DT9000
Development Kit
V1.1

6 Clock Tower Place
Suite 100
Maynard, MA 01754
USA

Tel: (866) 837-1931
Tel: (978) 461-1140
FAX: (978) 461-1146

http://www.diamondt.com/

Liability

Diamond Technologies Inc. shall not be liable for technical or editorial errors or omissions contained herein, nor for incidental or consequential damages resulting from the use of this material. Those responsible for the use of this device must ensure that all necessary steps have been taken to verify that the applications meet all performance and safety requirements including any applicable laws, regulations, codes, and standards.

There are many applications of this product. The examples and illustrations in this document are included solely for illustrative purposes. Because of the many variables and requirements associated with any particular implementation, Diamond Technologies Inc. cannot assume responsibility for actual use based on these examples and illustrations.

Diamond Technologies Inc., reserves the right to modify our products in line with our policy of continuous product development. The information in this document is subject to change without notice and should not be considered as a commitment by Diamond Technologies Inc.

Intellectual Property Rights

Diamond Technologies Inc. has intellectual property rights relating to technology embodied in the product described in this document. These intellectual property rights may include patents and pending patent applications in the US and other countries.

Diamond Technologies Inc. and the Diamond Technologies logo are trademarks of Diamond Technologies Inc. All other trademarks are the property of their respective holders.

i Revision History

Version	Date	Description
1.0	7/4/2015	Original Version
1.1	7/17/2015	Updated for Development Kit Version 1.0.2

ii Reference Documents

iii Table of Contents

1.0 Overview

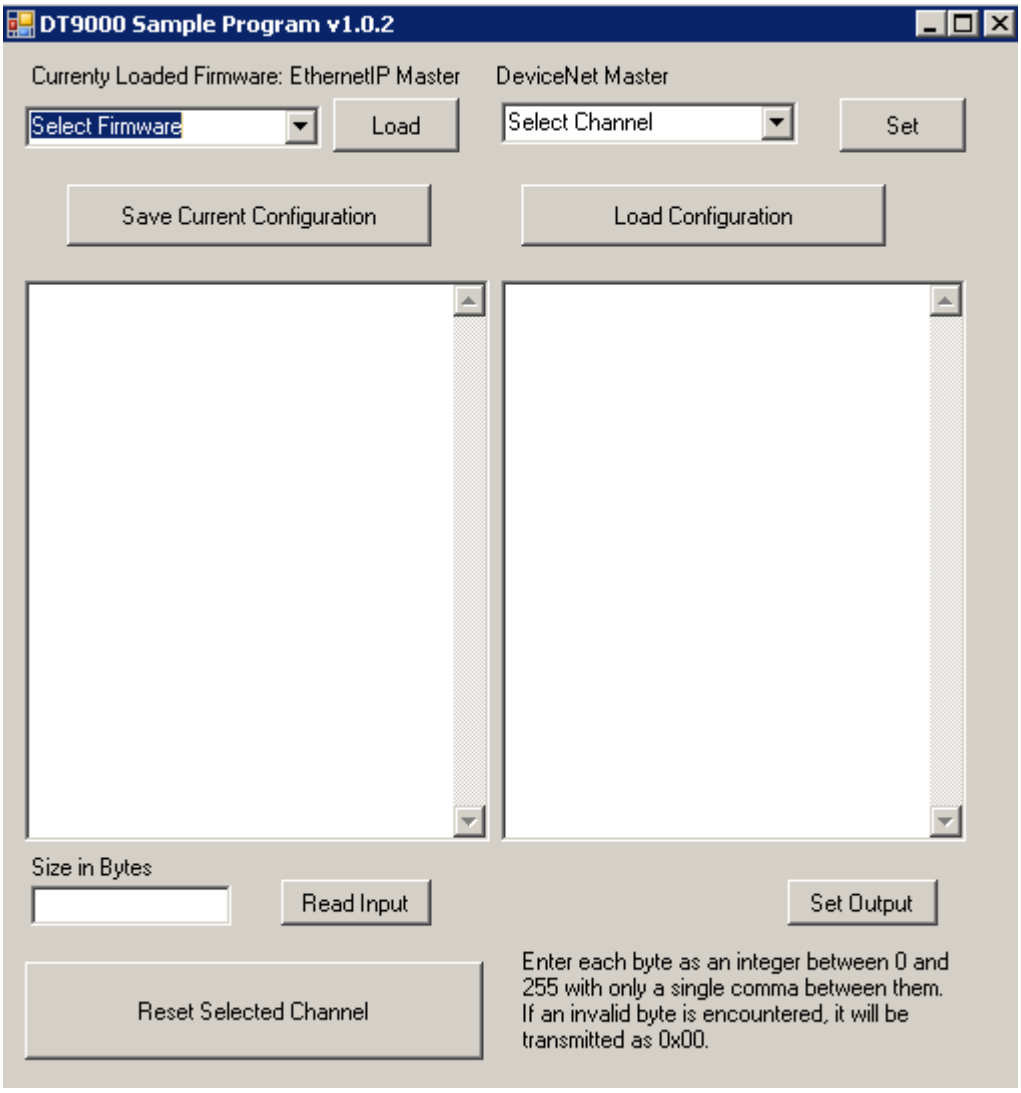
The DT9000 Development kit consists of two DLL files that can be used to easily communicate with the Industrial Network Master driver from any .NET application. The first file, **DT_IndustrialMaster.dll** must be added as a resource file to your .NET project. The second file, **cifx_interface.dll** just needs to be in the same directory as the first DLL file.

