Cat. No. W381-E1-03

3G8F7-DRM21-E

DeviceNet™ PCI Board

OPERATION MANUAL

OMRON

Trademarks and Copyrights

3G8F7-DRM21-E DeviceNet™ PCI Board

Operation Manual

Revised September 2013

Notice:

OMRON products are manufactured for use according to proper procedures by a qualified operator and only for the purposes described in this manual.

The following conventions are used to indicate and classify precautions in this manual. Always heed the information provided with them. Failure to heed precautions can result in injury to people or damage to property.

⚠DANGER Indicates an imminently hazardous situation which, if not avoided, will result in death or serious injury. Additionally, there may be severe property damage.

⚠WARNING Indicates a potentially hazardous situation which, if not avoided, could result in death or serious injury. Additionally, there may be severe property damage.

⚠Caution Indicates a potentially hazardous situation which, if not avoided, may result in minor or moderate injury, or property damage.

OMRON Product References

All OMRON products are capitalized in this manual. The word “Unit” is also capitalized when it refers to an OMRON product, regardless of whether or not it appears in the proper name of the product.

The abbreviation “Ch,” which appears in some displays and on some OMRON products, often means “word” and is abbreviated “Wd” in documentation in this sense.

Visual Aids

The following headings appear in the left column of the manual to help you locate different types of information.

Note Indicates information of particular interest for efficient and convenient operation of the product.

1,2,3... 1. Indicates lists of one sort or another, such as procedures, checklists, etc.

No patent liability is assumed with respect to the use of the information contained herein. Moreover, because OMRON is constantly striving to improve its high-quality products, the information contained in this manual is subject to change without notice. Every precaution has been taken in the preparation of this manual. Nevertheless, OMRON assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained in this publication.

TABLE OF CONTENTS

TABLE OF CONTENTS

About this Manual:

This manual describes the installation and operation of the 3G8F7-DRM21-E DeviceNet PCI Board and includes the sections described below.

Please read this manual carefully and be sure you understand the information provided before attempting to install and operate the 3G8F7-DRM21-E DeviceNet PCI Board.

Section 1 provides an overview of the DeviceNet PCI Board's functions, specifications, and system configurations.

Section 2 explains how to set the DeviceNet PCI Board's board ID, install the Board in the computer, and connect the communications cable.

Section 3 explains how to install the DeviceNet PCI Board's drivers and Scanner SDK software.

Section 4 provides flowcharts showing how to use the API functions as well as precautions to observe when using the API functions. Refer to this section when actually writing the applications required to use the DeviceNet PCI Board.

Section 5 provides details on the various API functions in the BusDScan.DLL that are used with the DeviceNet PCI Board.

Section 6 describes the sample programs that have been provided as reference when writing programs for the DeviceNet PCI Board.

Section 7 describes communications timing in remote I/O communications and message communications.

Section 8 describes troubleshooting and error processing procedures needed to identify and correct errors that can occur during DeviceNet PCI Board operation.

⚠ **WARNING** Failure to read and understand the information provided in this manual may result in personal injury or death, damage to the product, or product failure. Please read each section in its entirety and be sure you understand the information provided in the section and related sections before attempting any of the procedures or operations given.

Terms and Conditions Agreement

Read and understand this Manual

Please read and understand this catalog before purchasing the products. Please consult your OMRON representative if you have any questions or comments.

Warranty, Limitations of Liability

Warranties

- **Exclusive Warranty**

  Omron's exclusive warranty is that the Products will be free from defects in materials and workmanship for a period of twelve months from the date of sale by Omron (or such other period expressed in writing by Omron). Omron disclaims all other warranties, express or implied.

- **Limitations**

  OMRON MAKES NO WARRANTY OR REPRESENTATION, EXPRESS OR IMPLIED, ABOUT NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OF THE PRODUCTS. BUYER ACKNOWLEDGES THAT IT ALONE HAS DETERMINED THAT THE PRODUCTS WILL SUITABLY MEET THE REQUIREMENTS OF THEIR INTENDED USE.

  Omron further disclaims all warranties and responsibility of any type for claims or expenses based on infringement by the Products or otherwise of any intellectual property right.

- **Buyer Remedy**

  Omron's sole obligation hereunder shall be, at Omron's election, to (i) replace (in the form originally shipped with Buyer responsible for labor charges for removal or replacement thereof) the non-complying Product, (ii) repair the non-complying Product, or (iii) repay or credit Buyer an amount equal to the purchase price of the non-complying Product; provided that in no event shall Omron be responsible for warranty, repair, indemnity or any other claims or expenses regarding the Products unless Omron's analysis confirms that the Products were properly handled, stored, installed and maintained and not subject to contamination, abuse, misuse or inappropriate modification. Return of any Products by Buyer must be approved in writing by Omron before shipment. Omron Companies shall not be liable for the suitability or unsuitability or the results from the use of Products in combination with any electrical or electronic components, circuits, system assemblies or any other materials or substances or environments. Any advice, recommendations or information given orally or in writing, are not to be construed as an amendment or addition to the above warranty.

See http://www.omron.com/global/ or contact your Omron representative for published information.

Limitation on Liability; Etc

OMRON COMPANIES SHALL NOT BE LIABLE FOR SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, LOSS OF PROFITS OR PRODUCTION OR COMMERCIAL LOSS IN ANY WAY CONNECTED WITH THE PRODUCTS, WHETHER SUCH CLAIM IS BASED IN CONTRACT, WARRANTY, NEGLIGENCE OR STRICT LIABILITY.

Further, in no event shall liability of Omron Companies exceed the individual price of the Product on which liability is asserted.

Application Considerations

Suitability of Use

Omron Companies shall not be responsible for conformity with any standards, codes or regulations which apply to the combination of the Product in the Buyer's application or use of the Product. At Buyer's request, Omron will provide applicable third party certification documents identifying ratings and limitations of use which apply to the Product. This information by itself is not sufficient for a complete determination of the suitability of the Product in combination with the end product, machine, system, or other application or use. Buyer shall be solely responsible for determining appropriateness of the particular Product with respect to Buyer's application, product or system. Buyer shall take application responsibility in all cases.

NEVER USE THE PRODUCT FOR AN APPLICATION INVOLVING SERIOUS RISK TO LIFE OR PROPERTY WITHOUT ENSURING THAT THE SYSTEM AS A WHOLE HAS BEEN DESIGNED TO ADDRESS THE RISKS, AND THAT THE OMRON PRODUCT(S) IS PROPERLY RATED AND INSTALLED FOR THE INTENDED USE WITHIN THE OVERALL EQUIPMENT OR SYSTEM.

Programmable Products

Omron Companies shall not be responsible for the user's programming of a programmable Product, or any consequence thereof.

Disclaimers

Performance Data

Data presented in Omron Company websites, catalogs and other materials is provided as a guide for the user in determining suitability and does not constitute a warranty. It may represent the result of Omron's test conditions, and the user must correlate it to actual application requirements. Actual performance is subject to the Omron's Warranty and Limitations of Liability.

Change in Specifications

Product specifications and accessories may be changed at any time based on improvements and other reasons. It is our practice to change part numbers when published ratings or features are changed, or when significant construction changes are made. However, some specifications of the Product may be changed without any notice. When in doubt, special part numbers may be assigned to fix or establish key specifications for your application. Please consult with your Omron's representative at any time to confirm actual specifications of purchased Product.

Errors and Omissions

Information presented by Omron Companies has been checked and is believed to be accurate; however, no responsibility is assumed for clerical, typographical or proofreading errors or omissions.

PRECAUTIONS

This section provides general precautions for using the DeviceNet PCI Board and related devices.

The information contained in this section is important for the safe and reliable application of the DeviceNet PCI Board. You must read this section and understand the information contained before attempting to set up or operate a DeviceNet PCI Board as part of a control system.

1 Intended Audience

This manual is intended for the following personnel, who must also have knowledge of electrical systems (an electrical engineer or the equivalent).

- Personnel in charge of installing FA systems.
- Personnel in charge of designing FA systems.
- Personnel in charge of managing FA systems and facilities.

2 General Precautions

The user must operate the product according to the performance specifications described in the operation manuals.

Before using the product under conditions which are not described in the manual or applying the product to nuclear control systems, railroad systems, aviation systems, vehicles, combustion systems, medical equipment, amusement machines, safety equipment, and other systems, machines, and equipment that may have a serious influence on lives and property if used improperly, consult your OMRON representative.

Make sure that the ratings and performance characteristics of the product are sufficient for the systems, machines, and equipment, and be sure to provide the systems, machines, and equipment with double safety mechanisms.

This manual provides information for installing and operating the DeviceNet PCI Board. Be sure to read this manual before operation and keep this manual close at hand for reference during operation.

⚠ **WARNING** It is extremely important that all control products be used for the specified purpose and under the specified conditions, especially in applications that can directly or indirectly affect human life. You must consult with your OMRON representative before applying an OMRON control system to the abovementioned applications.

3 Safety Precautions

⚠ **WARNING** Never attempt to disassemble the Board or touch the Board while power is being supplied. Doing so may result in serious electrical shock or electrocution.

⚠ **WARNING** Provide safety measures in external circuits, i.e., not in the Programmable Controller (CPU Unit including associated Units; referred to as “PLC”), in order to ensure safety in the system if an abnormality occurs due to malfunction of the PLC or another external factor affecting the PLC operation. Not doing so may result in serious accidents.

1,2,3...

1. Emergency stop circuits, interlock circuits, limit circuits, and similar safety measures must be provided in external control circuits.
2. The PLC will turn OFF all outputs when its self-diagnosis function detects any error or when a severe failure alarm (FALS) instruction is executed. Unexpected operation, however, may still occur for errors in the I/O control section, errors in I/O memory, and other errors that cannot be detected by the self-diagnosis function. As a countermeasure for all such errors, external safety measures must be provided to ensure safety in the system.

3. The PLC outputs may remain ON or OFF due to deposition or burning of the output relays or destruction of the output transistors. As a countermeasure for such problems, external safety measures must be provided to ensure safety in the system.
4. When the 24-VDC output (service power supply to the PLC) is overloaded or short-circuited, the voltage may drop and result in the outputs being turned OFF. As a countermeasure for such problems, external safety measures must be provided to ensure safety in the system.

⚠ **WARNING** The CPU Unit refreshes I/O even when the program is stopped (i.e., even in PROGRAM mode). Confirm safety thoroughly in advance before changing the status of any part of memory allocated to I/O Units, Special I/O Units, or CPU Bus Units. Any changes to the data allocated to any Unit may result in unexpected operation of the loads connected to the Unit. Any of the following operation may result in changes to memory status.

