MODULE_STATUS_HI      M8011
#DEFINE UCS_STATUS_LO         M8012
#DEFINE UCS_STATUS_HI         M8013
#DEFINE TRIGGERIN_POINTER_LO  M8014
#DEFINE TRIGGERIN_POINTER_HI  M8015
#DEFINE TRIGGERIN_POINTER     M8016
#DEFINE EVENTIN_POINTER_LO    M8017
#DEFINE EVENTIN_POINTER_HI    M8018
#DEFINE EVENTOUT_POINTER_LO   M8019
#DEFINE EVENTOUT_POINTER_HI   M8020
#DEFINE EVENT_LO              M8021
#DEFINE EVENT_HI              M8022
#DEFINE BAUD                  M8023
#DEFINE NODE_NUMBER           M8024
#DEFINE COUNTER               M8025
#DEFINE COUNTER_L             L8025
#DEFINE INPUT_LATCH           L8026
#DEFINE OUTPUT_LATCH          L8027

#DEFINE I_O_UPDATE_TIME              M8028
#DEFINE SERVO_COUNTER         M8029
#DEFINE TEMP_UPDATE1          M8030
#DEFINE TEMP_UPDATE2          M8031

#DEFINE UCS_COMMAND           M8050
#DEFINE UCS_COMMAND_L         L8050

; -- PMAC Specific Memory Addresses

; -- PMAC Specific Memory Addresses M-VAR DEFS
UCS_DATALO->X:UCS_DATALO_ADDR,24                    //M8000
UCS_DATAHI->X:UCS_DATAHI_ADDR,24                    //M8001
UCS_ADDRESS->X:UCS_ADDRESS_ADDR,24                  //M8002
UCS_DATALO_L->X:UCS_DATALO_ADDR,24                  //L8000
UCS_DATAHI_L->X:UCS_DATAHI_ADDR,24                  //L8001
UCS_ADDRESS_L->X:UCS_ADDRESS_ADDR,24                //L8002
UCS_CONTROL_STATUS->X:UCS_CONTROL_STATUS_ADDR,24    //M8003
UCS_CONTROL->X:UCS_CONTROL_STATUS_ADDR,8,8          //M8004
UCS_STATUS->X:UCS_CONTROL_STATUS_ADDR,0,8           //M8005
```

```
UCS_STATUS_L->X:UCS_CONTROL_STATUS_ADDR,0,8          //L8005

UCS_COMMAND->Y:$10F0,0,24                            //M8050
UCS_COMMAND_L->Y:$10F0,0,24                          //L8050
MODULE_STATUS_LO->Y:$10F1,0,24                              //M8010
MODULE_STATUS_HI->X:$10F1,0,24                              //M8011
UCS_STATUS_LO->Y:$10F2,0,24                          //M8012
UCS_STATUS_HI->X:$10F2,0,24                          //M8013
TRIGGERIN_POINTER_LO->Y:$10F3,0,24                   //M8014
TRIGGERIN_POINTER_HI->X:$10F3,0,24                   //M8015
TRIGGERIN_POINTER->Y:$10F4,0,24                             //M8016
EVENTIN_POINTER_LO->Y:$10F5,0,24                            //M8017
EVENTIN_POINTER_HI->X:$10F5,0,24                            //M8018
EVENTOUT_POINTER_LO->Y:$10F6,0,24                    //M8019
EVENTOUT_POINTER_HI->X:$10F6,0,24                    //M8020
EVENT_LO->Y:$10F7,0,24                               //M8021
EVENT_HI->X:$10F7,0,24                               //M8022
BAUD->Y:$10F8,0,24                                   //M8023
NODE_NUMBER->X:$10F8,0,24                            //M8024
//COUNTER->Y:$10F9,0,24                                     //M8025
COUNTER_L->Y:$10F9,0,24                                     //ML025

INPUT_LATCH->X:$10F9,0,24                                   //L8026
OUTPUT_LATCH->X:$10FA,0,24                           //L8027
I_O_UPDATE_TIME->Y:$10FA,0,24,S                             //M8028
SERVO_COUNTER->X:$0,0,24                                    //M8029
TEMP_UPDATE1->X:$10FB,0,24                           //M8030
TEMP_UPDATE2->Y:$10FB,0,24                           //M8031

L7998->Y:$7B58[1024]                 ;MUST BE A FACTOR OF 2
L7999->X:$7B58[1024]                 ;MUST BE A FACTOR OF 2
```

ProfibusMasterServiceEvent.plc