The Development kit also includes an example program, which is a very simple example of what can be done with the DT9000 master. All source code for the sample program is included.

2.0 Master Example Program Overview

The master sample interface consists of a single form. It allows the user to change the firmware, select a communication channel to use, view input and set output.

Note: The DT9000 Master Sample application must be run as an administrator. Reading and switching the firmware requires manipulation of the Windows device registry. If your application is not running as an administrator, you will get exceptions when trying to perform these actions.



The *Select Firmware* combo-box can be used to select the firmware you wish to load into the master device. Once you have firmware loaded, the *Select Channel* combo-box will populate with the available communication channels (firmware and channel selection is explained in detail later in this document).

Note: Loading new firmware will clear the current configuration from the device. This is intentional, as the configuration for one network will not work on any other. In your application you should consider using the provided methods to save the currently loaded configuration file before switching the firmware. In this application, you will have to download the configuration using SYCON.NET each time you change the firmware.

Once a channel has been selected, you can use the *Read Input* and *Set Output* functions to view and change the data on the network.

You can also use the *Save Current Configuration* and *Load Configuration* buttons to save or load the configuration associated with the selected channel. Depending on the network, the configuration will be either one or two *.nxd* files. When loading a configuration with multiple files, the user must select both files from the file dialog window.

To set output data, enter each byte as an integer between 0 and 255 separated by commas. (eg. 0,5,99, 255). Any bytes that are not properly formatted will be sent as a 0 and the rest of the data will not be affected.

The *Reset Selected Channel* button will reset the hardware associated with the currently selected channel. This is useful when working with some networks because if your slave device is reset, the master may mark it as *Failed* until the bus resets. This button will allow you to reset the network without restarting the DT9000 or changing the firmware.

2.1 Installation and Running Details

In its current form, this application MUST be run as an Administrator. This is because of the registry modifications required to change the device firmware. Failure to run the application as an administrator could cause exceptions to be thrown indicating that the application cannot perform the expected operations.

As previously mentioned, the application requires 2 DLL files to run. They both must exist in the same directory, though only the **DT_IndustrialMaster.dll** needs to be added as a resource to your .NET project. In addition to these files, the included directory, cifX Firmware must be in the same directory as the application executable. This directory contains the firmware files to load on to the device.

2.2 Main Classes Overview

There are three main classes that you will use to control the DT9000 Master. They will be available to you if you include the **DT_IndustrialMaster.dll** in your application. First is the **MasterDevice** class. Each instance of this class represents a master hardware device in your DT9000. A static method can be used to discover the number of devices in your DT9000 and you can then create instances of this class using the device number. If you have one device, you will only have a device **0**. A DT9000 with 2 devices will be able to instantiate a device **0** and device **1.** Using a device number higher than the number of devices minus 1 will result in an exception. In a multiple-device DT9000, each device will have different hardware options. It is possible to have an open communication channel from each device simultaneously.