- Transferring I/O memory data to the CPU Unit from a Programming Device.
- Changing present values in memory from a Programming Device.
- Force-setting/-resetting bits from a Programming Device.
- Transferring I/O memory files from a Memory Card or EM file memory to the CPU Unit.
- Transferring I/O memory from a host computer or from another PLC on a network.

⚠ **Caution** Confirm safety at the destination node before transferring a program to another node or changing contents of the I/O memory area. Doing either of these without confirming safety may result in injury.

4 Operating Environment Precautions

Do not install the PCI Board in any of the following locations.

- Locations subject to direct sunlight.
- Locations subject to temperatures or humidities outside the range specified in the specifications.
- Locations subject to condensation as the result of severe changes in temperature.
- Locations subject to corrosive or flammable gases.
- Locations subject to dust (especially iron dust) or salt.
- Locations subject to exposure to water, oil, or chemicals.
- Locations subject to shock or vibration.

Provide proper shielding when installing in the following locations:

- Locations subject to static electricity or other sources of noise.
- Locations subject to strong electromagnetic fields.
- Locations subject to possible exposure to radiation.
- Locations near to power supply lines.

5 Application Precautions

Observe the following precautions when using the DeviceNet PCI Board.

- Install failsafe safety mechanisms to provide safety in the event of incorrect signals that may result from signal line disconnections or power interruptions.
- Always use the power supply voltage specified in this manual.
- Mount the Board only after checking the connectors and terminal blocks completely.
- Take appropriate measures to ensure that the specified power with the rated voltage and frequency is supplied in places where the power supply is unstable. An incorrect power supply may result in malfunction.
- Always connect to a ground of 100 Ω or less when installing. Not connecting to a ground of 100 Ω or less may result in electric shock.
- Install external breakers and take other safety measures against short-circuiting in external wiring. Insufficient safety measures against short-circuiting may result in burning.
- Always turn OFF the power supply to the computer or slave before attempting any of the following. Not turning OFF the power supply may result in malfunction or electric shock.
    - Mounting or dismounting DeviceNet PCI Board.
    - Setting rotary switches.
    - Assembling the Boards.
    - Connecting cables or wiring the system.
    - Connecting or disconnecting the connectors.
- Do not attempt to disassemble, repair, or modify any product.
- Be sure that all the board mounting screws, cable screws, and cable connector screws are tightened to the torque specified in the relevant manuals. Incorrect tightening torque may result in malfunction.
- Use crimp terminals for wiring. Do not connect bare stranded wires directly to terminals.
- Double-check all the wiring and switch settings before turning ON the power supply.
- Wire all connections correctly.
- Observe the following precautions when wiring the cable.
    - Separate the communications cables from the power lines or high-tension lines.
    - Do not bend the communications cables.
    - Do not pull on the communications cables.
    - Do not place heavy objects on top of the communications cables.
    - Be sure to wire communications cable inside ducts.
    - Place communications cables in ducts.
    - Use the specified communications cables.
    - Always wire communications and signal lines within the specified connection distances.
- Before touching the Board, be sure to first touch a grounded metallic object in order to discharge any static built-up. Not doing so may result in malfunction or damage.

- Test the operation of the ladder program and other user programs completely before starting actual system operation.
- Always transfer the contents of any required DM Area words, HR Area words, parameters, or other data to CPU Units, CPU Bus Units, and Special I/O Units before restarting operating after replacing any of these Units.
- Be sure that the communications cable connectors, and other items with locking devices are properly locked into place. Improper locking may result in malfunction.
- Do not touch circuit boards or the components mounted to them with your bare hands. There are sharp leads and other parts on the boards that may cause injury if handled improperly.
- When transporting or storing the product, cover the PCBs with electrically conductive materials to prevent LSIs and ICs from being damaged by static electricity, and also keep the product within the specified storage temperature range.
- When transporting or storing circuit boards, cover them in antistatic material to protect them from static electricity and maintain the proper storage temperature.
- Always enable the scan list before operating the control system.
- Check the baud rate of any new node added to an existing network to be sure that it agrees with the rest of the network.
- Do not use the computer’s standby or sleep function while you are using the DeviceNet PCI Board Scanner. If the computer’s standby or sleep function is activated during Scanner usage, communications may be broken or other unexpected errors may occur.
- The DeviceNet PCI Board Scanner does not support computer standby or sleep functions. Do not use the computer’s standby or sleep function while you are using the DeviceNet PCI Board Scanner.

6 Conformance to EC Directives

6-1 Applicable Directives

- EMC Directives

6-2 Concepts

EMC Directives

OMRON devices that comply with EC Directives also conform to the related EMC standards so that they can be more easily built into other devices or machines. The actual products have been checked for conformity to EMC standards. (See the following note.) Whether the products conform to the standards in the system used by the customer, however, must be checked by the customer.

EMC-related performance of the OMRON devices that comply with EC Directives will vary depending on the configuration, wiring, and other conditions of the equipment or control panel in which the OMRON devices are installed. The customer must, therefore, perform final checks to confirm that devices and the overall machine conform to EMC standards.

Note Applicable EMC (Electromagnetic Compatibility) standards are as follows:

EMS (Electromagnetic Susceptibility): EN61131-2
EMI (Electromagnetic Interference): EN61000-6-4
(Radiated emission: 10-m regulations)

6-3 Conformance to EC Directives

DeviceNet products that meet EC directives must be installed as follows:

1,2,3...

1. Used reinforced insulation or double insulation for the DC power supplies used for the communications power supply, internal circuit power supply, and the I/O power supplies.
2. DeviceNet products that meet EC directives also meet the common emission standard (EN61000-6-4). When DeviceNet products are built into equipment, however, the measure necessary to ensure that the standard is met will vary with the overall configuration of the control panel, the other devices connected to the control panel, and other conditions. You must therefore confirm that EC directives are met for the overall machine or device, particularly for the radiated emission requirement (10 m).

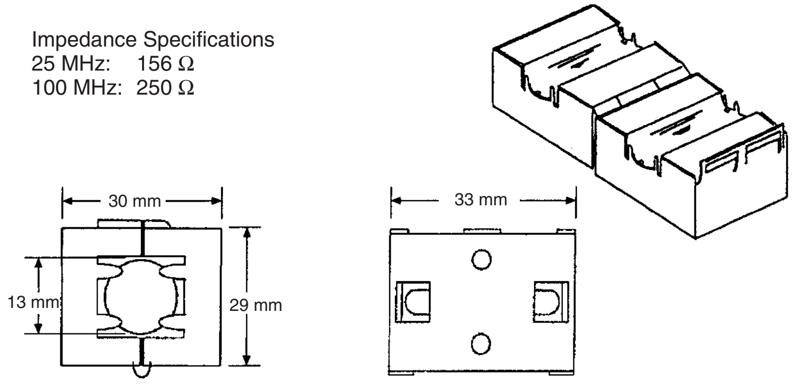
The following examples show means of reducing noise.

1,2,3...

1. Noise from the communications cable can be reduced by installing a ferrite core on the communications cable within 10 cm of the DeviceNet PCI Board.

Ferrite Core (Data Line Filter): 0443-164151 (manufactured by Nisshin Electric Co.)

Impedance Specifications
25 MHz: 156 Ω
100 MHz: 250 Ω



Contact:
Nisshin Electric Co., Sales Department No. 3
Tel: +81 4-2934-4151
Fax: +81 4-2934-4155

2. Keep DeviceNet communications cables as short as possible and ground to 100 Ω min.

7 Components

Be sure that you have received the following components.

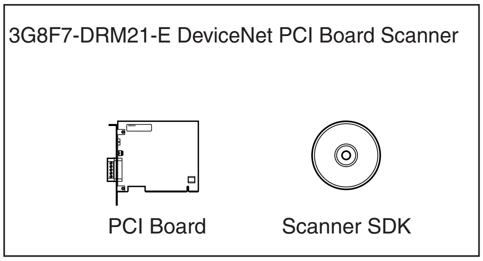
- One PCI Board (with communications connector)
- One installation disk (CD-ROM) for Scanner SDK
- One operation manual (this manual)
- One User Registration Card (which also serves as the software usage license agreement)

SECTION 1
Introduction

This section provides an overview of the DeviceNet Scanner SDK functions, specifications, and system configurations.

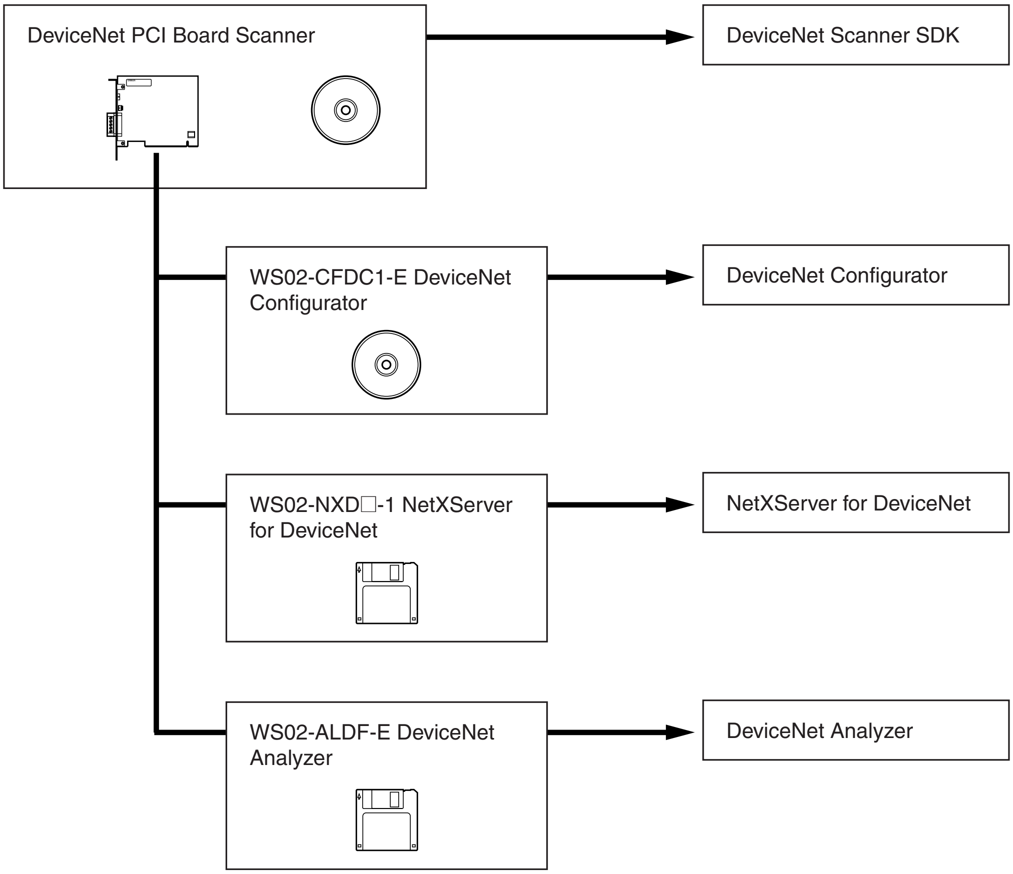
1-1 Product Configuration

The 3G8F7-DRM21-E DeviceNet PCI Board includes the PCI Board (hardware) and the Scanner SDK software on CD-ROM.