```
//SERVICE PLC FOR THE DATA TRANSFERS AND ERROR HANDLING
//SEE FLOW CHART ON UNET CD FOR LOGIC TRAIN

//IN=NETWORK IN FROM SLAVE
//OUT=NETWORK OUT TO SLAVE


OPEN PLC PLC_NUMBER CLEAR

;Check IRQ Status (INT FROM UCS)
If (UCS_IRQ != FALSE)         //CLEAR IRQ
UCS_STATUS=IRQ_CLEAR          //TELL UCS I SAW IRQ AND CLEAR IT

;*************************************************************************
;STORE MODULE STATUS AND CHECK IT FOR RUN MODE
While (UCS_BUSY = TRUE)
  ENDWHILE                    //WAIT UNTIL UCS IS NOT BUSY (1-8uSEC TYPICAL)
  UCS_ADDRESS = UCS_READ + MODULE_STATUS_ADDR       ;IRAM address for MODULE_STATUS
  While (UCS_BUSY = TRUE)
    ENDWHILE                  //WAIT UNTIL UCS IS NOT BUSY (1-8uSEC TYPICAL)

    MODULE_STATUS_LO = UCS_DATALO            //STORES MODULE STATUS AND CONTROL
    MODULE_STATUS_HI= UCS_DATAHI
    //*************************************************************************
    ;IF UCS IS NOT IS RUN MODE

    If (MODULE_STATUS_LO&$FF != RUN_MODE)           ;MODULE STATUS MODE IS LOW BYTE OF IRAM 0
    While (UCS_BUSY = TRUE)
    ENDWHILE
                                    //WAIT UNTIL UCS IS NOT BUSY (1-8uSEC TYPICAL)
    UCS_ADDRESS = UCS_READ + UCS_STATUS_ADDR ; IRAM address
    While (UCS_BUSY = TRUE)
    ENDWHILE
                                    //WAIT UNTIL UCS IS NOT BUSY (1-8uSEC TYPICAL)
    UCS_STATUS_LO = UCS_DATALO
        //STORES UCS_STATUS BITS
    UCS_STATUS_HI = UCS_DATAHI
```

```
UCS_COMMAND = LATCH_ERR
Else
    //UCS_COMMAND= IDLE
EndIf
  ;*****************************************************************************
  //PROCESS FOR EVENT HANDLER
  ///READ IN EVENT OUT POINTER/////////////////

  While (UCS_BUSY = TRUE)
  EndWhile
  UCS_ADDRESS = UCS_READ + EVENT_OUT_ADDR               ; Read EVENT_OUT
  While (UCS_BUSY = TRUE)
  EndWhile
  EVENTOUT_POINTER_LO = UCS_DATALO
  EVENTOUT_POINTER_HI = UCS_DATAHI

  ///READ IN EVENT IN POINTER/////////////////
  While (UCS_BUSY = TRUE)
  EndWhile
  UCS_ADDRESS = UCS_READ + EVENT_IN_ADDR         ; Read EVENT_IN
  While (UCS_BUSY = TRUE)
  EndWhile
  EVENTIN_POINTER_LO = UCS_DATALO
  EVENTIN_POINTER_HI = UCS_DATAHI

  //Compare EVENT_IN to EVENT_OUT//////////////

  If (EVENTIN_POINTER_LO > EVENTOUT_POINTER_LO)
    While (UCS_BUSY = TRUE)
    EndWhile
    UCS_ADDRESS = UCS_READ + EVENTOUT_POINTER_LO
    While (UCS_BUSY = TRUE)
    EndWhile
    EVENT_LO = UCS_DATALO
    EVENT_HI = UCS_DATAHI
    While (UCS_COMMAND = UCS_OPEN And EVENT_LO&$FF != OPEN_EVENT)
      While (UCS_BUSY = TRUE)
      EndWhile
      UCS_ADDRESS = UCS_READ + EVENTOUT_POINTER_LO
      While (UCS_BUSY = TRUE)
      EndWhile
      EVENT_LO = UCS_DATALO
      EVENT_HI = UCS_DATAHI
    EndWhile

    EVENTOUT_POINTER_LO = EVENTOUT_POINTER_LO + 1
    If (EVENTOUT_POINTER_LO = $200)
      EVENTOUT_POINTER_LO=$100
    EndIf

    While (UCS_BUSY = TRUE)
    EndWhile
    UCS_ADDRESS = EVENT_OUT_ADDR
    While (UCS_BUSY = TRUE)
    EndWhile
    UCS_DATALO = EVENTOUT_POINTER_LO
    UCS_DATAHI = 0
    While (UCS_BUSY = TRUE)
    EndWhile
    If ((EVENT_LO&$FF) != OPEN_EVENT)
      UCS_COMMAND=LATCH_ERR
    Else
      ENABLE PLCC13
      UCS_COMMAND = TRANSFER
      DISABLE PLC13
    EndIf
  EndIf
  //