The **MasterDevice** class is mainly used to gather information and change the settings of the hardware. This includes reading and loading firmware, loading configuration, getting the hardware serial, device name, available network hardware and opening communication channels.

The **ComChannel** class is used to actually communicate over the network. Each MasterDevice can have up to 8 communication channels which represent different areas of the dual-port memory on the master device. The only channel covered by this document is channel 0, the communication channel. It can be used to read input and output byte arrays and write to the output byte array. For information on

other communication channels, see the document *netX Dual-Port Memory Interface DPM 12 EN.pdf*, included on the *DT9000 Quick Start CD.*

Finally, there is the class **FirmwareInfo**. This class is provided as a convenient way to get information about firmware options. Each FirmwareInfo object contains a display name (eg. Ethernet IP), the name of the corresponding firmware file (eg. cifxeim.nxf) and the integer code for the required hardware to use the firmware. Each MasterDevice object will provide you with a list of compatible FirmwareInfo objects, and a list of all FirmwareInfo objects. It is important to note that these objects do not actually contain the firmware in any way. They contain the base filename for the firmware- you will still have to point your application to the directory the firmware is stored in.

3.0 MasterDevice class

Each instance of the master device class corresponds to a physical master device within the DT9000.

3.2 Property: Int SerialNumber

This is a read-only property containing the serial number of the device. This is populated when the object is constructed and is used to determine the registry and filepath relating to this particular device. Most of these operations are already implemented by the MasterDevice class.

3.3 Property: Int DeviceNumber

This is a read-only property containing the DeviceNumber (product number) of the device. It will be the same for all hardware devices. It is used to determine the registry and filepath relating to this particular device. Most of these operations are already implemented by the MasterDevice class.

3.4 Property: String BoardName

This is a read-only property containing the text name of the device. Each device will have a text name in the form of cifx#. The number included in the name is not necessarily the same as the board number and should not be used as such. The board name is used in some low level hardware operations to address a particular device. Most of these operations are already implemented by the MasterDevice class.

3.5 Property: Int BoardNumber

This is a read-only property containing the board number used to instantiate this MasterDevice object. It is used by some low level hardware operations to address this particular device. Most of these operations are already implemented by the MasterDevice class.

3.5 Static Method: int getNumberDevices()

This method returns the number of master devices included in the DT9000. With the current DT9000 hardware this will be either 1 or 2. Using the device number, you can create instances of the MasterDevice class. Each master device has a board number from 0 through (totalDevices-1). This number must be passed to the MasterDevice constructor to indicate which device to point to. For most applications, you will want to create and maintain a master device object for each hardware device.

3.6 Constructor: MasterDevice(int devNumber)

PARAMETERS	
Int devNumber	The board number of the device this instance will point to.
RETURN	
N/A	

The constructor will create an instance of the MasterDevice class corresponding to the hardware device indicated by the devNumber passed to the method. The devNumbers are always sequential starting at 0, so you can use the number of devices obtained from the getNumberDevices() method to instantiate an object for each of your devices.

The constructor will populate all of the public Property fields mentioned above (3.1-3.4) but it WILL NOT populate the *hardware options* array (see section 3.8). Discovering the hardware options is a long operation (1-4 seconds) and it disrupts bus communication. We have decided to separate it from the other information gathering functions to allow the programmer to choose when this operation will be most appropriate for their particular application.

It is also advised that the constructor be contained in a try-catch block. An exception will be thrown if the constructor is unable to obtain a driver handle. Only one application may hold a driver handle at a time and if the handle is not explicitly closed, no other applications will be able to access the driver until the device is restarted.

This can occur if an application using the driver crashes before it is able to close the handle. We have also observed that SYCON.NET will sometimes not properly release the driver handle even when it exits gracefully.

If you are having problems with exceptions thrown in the constructor, the first thing to do is restart the DT9000 to ensure that no other application is blocking your use of the hardware driver.

3.7 Method: void setCifxDriverPath(string newPath)

PARAMETERS	
String newPath	The filepath the MasterDevice object will use to find the cifX driver
RETURN	
N/A	

This method was included for compatibility with other systems. The MasterDevice object will look in the default DT9000 install location for the hardware driver, unless a different install location is set using this method.