1-2 DeviceNet PCI Board

The PCI Board is used as an interface to other software, such as the DeviceNet Configurator, NetXServer, and Analyzer.


DeviceNet PCI Board Scanner
DeviceNet Scanner SDK
WS02-CFDC1-E DeviceNet Configurator
DeviceNet Configurator
WS02-NXD□-1 NetXServer for DeviceNet
NetXServer for DeviceNet
WS02-ALDF-E DeviceNet Analyzer
DeviceNet Analyzer

DeviceNet Scanner SDK

The DeviceNet Scanner SDK (this product) is a library for developing applications that operate as DeviceNet Masters or Slaves. It is supplied as a DLL file for a Windows environment.

Use the Scanner SDK to develop Master/Slave applications with industry-leading performance and functions.

DeviceNet Configurator

The DeviceNet Configurator is a Windows-based application that supports construction of DeviceNet networks. The Configurator is used not only for setting parameters and monitoring OMRON Master and Slave devices, but also for setting parameters for slaves from other manufacturers, simply by installing the EDS files.

The Configurator provides extensive support for managing networks, from design through to maintenance.

NetXServer for DeviceNet

The NetXServer is middleware that operates in a Windows environment. The NetXServer collects I/O data from a DeviceNet network and provides it to monitoring and other applications. It operates as a DDE server.

NetXServer enables I/O data monitoring without affecting Master or Slave communications.

The following two types of NetXServer are available:

DDE Edition: For monitoring I/O data using a DDE client (e.g., Microsoft Excel)

SDK Edition: Library for developing monitoring applications using NetXServer functions

DeviceNet Analyzer

The DeviceNet Analyzer is a Windows-based application for analyzing message frames on a DeviceNet network.

The DeviceNet Analyzer can display the message frames being transmitted on a network and indicate traffic status. It can be used to find the source of errors and for developing DeviceNet-compatible devices.

1-3 Scanner SDK Functions and Features

DeviceNet Communications Functions

The Scanner SDK is equipped with the following communications functions.

- I/O communications functions that exchange I/O data with other DeviceNet nodes:

  DeviceNet Master function
  DeviceNet Slave function

- DeviceNet explicit messaging functions (client and server functions)

In addition to the communications functions above, the Scanner SDK has a status function that reads the status of the node (Master/Slave) and the network and an error log function that records errors and their time of occurrence.

Note The DeviceNet network is capable of exchanging I/O with distant Slaves through a single cable. Moreover, Slaves and other Masters can be controlled and monitored by sending and receiving explicit messages. Refer to the DeviceNet Operation Manual (W267) for more details.

In this manual, the “client” is the node that sends a message requesting services and the “server” is the node that receives the message, performs the requested processing, and returns a response.

DeviceNet Communications Features

The Scanner SDK has the following features:

Exchange I/O Data with DeviceNet Slaves

The status of I/O points on DeviceNet Slaves is mirrored in the DeviceNet PCI Board. I/O can be performed with a specified Slave by calling the functions for reading and writing I/O data.

Use Other Vendor's DeviceNet-compatible Devices

DeviceNet is a worldwide standard, so any manufacturer's Slave can be connected as long as it is DeviceNet compatible.

I/O Capacity of 37,800 Bytes for Up To 63 Slaves

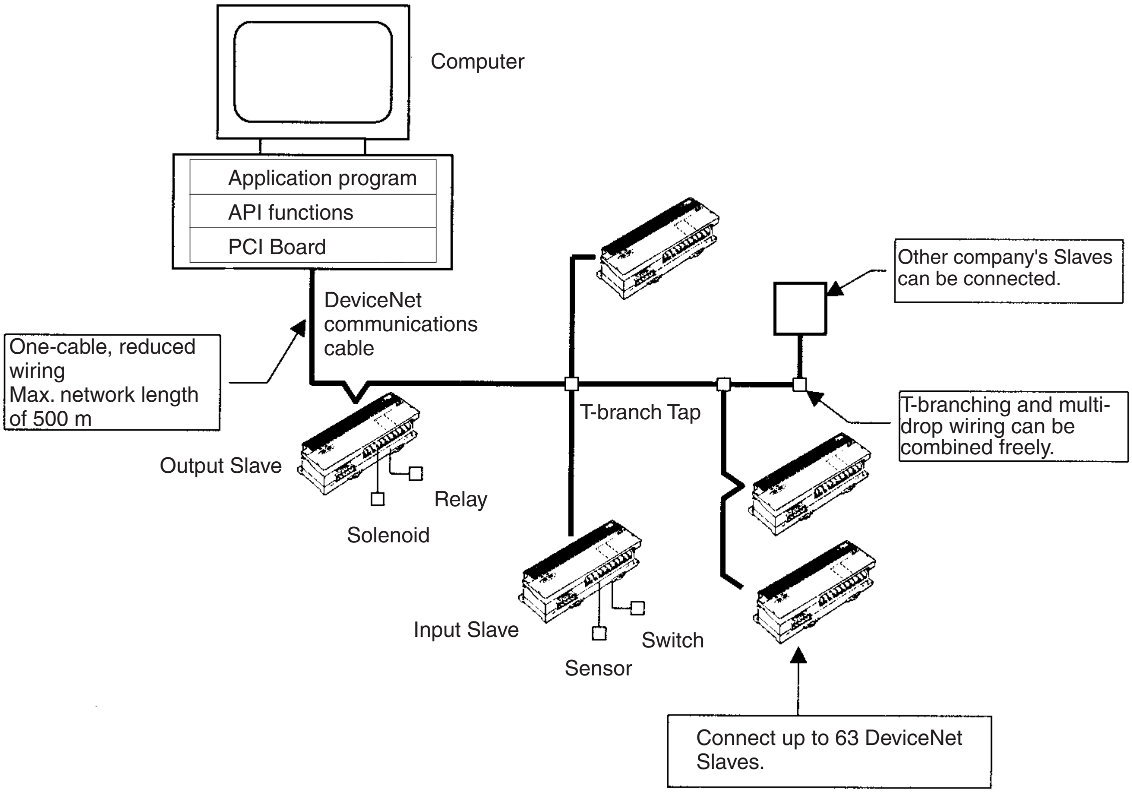
The Scanner SDK provides 37,800 bytes for I/O allocation to up to 63 Slaves (input: 25,200 bytes; output: 12,600 bytes).

Use API Functions to Control Devices

All Scanner SDK functions are provided as API functions. User applications are created using the API functions.

Check Events with Windows Messaging or Polling

Events can be checked in two ways: automatic notification by Windows messaging and monitoring (polling) of the Board's event queue by user applications. Use the method most appropriate for each application.


Computer
Application program
API functions
PCI Board
DeviceNet communications cable
One-cable, reduced wiring
Max. network length of 500 m
Output Slave
Relay
Solenoid
T-branch Tap
Input Slave
Sensor
Switch
Other company's Slaves can be connected.
T-branching and multi-drop wiring can be combined freely.
Connect up to 63 DeviceNet Slaves.

1-4 Scanner SDK Functions

1-4-1 I/O Communications

Master Function

The Scanner SDK Master function provides two 200-byte input areas (100 words or 1,600 points) and one 200-byte output area (100 words or 1,600 points) for allocation to each slave.

I/O communications are executed according to the scan list registered by the Scanner SDK. Scan lists record information such as the number of input and output bytes for each slave.

Maximum Numbers of I/O Points and Slaves

The following table shows the max. number of I/O points, max. number of Slaves, and max. number of I/O connections allowed by the Scanner SDK's Master function.

Item	Specification
Max. number of I/O points	Input: 25,200 bytes (= 12,600 words or 201,600 points) Output: 12,600 bytes (= 6,300 words or 100,800 points)
Max. number of I/O points per Slave	Input: 200 bytes × 2 (= 100 words × 2 or 1,600 points × 2) Output: 200 bytes (= 100 words or 1,600 points)
Max. number of Slaves	63 Slaves (Node addresses 0 to 63 can be used.)
Max. number of I/O connections per Slave	2 max.

Note Two input areas have been provided for each slave, but normally only the first area is used. If two connections are used at the same time, then the second input area can be used.

Slave Function

The Scanner SDK Slave function provides two 200-byte input areas (100 words or 1,600 bits) and one 200-byte of output area (100 words or 1,600 bits). The following methods can be used to register the Master in the slave scan list.

1,2,3...
1. Use functions to register Slaves individually or in a group.
2. Register Slaves in a group by specifying a parameter file that was created with the OMRON DeviceNet Configurator.

A slave scan list must be registered in the Scanner SDK for nodes to operate as Slaves.

Maximum Numbers of I/O Points and Masters

The following table shows the max. number of I/O points and max. number of Masters allowed by the Scanner SDK's Slave function.

Item	Specification
Max. number of I/O points	Input: 200 bytes × 2 (= 100 words × 2 or 1,600 points × 2) Output: 200 bytes (= 100 words or 1,600 points)
Max. number of Masters	1 Master

Note Two input areas have been provided, but normally only the first area is used. If two connections are used at the same time, then the second input area can be used.

1-4-2 Message Communications Function

Explicit Message Communications

The DeviceNet PCI Board supports explicit message communications.

As a client, the DeviceNet PCI Board can send explicit messages to control or monitor other nodes in the DeviceNet network when necessary.

As a server, the DeviceNet PCI Board can receive explicit messages from other nodes. (The requested processing and responses must be handled in user applications.)

Explicit message communications can be used to freely communicate with DeviceNet-compatible devices produced by other companies.

Maximum Number of Connections

The following table shows maximum number of connections allowed.