  ;*****************************************************************************
```

```
      ELSE                    //IRQ NOT PRESENT
      ;*************************************************************************
      ;COMMAND RESET
      If (UCS_CMD = RESET)
        UCS_CONTROL=RESET     // toggle lsb of control byte for reset
        TIMER=10*8388608/I10                                  //NEED TO WAIT MIN OF 50usec
according to client.pdf
        While (TIMER > 0)
        EndWhile       //THIS VALUE IS A CONSERVATIVE 2 SERVO CYCLES
        UCS_STATUS=$0E //INTO STATUS CLEAR ALL FAULTS--WHY SEPARATE???
        UCS_CONTROL=0          //SET CONTROL TO ZERO
        While (UCS_IRQ = FALSE)
        EndWhile
        UCS_STATUS=IRQ_CLEAR
        UCS_COMMAND=IDLE
        TRIGGERIN_POINTER=0
      EndIf
      ;*************************************************************************
; Send a CLOSE Trigger

If (UCS_CMD = UCS_CLOSE)

  //READ TRIGGER IN POINTER
  While (UCS_BUSY = TRUE)
  EndWhile
  UCS_ADDRESS = UCS_READ + TRIGGER_IN_ADDR          ; Fetch TRIGGER_IN pointer
  While (UCS_BUSY = TRUE)
  EndWhile
  TRIGGERIN_POINTER_LO = UCS_DATALO
  TRIGGERIN_POINTER_HI = UCS_DATAHI
  While (TRIGGERIN_POINTER_LO < TRIGGERIN_POINTER)
    While (UCS_BUSY = TRUE)
    EndWhile
    UCS_ADDRESS = UCS_READ + TRIGGER_IN_ADDR        ; Fetch TRIGGER_IN pointer
    While (UCS_BUSY = TRUE)
    EndWhile
    TRIGGERIN_POINTER_LO = UCS_DATALO
    TRIGGERIN_POINTER_HI = UCS_DATAHI
    P8003=2
  EndWhile
//STUFF IN A CLOSE TRIGGER
While (UCS_BUSY = TRUE)
EndWhile
//
UCS_ADDRESS = TRIGGERIN_POINTER_LO ; Setup address to write to current TRIGGER_IN
While (UCS_BUSY = TRUE)
EndWhile

UCS_DATALO = $0104                          ; Trigger command = CLOSE + OPERATOR SHUTDOWN SEE
CLIENT FOR DETAILS
UCS_DATAHI = $0000

TRIGGERIN_POINTER=TRIGGERIN_POINTER_LO+1
If (TRIGGERIN_POINTER = $300)
  TRIGGERIN_POINTER=$200
EndIf
While (UCS_BUSY = TRUE)
EndWhile
UCS_ADDRESS = TRIGGER_IN_ADDR
While (UCS_BUSY = TRUE)
EndWhile
UCS_DATALO     =      TRIGGERIN_POINTER
UCS_DATAHI     =      $0
While (UCS_BUSY = TRUE)
EndWhile
UCS_CONTROL = SEND_INT               ;SEND INTERUPT TO PROCESS TRIGGER DATA
While (UCS_IRQ = FALSE)
EndWhile
UCS_COMMAND = IDLE     ; Enable post TRIGGER Handler
EndIf
```

```
;*************************************************************************
; Send an OPEN Trigger