3.8 Method: int[] readHardwareOptions()

PARAMETERS	
N/A	
RETURN	

Int[4] HardwareOptions	An array of the 4 hardware options available on this device

This method discovers the four hardware options associated with the particular device and returns them as an array of integers. The method getHardwareOptionName(int) can be used to get a human readable string describing each of the discovered options.

The hardware options allow the user to determine which network interfaces are associated with which hardware device. Devices will have 1-2 different network interfaces, though the returned array will always be of length 4 (maximum number of interfaces). This array can be used to check against the hardware requirement property of **FirmwareInfo** objects to determine if the particular device will support a firmware choice.

This method requires administrator permission to run. If your application is not running with administrator permission, a *DTCommException* will be thrown.

The discovery process can take a few seconds to complete, so it is recommended that this method only be called once. Once this method has been called, the result is saved and you can read it back using the method *getHardwareOptions().*

3.9 Method: int[] getHardwareOptions()

PARAMETERS	
N/A	
RETURN	
Int[4] HardwareOptions	An array of the 4 hardware options available on this device

This method can be called to read the hardware options array of a MasterDevice object (*see section 3.8)*. If called before the hardware options have been read, the array will contain all 0's.

3.10 Method: bool loadFirmware(string filepathToLoad, string saveOldFirmwareFilepath, bool andRestart)

PARAMETERS	
String filepathToLoad	Filepath of firmware to load
String saveOldFirmwareFilepath	Filepath to save currently loaded firmware
Bool andRestart	Flag indicating weather device should restart afterwards
RETURN	
Bool success	Indicates success or failure of loading new firmware

This method will load a new firmware file to the device and attempt to save the currently loaded firmware to the provided filepath. This method will overwrite the old file regardless of its success moving the old file out of the device, so you should always maintain a directory containing all firmware files you intend to use with your application.

The andRestart flag will cause the device to automatically reset after the new firmware is loaded. The new firmware will not be active until the device is restarted so if this is set to false, you will have to call the restart() method on the device before it will be usable.

This method will return a Boolean indicating the success of the firmware download. It will return true if the download is successful, even if the restart flag is set to false (firmware is not yet active).

3.11 Method: bool loadFirmware(string filepathToLoad, bool andRestart)

PARAMETERS	
String filepathToLoad	Filepath of firmware to load
Bool andRestart	Flag indicating weather device should restart afterwards
RETURN	
Bool success	Indicates success or failure of loading new firmware

Same as section 3.10, but does not save the old firmware file.

3.12 Method: ComChannel getComChannel(int channelNumber)

PARAMETERS	
Int channelNumber	Channel number to return
RETURN	
ComChannel channel	The ComChannel object at the instance specified by channelNumber

Used to get a ComChannel object from the MasterDevice. Each device can contain up to 8 channels, though only channel 0 is supported at this time. Channel 0 is the communication channel and can be used to read and manipulate the input/output byte arrays. (*see section 4.0 for more info).*

3.13 Method: int numComChannels()

PARAMETERS	
N/A	
RETURN	
Int numChannels	The number of available ComChannels associated with this device

Each device can have up to 8 ComChannels, used for different purposes depending on the current network firmware. Every device with attached network adapters will have channel 0, allowing manipulation of the input/output byte arrays. Only channel 0 is supported at this time.

3.14 Method: string getHardwareOptionName(int hardwareOptionNumber)

PARAMETERS	
Int hardwareOptionNumber	The hardware option code to evaluate
RETURN	
String readableHardwareOption	A human-readable description of the hardware option indicated

This method will translate a hardware option code returned in the HardwareOptionsArray into a human-readable description of the hardware option (eg. DeviceNet, Ethernet, Profibus).

3.15 Method: string getFirmwareFileName()

PARAMETERS	
N/A	
RETURN	
String firmwareFilename	The base filename of the currently loaded firmware

This method can be used to get the base filename of the currently loaded firmware. This will return an empty string if there is no currently loaded firmware.

3.16 Method: FirmwareInfo getCurrentFirmwareInfo()

PARAMETERS	
N/A	
RETURN	
FirmwareInfo firmware	A Firmware info object describing the currently loaded firmware

This method can be used to get a FirmwareInfo object describing the currently loaded firmware *(see section 5.0).* This object will be null if there is no currently loaded firmware.