Item	Specification
Max. number of client connections	63 connections (1 connection per server)
Max. number of server connections	4 connections (1 connection per client)

1-4-3 Maintenance Functions

Read Status Functions

The DeviceNet PCI Board can read the following information, including settings and the operating status of the nodes (Master/Slaves) and network.

- Scanner SDK's DLL version
- DeviceNet PCI Board's driver version
- Whether or not the DeviceNet PCI Board is installed
- Network status
- Operational status in the network/status in remote I/O communications
- Communications status
- Whether or not each Slave is registered in the scan list
- Each Slave's device status

Reset Function

The DeviceNet PCI Board can be reset (initialized) with a command from the computer.

Communications Cycle Time Management

This function can set the communications cycle time (interval between the exchange of the Slave's I/O) and read or clear the minimum and maximum values.

Error Log

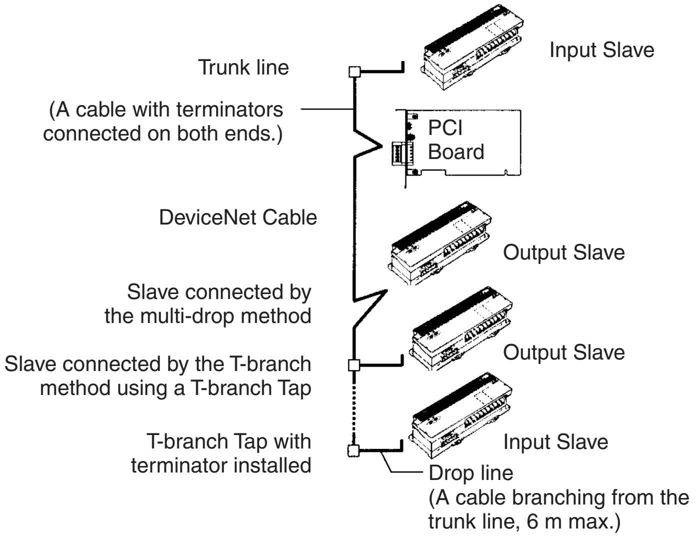
The DeviceNet PCI Board has an error log function that records information on errors that occur during operation. The error log can be checked to pinpoint errors for faster error processing and recovery.

PC Watchdog Timer Management

Remote I/O can be made to stop automatically if the application that controls the DeviceNet PCI Board stops for some reason. The Board's PC watchdog timer is refreshed regularly from the computer (application) to notify the Board that the application is operating normally.

1-5 System Configuration

The following diagram shows the various device connections allowed.


Trunk line
(A cable with terminators connected on both ends.)
DeviceNet Cable
Slave connected by the multi-drop method
Slave connected by the T-branch method using a T-branch Tap
T-branch Tap with terminator installed
Input Slave
PCI Board
Output Slave
Output Slave
Input Slave
Drop line
(A cable branching from the trunk line, 6 m max.)

Note Refer to the following manuals for information on Slaves.

- DRT2 Series DeviceNet Slave Operation Manual (W404)
- C200HW-DRT21, CQM1-DRT21, and DRT1 Series DeviceNet Slave Operation Manual (W347)
- DRT1-COM and GT1 Series DeviceNet MULTIPLE I/O TERMINAL Operation Manual (W348)

Baud Rate and Distance

The following table shows the relationship between the baud rate and communications distance in the DeviceNet network.

Baud rate	Maximum network length		Drop line length	Total drop line length
	Thick cable	Thin cable		
500 kbps	100 m max.	100 m max.	6 m max.	39 m max.
250 kbps	250 m max.			78 m max.
125 kbps	500 m max.			156 m max.

Slave Connection Methods

Slave devices can be connected in two ways. These connection methods can be combined in the same network.

Method	Description
T-branch Method	Slaves are connected to a drop line from the trunk line or branch line created with a T-branch Tap.
Multi-drop Method	Slaves are directly connected to the trunk line or the drop line.

Note Refer to the *DeviceNet Operation Manual* (W267) for details on connection methods and grounding.

1-6 Specifications

1-6-1 DeviceNet PCI Board General Specifications

Item	Specifications
Dimensions	119.9 × 106.7 mm (W × H)
Operating voltage range	5 VDC ± 5% (3.3 VDC is not used.)
Current consumption	Internal power supply: 290 mA max. at 5 VDC Communications power supply: 30 mA max. at 24 VDC
Vibration resistance	10 to 57 Hz, 0.075-mm double amplitude, 57 to 150 Hz, acceleration: 9.8 m/s^2 in X, Y, and Z directions for 80 minutes each (Time coefficient; 8 minutes × coefficient factor 10 = total time 80 minutes) DIN Track installation: 2 to 55 Hz, 2.94 m/s^2 in X, Y, and Z directions for 20 minutes each
Shock resistance	147 m/s^2 three times each in X, Y, and Z directions
Ambient temperature	Operating: 0 to 55°C Storage: –20 to 60°C
Humidity	10% to 90% (with no condensation)
Atmosphere	Must be free from corrosive gas
Weight	91 g max.
Max. number of Boards	3 Boards/computer max.

The DeviceNet PCI Board conforms to PCI Local Bus Specification Rev. 2.

1-6-2 DeviceNet Communications Specifications

Item		Specification
Communications protocol		DeviceNet
Connection forms		Multi-drop and T-branch connections can be used for trunk or drop lines. Terminators must be connected at both ends of the trunk line.
Baud rate		500 kbps, 250 kbps, or 125 kbps (Specified with the SCAN_Online function.)
Communications media		Special 5-wire cables (2 signal lines, 2 power lines, 1 shield line)
Communica-tions distances	500 kbps	Network length: 100 m max. Drop line length: 6 m max. Total drop line length: 39 m max.
	250 kbps	Network length: 250 m max. (see note 1) Drop line length: 6 m max. Total drop line length: 78 m max.
	125 kbps	Network length: 500 m max. (see note 1) Drop line length: 6 m max. Total drop line length: 156 m max.
Communications power supply		11 to 24 VDC, 30 mA (supplied through the communications connector)
Max. number of Slaves		63 Slaves
Communications cycle time (see note 2)		Set between 1 and 500 ms with the SCAN_SetScanTimeValue() function.
Error control checks		CRC error check, node address duplication check, scan list verification
Cable		5 conductors (two signal wires, two power supply wires, and one shield wire)

Note Indicates the max. length when thick cables are used. Reduce the network length to 100 m max. when using thin cables. When using both thick and thin cables together, refer to the *DeviceNet Operation Manual (W267)* for details on the maximum network length.

1-6-3 Scanner SDK Communications Specifications

Item	Specifications
Supported I/O connections	• Bit Strobe • Polling • Cyclic • Change of State (COS) • Explicit Peer-to-peer Messaging
Communications cycle time (See note.)	2 to 500 ms (Can be specified using API functions.)
Number of server nodes capable of simultaneous communications as explicit clients	63 nodes
Number of client nodes capable of simultaneous communications as explicit servers	4 nodes
Data length for explicit messages	Client: Explicit message request: 552 bytes Explicit message response: 552 bytes
	Server: Explicit message request: 552 bytes Explicit message response: 552 bytes
Response monitoring time for explicit messages (for clients)	2 s (default) (Can be specified using API functions.)
Retries for explicit messages	0 (Retries must be performed by the user application.)

Note The communications cycle time is the maximum time from when remote I/O communications are executed by the Master to a Slave until remote I/O communications are executed again for the same Slave.

Minimum System Requirements

Hardware Requirements

IBM PC/AT or Compatible

- At least one PCI bus slot (PCI bus Rev. 2.0 or later)
- 5 MB min. free hard disk space
  (plus additional space for the user program)
- One CD-ROM drive is required to install the software.
- VGA or better display functions.

The processor, memory capacity, and other specifications not listed above should conform to the recommendations for the operating system used.

OS

Microsoft Windows 95, 98, Me, NT 4.0, 2000, XP, 7, 3.1, and NT 3.5 are not supported.

1-6-4 Development Environment

Recommended Development Environment

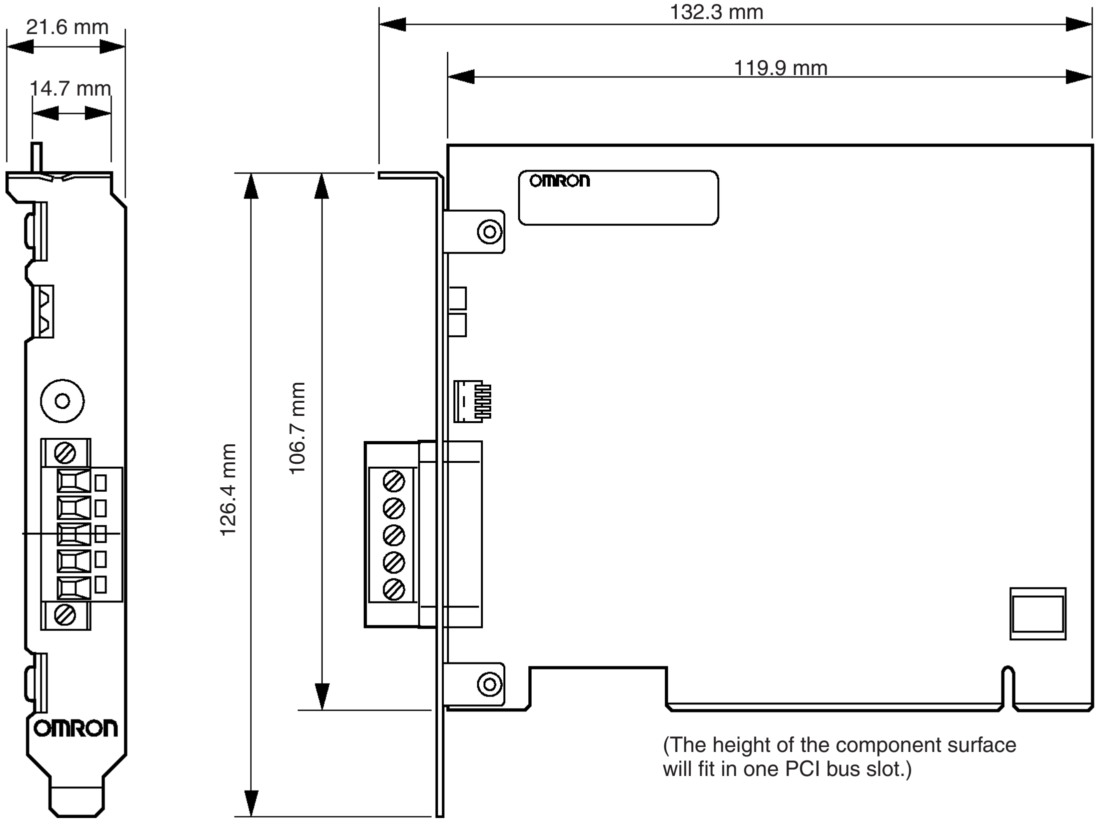
Microsoft Visual C++ (Ver. 6.0 or later.)