If (UCS_CMD = UCS_OPEN)
  If (SLAVE = TRUE)
    ;FIRST WRITE TO INPUT DEFINITION-IF IT IS A MASTER PERSONALITY WILL OVERWRITE IT
    While (UCS_BUSY = TRUE)
    EndWhile
    UCS_ADDRESS = $0022                                              ;ADDRESS FOR INPUT DEF
    While (UCS_BUSY = TRUE)
    EndWhile
    UCS_DATALO = SLAVE_INP_SIZE
    UCS_DATAHI = SLAVE_INP_IRAM
    ;NEXT WRITE TO OUTPUT SIZE AND LOCATION-MASTER'S PERSONALITY WILL OVERWRITE IT
    While (UCS_BUSY = TRUE)
    EndWhile
    UCS_ADDRESS = $0023                                              ;ADDRESS FOR OUTPUT
DEF
    While (UCS_BUSY = TRUE)
    EndWhile
    UCS_DATALO = SLAVE_OUT_SIZE
    UCS_DATAHI = SLAVE_OUT_IRAM
  EndIf
  ; Sending open trigger and handle event queue
  While (UCS_BUSY = TRUE)
  EndWhile
  UCS_ADDRESS = UCS_READ + TRIGGER_IN_ADDR                          ;READ + ADDRESS FOR TRIGGER
IN POINTER
  While (UCS_BUSY = TRUE)
  EndWhile
  TRIGGERIN_POINTER_LO = UCS_DATALO
  TRIGGERIN_POINTER_HI = UCS_DATAHI
  While (TRIGGERIN_POINTER_LO < TRIGGERIN_POINTER)
    While (UCS_BUSY = TRUE)
    EndWhile
    UCS_ADDRESS = UCS_READ + TRIGGER_IN_ADDR        ; Fetch TRIGGER_IN pointer
    While (UCS_BUSY = TRUE)
    EndWhile
    TRIGGERIN_POINTER_LO = UCS_DATALO
    TRIGGERIN_POINTER_HI = UCS_DATAHI
  EndWhile
  //increment trigger in pointer
  While (UCS_BUSY = TRUE)
  EndWhile
  //write the trigger
  UCS_ADDRESS = TRIGGERIN_POINTER_LO ; Setup address to write to current TRIGGER_IN
  While (UCS_BUSY = TRUE)
  EndWhile
        If (SLAVE = TRUE)
        UCS_DATALO = $0701            ;7->DISABLES C.O.S; ENABLES PLUG AND PLAY; AND SWITCH DATA
SUPPLIED(BAUD RATE ADN NODE NUMBER)
        Else
        UCS_DATALO = $0401
        EndIf
  ;1->OPEN TRIGGER
  If (DEVICENET = TRUE And SLAVE = TRUE)
    If (UCS_BAUDRATE = 125)
      BAUD=$00
    EndIf
    If (UCS_BAUDRATE = 250)
      BAUD=$40
    EndIf
    If (UCS_BAUDRATE = 500)
      BAUD=$80
    EndIf
  Else
    BAUD=$00
  EndIf

  UCS_DATAHI = BAUD + (UCS_NODE_NUMBER & $3F)
  TRIGGERIN_POINTER=TRIGGERIN_POINTER_LO+1
```

```
  If (TRIGGERIN_POINTER = $300)
    TRIGGERIN_POINTER=$200
  EndIf

  While (UCS_BUSY = TRUE)
  EndWhile
  UCS_ADDRESS = TRIGGER_IN_ADDR
  While (UCS_BUSY = TRUE)
  EndWhile
  UCS_DATALO = TRIGGERIN_POINTER
  UCS_DATAHI = 0
  While (UCS_BUSY = TRUE)
  EndWhile
  //send INT TO UCS
  UCS_CONTROL = SEND_INT                    ;SEND INTERUPT TO PROCESS TRIGGER DATA
  TIMER=2500*8388608/I10
  While (TIMER > 0)
  EndWhile
  While (UCS_IRQ = FALSE)
       EndWhile
  EndIf               ;END OF OPEN TRIGGER

  ;*************************************************************************
  //IF CARD IS WATCHDOGGED THEN SEND A RESET AUTOMATICALLY
  If (UCS_STATUS&$8 = $8)
    UCS_COMMAND=RESET
  EndIf
EndIf
close




;*************************************************************************

//PLCC code for transfers
OPEN PLCC PLC_NUMBER CLEAR
If (UCS_CMD_L = TRANSFER And UCS_STATUS_L&$8 != $8)// And(UCS_IRQ = FALSE)
  ;*************************************************************************
  ; -- Output the data to NETWORK from PMAC
  ;*************************************************************************
  ;                    -- Master Mode
  COUNTER_L = 0
  While (UCS_BUSY_L = TRUE)
  EndWhile
  UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4 ;$1300
  While (COUNTER_L < MASTER_INP_SIZE+1)
    //WE DON'T WANT TO SKIP OUT OF HERE SO WE USE MULTIPLE IF STATEMENTS TO MAX BANDWIDTH
    //TRIAL 1 FOR BUSY FLAG
    If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
      UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
      UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
      UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
      COUNTER_L = COUNTER_L + 2
      INPUT_LATCH=INPUT_LATCH+1
    EndIf
    //TRIAL 2 FOR BUSY FLAG
    If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
      UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
      UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
      UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
      COUNTER_L = COUNTER_L + 2
      INPUT_LATCH=INPUT_LATCH+1
    EndIf
    //TRIAL 3 FOR BUSY FLAG
    If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
      UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
      UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
      UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
      COUNTER_L = COUNTER_L + 2
```