3.17 Method: FileInfo getCurrentFirmwareFile()

PARAMETERS	
N/A	
RETURN	
FileInfo firmwareFile	A FileInfo object pointing to the currently loaded firmware file

This method will return a FileInfo object pointing to the currently loaded firmware file. This will be null if no firmware is loaded. The FileInfo object can be used to easily manipulate (delete, copy etc.) the current firmware.

3.18 Method: List<FileInfo> getConfigurationFiles()

PARAMETERS	
N/A	
RETURN	
List<FileInfo> configurationFiles	A list of FileInfo objects representing all currently loaded firmware files

This method can be used to save the configuration currently loaded in the device. Because the configuration for some of the networks is stored across multiple files, a list of FileInfo objects will be

returned. These can then be used to copy, move or delete the files associated with the current configuration.

3.19 Method: bool loadConfiguration(List<string> filepathsToLoad, bool andRestart)

PARAMETERS	
List<string> filepathsToLoad	List of the full file paths of the configuration files to load
Bool andRestart	Flag indicating weather device should restart afterwards
RETURN	
Bool success	Indicates the success of the operation

This method is used to load configuration files into a particular device. Some networks require more than one file for their configuration. The first parameter is a list of the full file paths of each configuration file to load. The second parameter indicates weather the device should be automatically restarted when the operation is complete. A restart will be required before the device can function with the new configuration.

3.20 Method: bool loadConfiguration(List<string> filepathsToLoad, string filepathToSave, bool andRestart)

PARAMETERS	
List<string> filepathsToLoad	List of the full file paths of the configuration files to load
String filepathToSave	Path of directory to save old configuration to before overwriting
Bool andRestart	Flag indicating weather device should restart afterwards
RETURN	
Bool success	Indicates the success of the operation

Same as the previous method (section 3.19) but will attempt to save any current configuration files to the provided directory path before overwriting them. If files with conflicting names already exist at the specified “filepathToSave” the operation will not continue and return false. If there is some other error preventing the files from being successfully saved, an Exception will be thrown indicating the issue, and the operation will not continue. If the files are successfully saved and the new configuration successfully loaded, the operation will return true.

3.21 Method: bool restart()

PARAMETERS	
N/A	
RETURN	
Bool Success	Indicates the success of the restart operation

This method is used to restart the device. A restart is required after loading a new firmware or configuration. Some networks will not recognize slaves that have previously dropped from the network

until a restart occurs. Restarting the device will interrupt bus communication but will not affect the loaded firmware or configuration.

3.22 Method: List<FirmwareInfo> getSupportedFirmware()

PARAMETERS	
N/A	
RETURN	
List<FirmwareInfo> supportedFirmware	A list of supported firmware as FirmwareInfo objects

This method will return a list of supported firmware in the form of FirmwareInfo objects based on the hardware options of the device. If the readHardwareOptions() method has not yet been called, this method will return a list of 4 FirmwareInfo objects indicating NONE as the display name.

3.23 Method: List<FirmwareInfo> getAllFirmware()

PARAMETERS	
N/A	
RETURN	
List<FirmwareInfo> allPossibleFirmware	A list of all possible firmware as FirmwareInfo objects

This method will return a list of all possible firmware options in the form of FirmwareInfo objects. This method will always return the same list, regardless of a specific objects hardware options.

4.0 ComChannel

Each MasterDevice can contain up to 8 ComChannels. Each ComChannel represents a segment of the Dual-Port memory and is used for a specific purpose. Channel 0 is the communication channel and is used to read and write bytes to and from the network. Channel 0 is the only supported channel at this time. For more information on the other channels, see the document *netX Dual-Port Memory Interface DPM 12 EN.pdf* included on your DT9000 Master Quick Start CD.

A ComChannel is acquired through the getComChannel(int channelNumber) method of a MasterDevice object. All devices with connected network hardware will have a channel 0. If channel 0 is null, this could indicate that the currently loaded configuration is not correct, or that another application is blocking use of the hardware driver.

It is important to note that all of the DT9000 Master Hardware is auto-starting. This means that the network communication will automatically start when an appropriate firmware and configuration is loaded. This allows network communication to begin without intervention after a master device or DT9000 restart. An explicit restart command must be sent after you change the firmware or configuration, but the device will start communication automatically when it comes back online (assuming that there are no firmware or configuration errors). If you wish to stop the network communication, remove the firmware or configuration files using methods in the MasterDevice object.