Other Development Environments

- Microsoft Visual Basic
  Some functions are limited. Refer to *Precautions when Using Other Development Environments* under *3-1 Application Development Environments* for details.
- Borland C++ Builder
  Refer to Refer to *Precautions when Using Other Development Environments* under *3-1 Application Development Environments* for details.

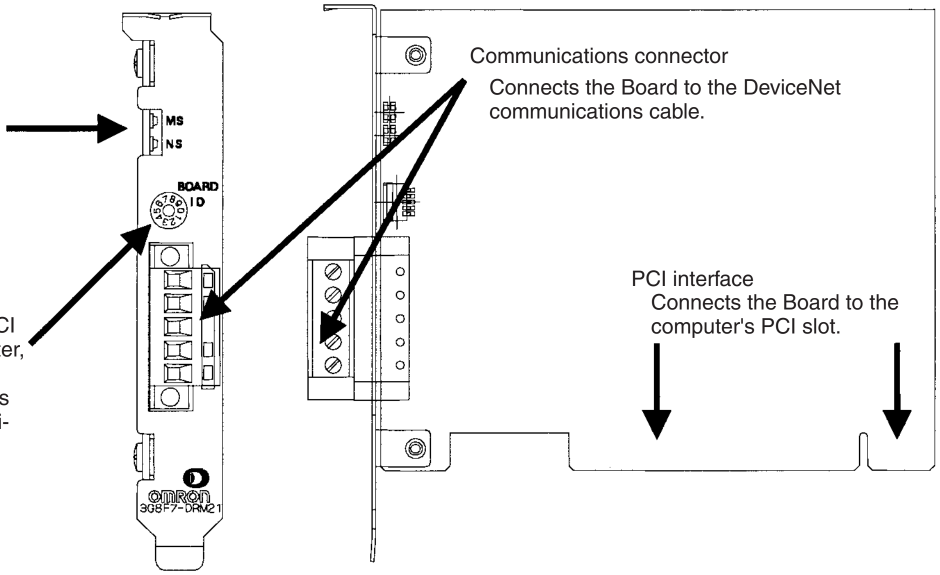
1-6-5 Dimensions

The following diagram shows the dimensions of the DeviceNet PCI Board. (The height of components on the Board is within specifications for a single PCI slot.)



(The height of the component surface will fit in one PCI bus slot.)

1-7 Board Components



LED indicators (MS and NS)
These are the DeviceNet MS (module status) and NS (network status) indicators.

Board ID switch
When two or more DeviceNet PCI Boards are installed in a computer, the computer uses the board ID settings to distinguish the Boards from each other. Set unique decimal board IDs between 0 and 7.

Communications connector
Connects the Board to the DeviceNet communications cable.

PCI interface
Connects the Board to the computer's PCI slot.

LED Indicators

The following table explains the operation of the LED indicators.

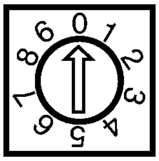
Indicator status		Meaning
MS	NS	
OFF	OFF	Boot program initialization is in progress.
Flashing green	OFF	Scanner firmware initialization is in progress.
Lit green	OFF	Waiting for online request.
Lit green	Flashing green	A connection was established and I/O communications are in progress. Waiting for a connection from the Master.
Lit green	Lit green	I/O communications or message communications are in progress.

Board ID

When two or more DeviceNet PCI Boards are installed in a computer, the computer uses the board ID settings to distinguish the Boards from each other. Specify the board ID in API functions to identify the desired board.

Set the board ID in decimal as shown in the following diagram. The allowed setting range is 0 to 7. (The factory setting is 0.)

Up to 3 DeviceNet PCI Boards can be installed in one computer.



Note Any board ID from 0 to 7 can be set, as long as the ID is not set on another DeviceNet PCI Board in the computer. (It is physically possible to set board IDs 8 and 9, but the Board cannot be used properly with these settings.)

1-8 Preparation for Operation

Hardware Settings

If more than one DeviceNet PCI Board is being installed in one computer, set the board IDs on the Boards' rotary switches so that the different Boards can be distinguished from one another. Refer to *2-2 Installing the Board in the Computer* for details.

Always set the rotary switches before turning ON the computer.

Installation on Computer

Install the Board in the computer. Refer to *2-2 Installing the Board in the Computer* for details.

Software Installation

Install the DeviceNet PCI Board driver and software required to use the Board from the computer. Refer to *2-3 Installing the Drivers* and *2-4 Installing the Scanner SDK* for details.

Writing the Program

Write the programs (user applications) that make software settings and control the Board. Refer to *SECTION 3 Using API Functions* through *SECTION 7 Error Processing* for details.

Reference Information

Refer to *2-5 DeviceNet Connections* for information on communications cable connections.

Refer to the *DeviceNet Operation Manual (W267)* for information on wiring DeviceNet networks.

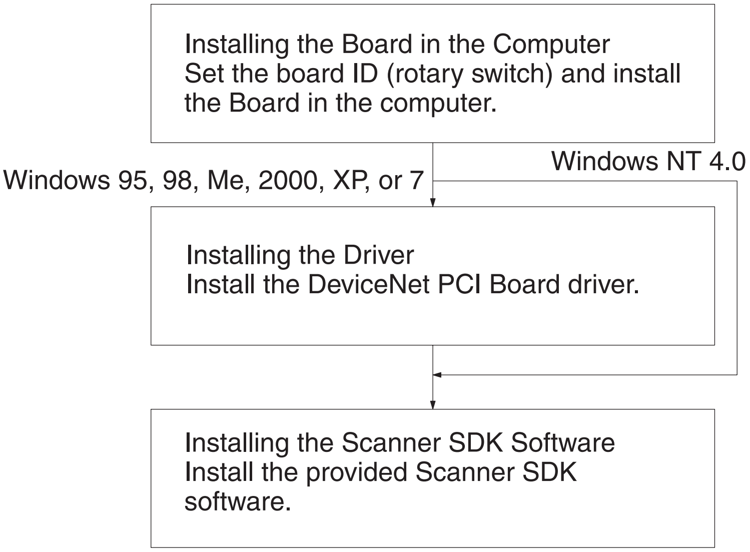
Refer to the *DeviceNet Slave Operation Manuals* (W404 and W347) and the *DeviceNet MULTIPLE I/O TERMINAL Operation Manual* (W348) for information on Slaves.

SECTION 2
Software Installation

This section explains how to install the DeviceNet PCI Board in a computer, how to install the software, and how to connect the communications cables.

2-1 Installation Procedure

There procedure for installing the DeviceNet PCI Board and its software is outlined below.



Installing the Board in the Computer
Set the board ID (rotary switch) and install the Board in the computer.

Windows 95, 98, Me, 2000, XP, or 7

Windows NT 4.0

Installing the Driver
Install the DeviceNet PCI Board driver.

Installing the Scanner SDK Software
Install the provided Scanner SDK software.

Note For Windows 2000, XP, or 7, you must log in with administrator rights to install the driver and Scanner SDK Software.

2-2 Installing the Board in the Computer

This section explains how to make the required settings on the DeviceNet PCI Board and install the Board in the computer.

Preparation for Installation

The DeviceNet PCI Board is plug-and-play compatible with MicroSoft Windows. Check the following points before installing the Board.

Check point	Description
Available PCI slot	Make sure that the computer has an available PCI slot.
IRQ conflict	The DeviceNet PCI Board uses an IRQ. IRQ settings are allocated automatically to the PCI bus, but not to the ISA bus. In computers with ISA slots, the computer may not boot completely if there is an IRQ conflict with the ISA bus. In this case, make one of the following changes in the BIOS settings: • Enter the BIOS setup and specify a non plug-and-play (PnP) operating system. • Enter the BIOS setup and reserve the IRQ numbers (disable automatic allocation of these IRQ numbers) that are used by the ISA bus so that these IRQ numbers will not be allocated automatically to the PCI bus. Refer to the computer's User's Guide for details on entering the BIOS setup and changing the settings. (See note below for details on identifying IRQ numbers.)

Note Use the following procedure to identify which IRQ numbers are being used by the ISA bus.

1,2,3...

1. Start the computer without the DeviceNet PCI Board installed.
2. Start the Device Manager (***Control Panel/System/Device Manager***).
3. Open the properties of the relevant devices and check their IRQ (Interrupt Request) settings in the Resources tab.

Board ID Setting

Set the board ID on the DeviceNet PCI Board's rotary switch before installing the board.

Note Static electricity can damage the DeviceNet PCI Board's electronic components. Do not touch the Board's connector or components.

Set a board ID between 0 and 7. If two or more DeviceNet PCI Boards are being installed in the computer, be sure that each Board has a unique board ID setting. The factory setting is 0.

Installing the Board in the Computer

Install the DeviceNet PCI Board in one of the computer's PCI slots.

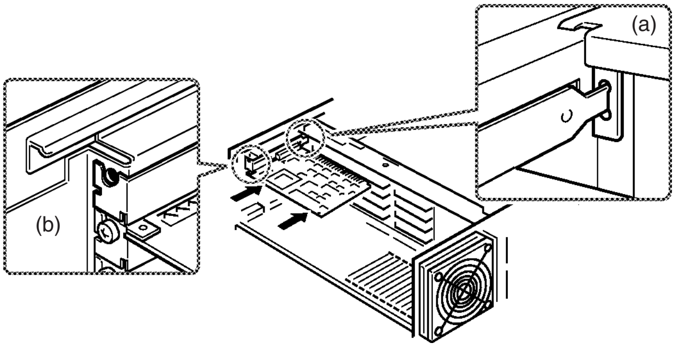
Refer to your computer's User's Guide for details on installing a PCI card in the computer.

Note (1) Turn OFF the computer power supply before installing the DeviceNet PCI Board.

(2) Take precautions against static electricity.

1,2,3...

1. Disconnect all cables from the DeviceNet PCI Board.
2. Turn OFF the computer in which the board is being installed and unplug the computer's power cord.
3. Remove the computer's case and make any other preparations needed to install a PCI board. (Refer to your computer's User's Guide for details.)
4. Align the Board with the computer's PCI slot, slide it into position, and press it firmly into the slot. Check that the Board's PCI interface is completely and evenly installed into the PCI slot. Do not use much force when pressing the board; it should slide into the slot with little resistance.