```
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 4 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
  UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 5 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
  UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 6 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
  UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 7 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
  UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 8 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
  UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 9 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
  UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 10 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
  UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 11 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
  UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 12 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
  UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
```

```
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 13 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
  UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 14 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
  UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 15 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
  UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 16 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
  UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 17 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
  UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 18 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
  UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 19 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
  UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 20 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
  UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
  UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
  COUNTER_L = COUNTER_L + 2
  INPUT_LATCH=INPUT_LATCH+1
EndIf
//TRIAL 21 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
  UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
  UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
```

```
    UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
    COUNTER_L = COUNTER_L + 2
    INPUT_LATCH=INPUT_LATCH+1
  EndIf
  //TRIAL 22 FOR BUSY FLAG
  If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
    UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
    UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
    UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
    COUNTER_L = COUNTER_L + 2
    INPUT_LATCH=INPUT_LATCH+1
  EndIf
  //TRIAL 23 FOR BUSY FLAG
  If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
    UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
    UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
    UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
    COUNTER_L = COUNTER_L + 2
    INPUT_LATCH=INPUT_LATCH+1
  EndIf
  //TRIAL 24 FOR BUSY FLAG
  If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
    UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
    UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
    UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
    COUNTER_L = COUNTER_L + 2
    INPUT_LATCH=INPUT_LATCH+1
  EndIf
  //TRIAL 25 FOR BUSY FLAG
  If (UCS_BUSY_L = FALSE And COUNTER_L < MASTER_INP_SIZE+1)
    UCS_ADDRESS_L = MASTER_INPUT_STARTADDRESS/4+COUNTER_L/2
    UCS_DATALO_L = L7998[COUNTER_L]& $FFFF
    UCS_DATAHI_L = L7998[COUNTER_L+1]& $FFFF
    COUNTER_L = COUNTER_L + 2
    INPUT_LATCH=INPUT_LATCH+1
  EndIf

EndWhile
;*************************************************************************
;                    -- Master Mode
COUNTER_L = 0
While (UCS_BUSY_L = TRUE)
EndWhile
UCS_ADDRESS_L = UCS_READ + MASTER_OUTPUT_STARTADDRESS/4 ;$300
While (COUNTER_L < MASTER_OUT_SIZE+1)
  //WE DON'T WANT TO SKIP OUT OF HERE SO WE USE MULTIPLE IF STATEMENTS TO MAX BANDWIDTH
  //TRIAL 1 FOR BUSY FLAG
  If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
    L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
    L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
    COUNTER_L = COUNTER_L + 2
    OUTPUT_LATCH=OUTPUT_LATCH+1
  EndIf
  //TRIAL 2 FOR BUSY FLAG
  If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
    L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
    L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
    COUNTER_L = COUNTER_L + 2
    OUTPUT_LATCH=OUTPUT_LATCH+1
  EndIf
  //TRIAL 3 FOR BUSY FLAG
  If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
    L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
    L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
    COUNTER_L = COUNTER_L + 2
    OUTPUT_LATCH=OUTPUT_LATCH+1
  EndIf
  //TRIAL 4 FOR BUSY FLAG
  If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
    L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
    L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
```

```
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 5 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 6 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 7 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 8 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 9 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 10 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 11 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 12 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 13 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 14 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
```

```
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 15 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 16 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 17 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 18 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 19 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 20 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 21 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 22 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 23 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
EndIf
//TRIAL 24 FOR BUSY FLAG
If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
  L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
  L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
  COUNTER_L = COUNTER_L + 2
  OUTPUT_LATCH=OUTPUT_LATCH+1
```

```
    EndIf
    //TRIAL 25 FOR BUSY FLAG
    If (UCS_BUSY_L = FALSE And COUNTER_L < SLAVE_OUT_SIZE+1) //And OUTPUT_LATCH = FALSE)
      L7999[COUNTER_L] = UCS_DATALO_L & $FFFF
      L7999[COUNTER_L+1] = UCS_DATAHI_L & $FFFF
      COUNTER_L = COUNTER_L + 2
      OUTPUT_LATCH=OUTPUT_LATCH+1
    EndIf

  EndWhile
  TEMP_UPDATE2=TEMP_UPDATE1
  TEMP_UPDATE1=SERVO_COUNTER
  I_O_UPDATE_TIME=ABS(TEMP_UPDATE2-TEMP_UPDATE1)*(I10/8388608)
  ;*************************************************************************
Else
  ENABLE PLC13
  DISABLE PLCC13
EndIf


CLOSE
;*****************************************************************************

ena plc PLC_NUMBER
```