4.1 Constructor

The constructor should not be called outside the MasterDevice class. This will likely result in a non-functioning ComChannel. Instead, obtain an instance of the ComChannel class using the getComChannel(int channelNumber) method of a MasterDevice. If the resulting channel is null, you may have a problem in your master configuration (*see DT9000 Quick Start Guide*).

4.2 Method: MasterDevice getDevice()

PARAMETERS	
N/A	
RETURN	
MasterDevice associatedDevice	A MasterDevice object indicating the associated hardware

This method can be used to get the MasterDevice object that owns this ComChannel. This is useful for gathering information about the channel, such as the associated network firmware.

4.3 Method: string getChannelName()

PARAMETERS	
N/A	
RETURN	
String channelName	Returns the DisplayName of the associated firmware

This method returns the DisplayName property of the Firmware that is currently loaded into the master device that owns the ComChannel. This allows you to easily determine the network protocol being implemented on this channel.

4.4 Method: int[] getInputData(int numBytes, int offset)

PARAMETERS	
Int numBytes	Number of bytes to read
Int offset	Offset to start at in the Input Data Block
RETURN	
Int[] inputByteData	The requested portion of the Input Data Block

This method is used to read from the Input Data Block of the dual-port memory on the master hardware. The input data block is 5760 bytes long, and any bytes above the final index of the configured input array for your current network will contain *garbage data*.

Though the returned array is of type int[] each element will actually be a byte value (0 – 255).

The offset indicates where to begin reading (0 – 5760) and the numBytes is how many bytes to read beyond the start index.

A null pointer or exception thrown by this method indicates an incorrect configuration loaded into the master.

4.5 Method: int[] getLastOutputData(int numBytes, int offset)

PARAMETERS	
Int numBytes	Number of bytes to read
Int offset	Offset to start at in the Output Data Block
RETURN	
Int[] outputByteData	The requested portion of the Output Data Block

This method is used to read from the Output Data Block of the dual-port memory on the master hardware. The output data block is 5760 bytes long, and any bytes above the final index of the configured output array for your current network will contain *garbage data*.

Though the returned array is of type int[] each element will actually be a byte value (0 – 255).

The offset indicates where to begin reading (0 – 5760) and the numBytes is how many bytes to read beyond the start index.

A null pointer or exception thrown by this method indicates an incorrect configuration loaded into the master.

This method should be called at least once to ensure that you have the correct starting configuration of the output array, or cyclically if there are other processes changing the output array to ensure that the output values in your application are up to date.

4.6 int setOutputData(int[] data, int offset)

PARAMETERS	
Int[] data	Byte data to write
Int offset	Offset to start at in the Output Data Block
RETURN	
Int returnStatus	Indicates the success of the operation

This method is used to set values in the output data block. The output data block is 5760 bytes long, but any bytes written above the last configured index in your master configuration will not be sent over the network.

Though the array is of type int[] each element must actually be a byte value (0 – 255).

The offset indicates where to begin writing (0 – 5760) and the numBytes is how many bytes to write beyond the start index.

A non-zero return status indicates an error has occurred. This likely means that your master configuration is incorrect or another process is blocking use of the hardware driver.

5.0 FirmwareInfo Class

This class was included as a convenient way to gather information about a particular network firmware. You can get a complete list of firmware by calling *getAllFirmware()* on a MasterDevice object.

It is important to note that a FirmwareInfo class does not actually contain the firmware file, it only provides the base filename (eg. Cifxeim.nxf) for the file it describes. You will have to maintain a directory somewhere containing all of the firmware files required for your application.

5.1 Property: String DisplayName

This is a read-only property that indicates the human-readable name of a firmware object. You can use this property to display firmware choices to a user.

5.2 Property: String FileName

This is a read-only property that indicates the base filename of a firmware file. This can be used to pick a particular firmware file out from a directory containing multiple files.

5.3 Property: Int HardwareRequirement

This integer corresponds to the hardwareOptions array of a MasterDevice object. This can be used to match compatible firmware to a MasterDevice object.