5. Pull on the Board lightly to check that it is installed securely and won't slip out.
6. Secure the Board by tightening the retaining screw, indicated by (b) in the diagram, to 0.5 N·m.
7. Attach the case to the computer and turn ON the power.
8. The installation method differs depending on the type of Windows operating system used. Refer to the following during installation.
   - Windows 95, 98, Me, 2000, XP, or 7
     After the computer is started, the Board will be detected as new hardware and the InstallShield Wizard will start. Refer to *2-3 Installing the Drivers* for the rest of the installation procedure.
   - Windows NT 4.0
     The PCI Board will not be recognised when the computer is started. Install the driver for the PCI Board at the same time as Scanner SDK. Refer to *2-4 Installing the Scanner SDK* for the rest of the installation procedure.

2-3 Installing the Drivers

After the Board is installed in the computer and the computer is started, the Board will be detected as new hardware (except for Windows NT computers, see note). Install the driver to ensure correct operation of the Board.

The installation method and windows displayed when installing the driver will differ depending on the Windows operating system used.

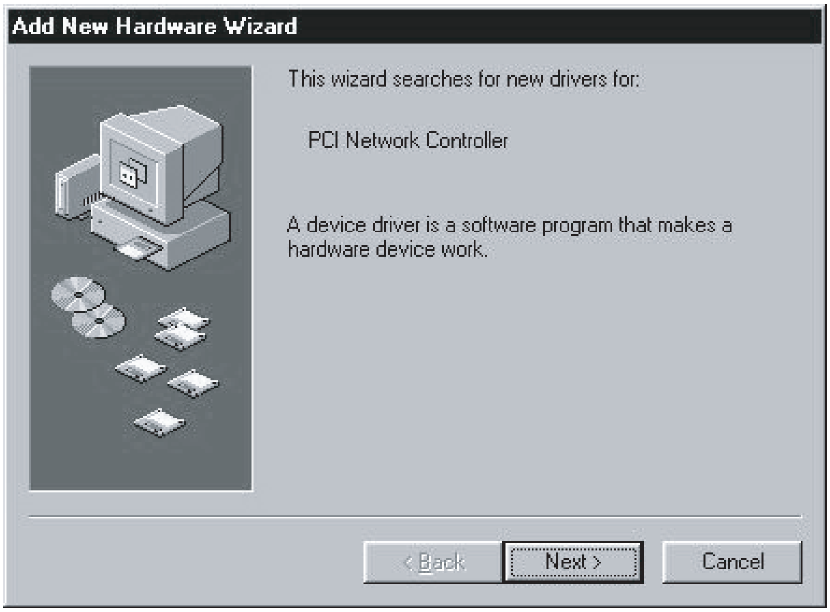
This section describes driver installation for Windows 98, XP, and 7.

Note The Board will not be automatically detected with Windows NT 4.0. The driver must be installed with Scanner SDK. Perform the installation procedure described in *2-4 Installing the Scanner SDK*.

Windows 98 Installation

1,2,3...

1. After the Board is installed in the computer and the computer is started, the Board will be detected as new hardware and the Add New Hardware Wizard will start automatically as shown in the following diagram.
   Click the **Next** Button.



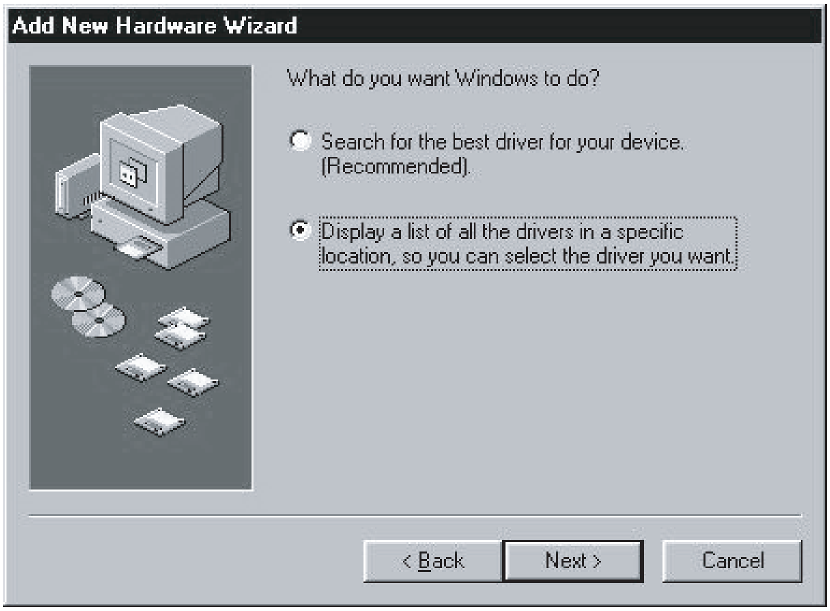
Add New Hardware Wizard

This wizard searches for new drivers for:

PCI Network Controller

A device driver is a software program that makes a hardware device work.

2. A window will be displayed to select the driver search method. Select *Display a list of all ...* and click the **Next** Button.



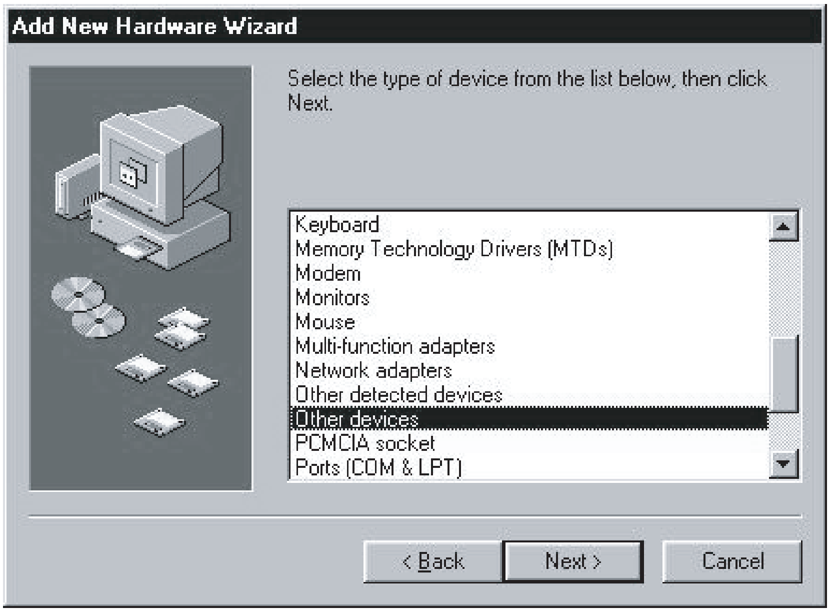
Add New Hardware Wizard

What do you want Windows to do?

○ Search for the best driver for your device. (Recommended).

● Display a list of all the drivers in a specific location, so you can select the driver you want.

3. Select *Other devices* as the device type and click the **Next** Button.

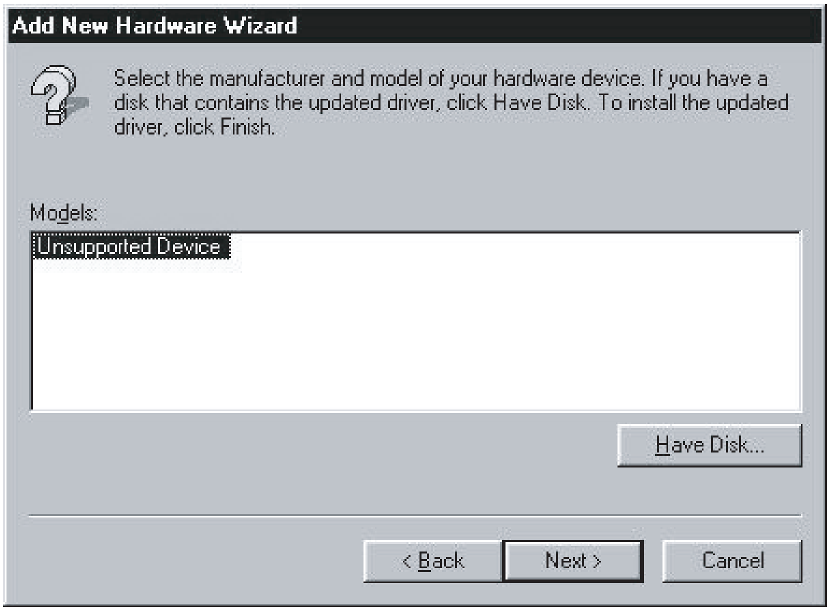


Add New Hardware Wizard

Select the type of device from the list below, then click Next.

Keyboard
Memory Technology Drivers (MTDs)
Modem
Monitors
Mouse
Multi-function adapters
Network adapters
Other detected devices
Other devices
PCMCIA socket
Ports (COM & LPT)

4. Click the **Have Disk** Button to specify the hardware model.



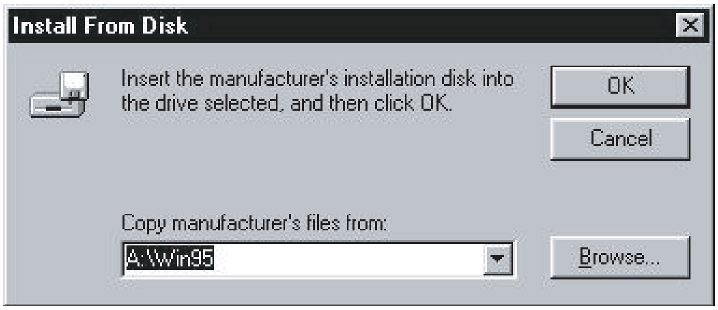
Add New Hardware Wizard

Select the manufacturer and model of your hardware device. If you have a disk that contains the updated driver, click Have Disk. To install the updated driver, click Finish.

Models:

Unsupported Device

5. Insert the Scanner SDK CD-ROM disk in the CD-ROM drive.
6. Specify the CD-ROM drive's Win98 directory as the source location and click the **OK** Button.
   (You can click the **Browse** Button to display a list of the actual directories and select the Win98 directory from that list. The following example window shows the CD-ROM drive as drive A.)



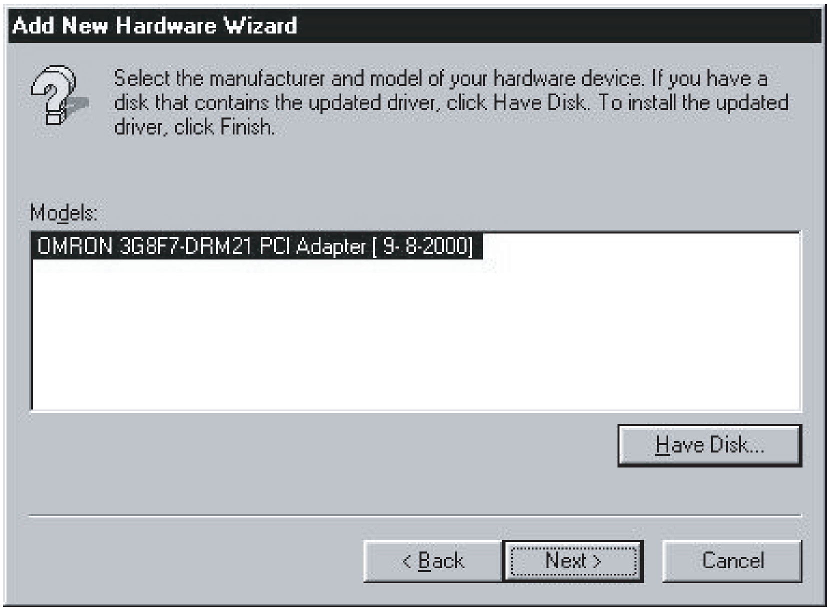
Install From Disk

Insert the manufacturer's installation disk into the drive selected, and then click OK.

Copy manufacturer's files from:

A:\Win95

7. When the driver information has been read from the CD-ROM, select *OMRON 3G8F7-DRM21-E PCI Adapter* as shown in the following diagram.
   Click the **Next** Button.



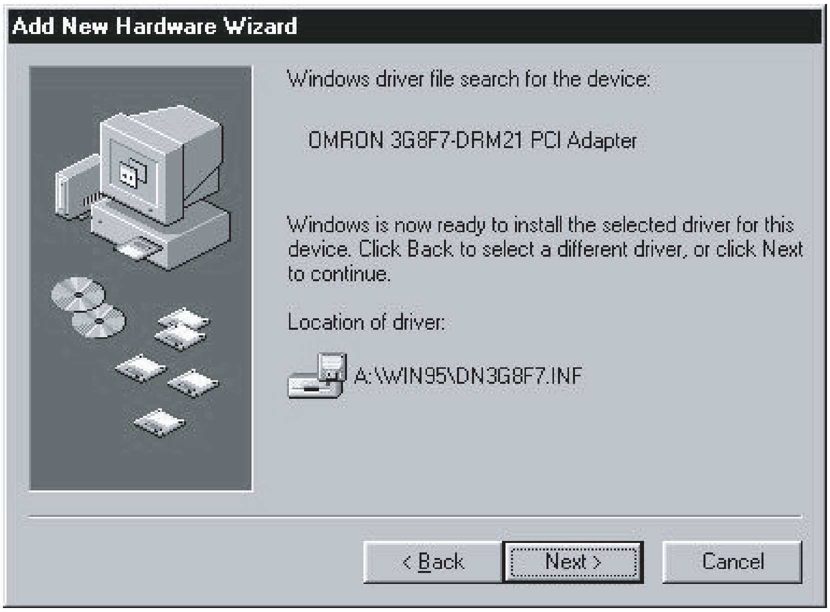
Add New Hardware Wizard

Select the manufacturer and model of your hardware device. If you have a disk that contains the updated driver, click Have Disk. To install the updated driver, click Finish.

Models:

OMRON 3G8F7-DRM21 PCI Adapter [9- 8-2000]

8. Check the displayed message and click the **Next** Button if the correct driver is displayed. The drivers will be installed.



Add New Hardware Wizard

Windows driver file search for the device:

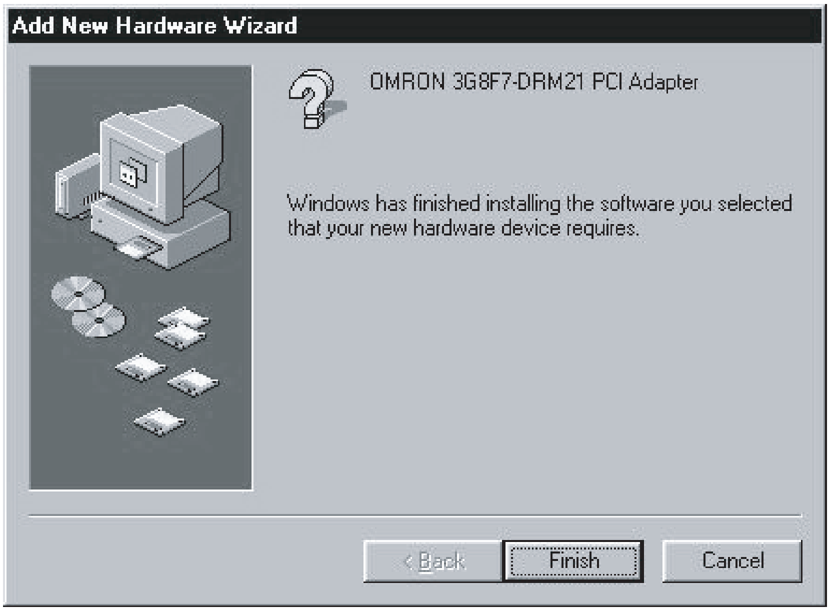
OMRON 3G8F7-DRM21 PCI Adapter

Windows is now ready to install the selected driver for this device. Click Back to select a different driver, or click Next to continue.

Location of driver:

A:\WIN95\DN3G8F7.INF

9. A completion message will be displayed when installation of the Windows 98 drivers is completed. Click the **Finish** Button to complete the installation.



Add New Hardware Wizard

OMRON 3G8F7-DRM21 PCI Adapter

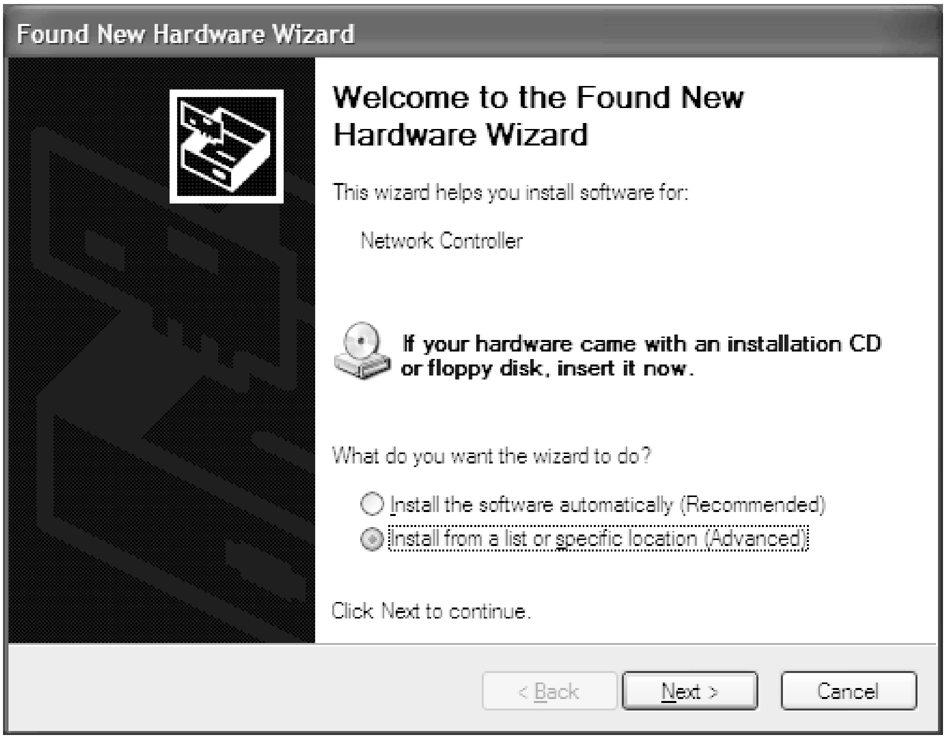
Windows has finished installing the software you selected that your new hardware device requires.

Windows XP Installation

Note You must login to Windows XP with Administrator rights to install the driver.

1,2,3...

1. After the Board is installed in the computer and the computer is started, the Board will be detected as new hardware and the Found New Hardware Wizard will start automatically as shown in the following diagram. Select *Install from a list or specific location (Advanced)*, as shown in the following diagram, and then click the **Next** Button.



Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

This wizard helps you install software for:

Network Controller

If your hardware came with an installation CD or floppy disk, insert it now.

What do you want the wizard to do?

○ Install the software automatically (Recommended)

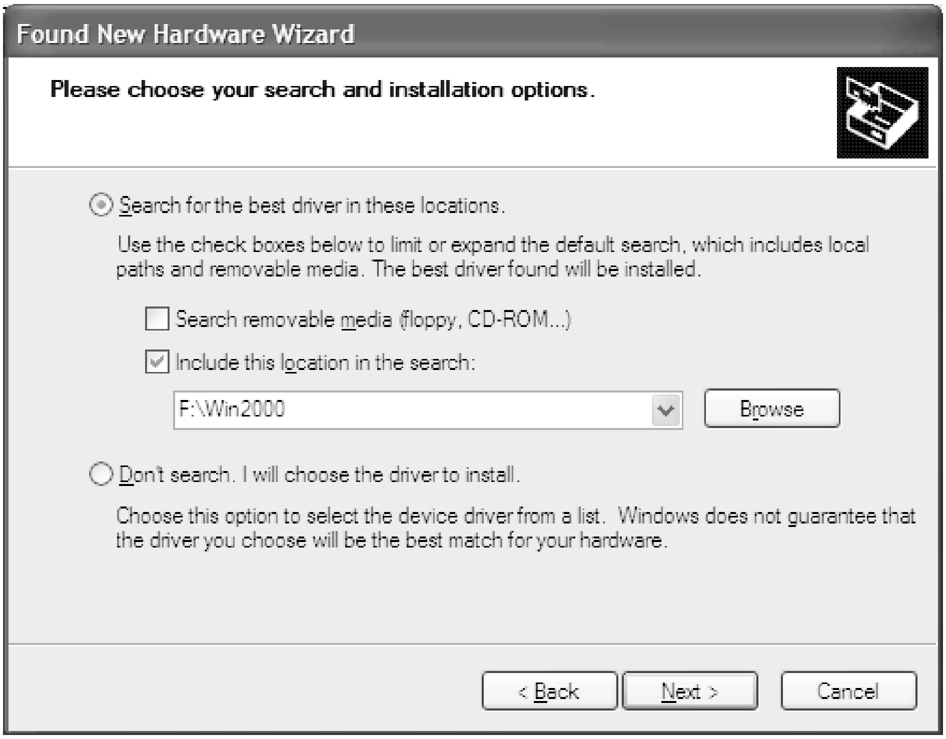
● Install from a list or specific location (Advanced)

Click Next to continue.

2. Insert the Scanner SDK CD-ROM into the CD-ROM drive.

3. Select *Search for the best driver in these locations* for the search and install option, and select the *Include this location in the search* option. Click the **Browse** Button, specify the CD-ROM drive's Win2000 folder (see following diagram), and then click the **Next** Button.

   Note In this example, the CD-ROM drive is drive F.



Found New Hardware Wizard

Please choose your search and installation options.

● Search for the best driver in these locations.

Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.

☐ Search removable media (floppy, CD-ROM...)

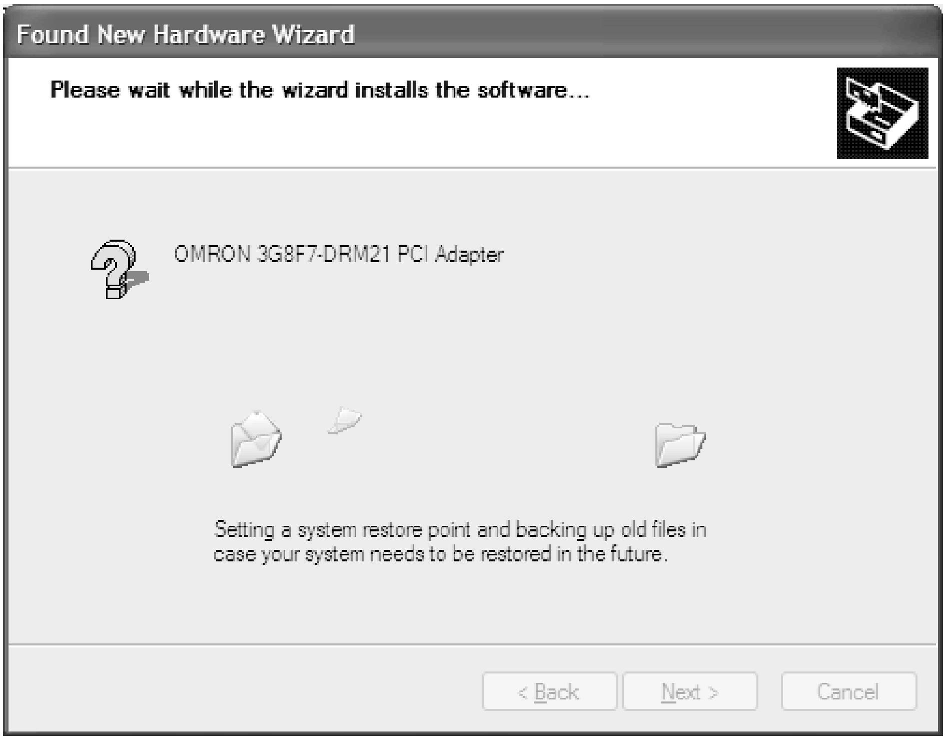
☑ Include this location in the search:

F:\Win2000

○ Don't search. I will choose the driver to install.

Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.

4. The driver installation will start.



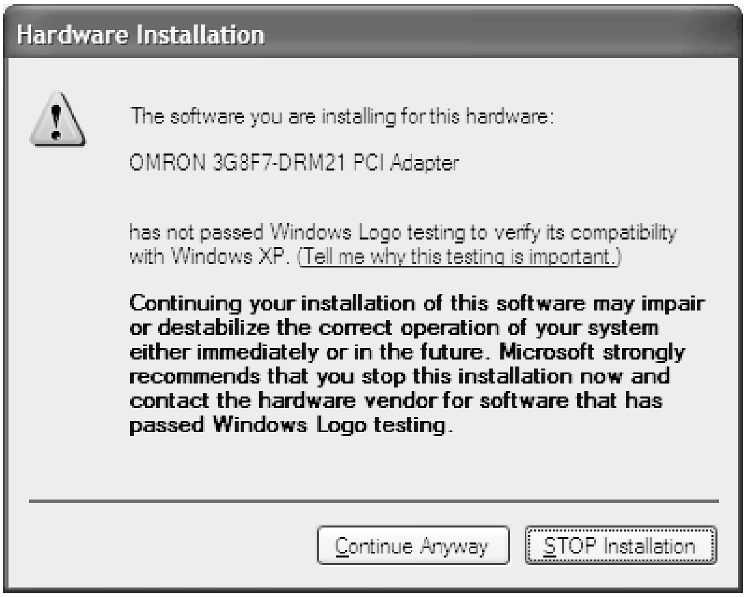
Found New Hardware Wizard

Please wait while the wizard installs the software...

OMRON 3G8F7-DRM21 PCI Adapter

Setting a system restore point and backing up old files in case your system needs to be restored in the future.

5. The following window will be displayed during installation, but it does not indicate an error. Click the **Continue** Button to continue the installation.



Hardware Installation

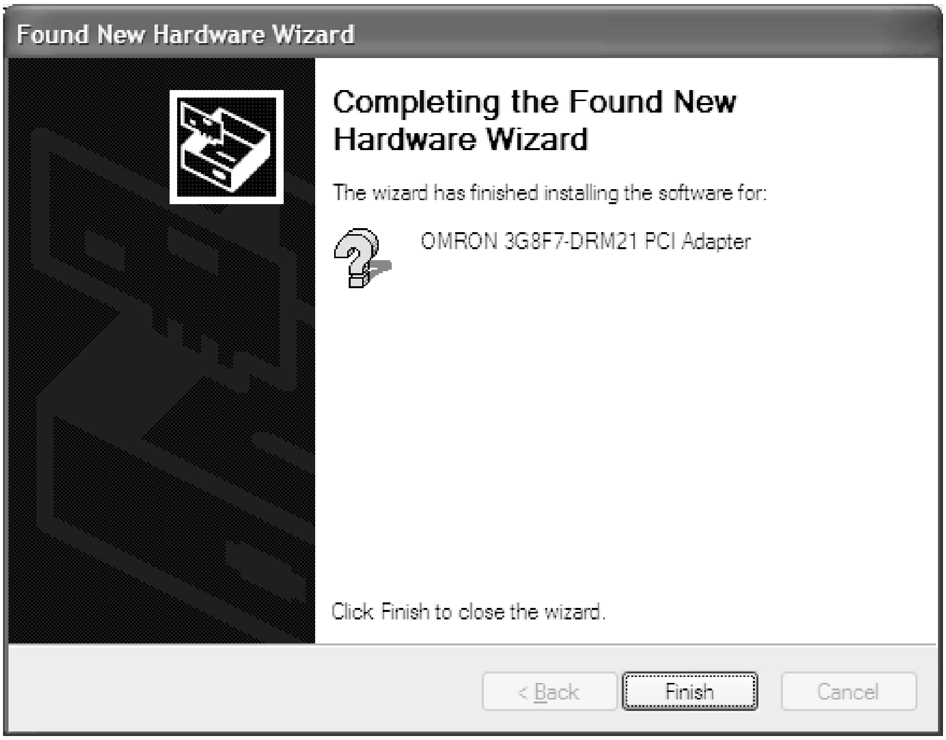
The software you are installing for this hardware:

OMRON 3G8F7-DRM21 PCI Adapter

has not passed Windows Logo testing to verify its compatibility with Windows XP. (Tell me why this testing is important.)

Continuing your installation of this software may impair or destabilize the correct operation of your system either immediately or in the future. Microsoft strongly recommends that you stop this installation now and contact the hardware vendor for software that has passed Windows Logo testing.

6. A completion message will be displayed when installation of the driver has been completed (see diagram.) Click the **Finish** Button to complete the installation procedure.



Found New Hardware Wizard

Completing the Found New Hardware Wizard

The wizard has finished installing the software for:

OMRON 3G8F7-DRM21 PCI Adapter

Click Finish to close the wizard.

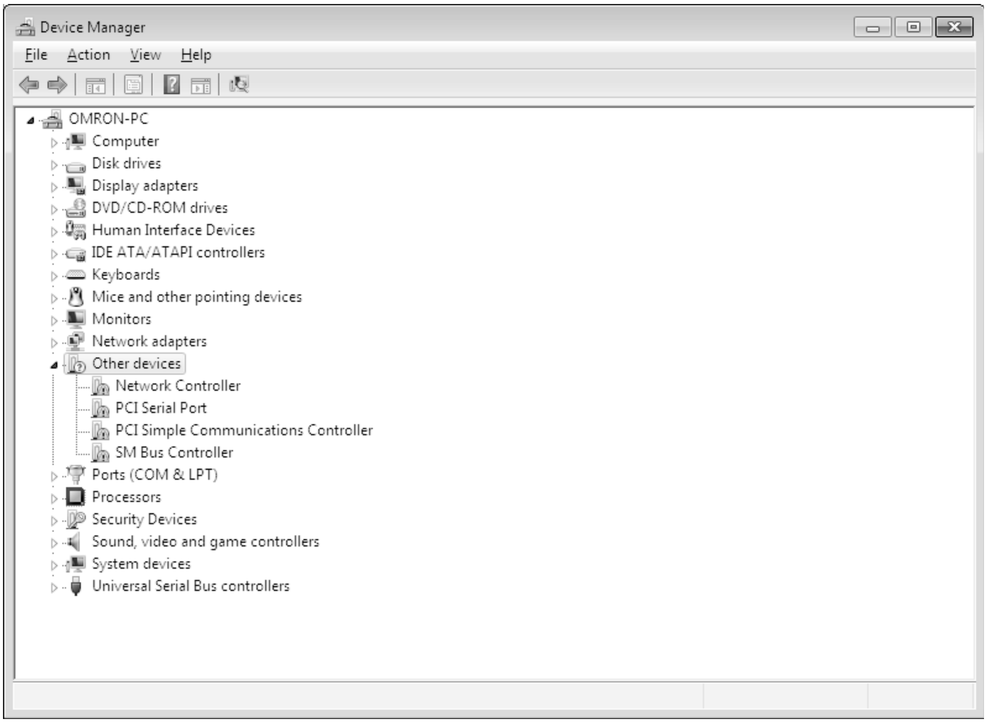
Windows 7 Installation

Note For Windows 7, you must log in with administrator rights to install the driver.

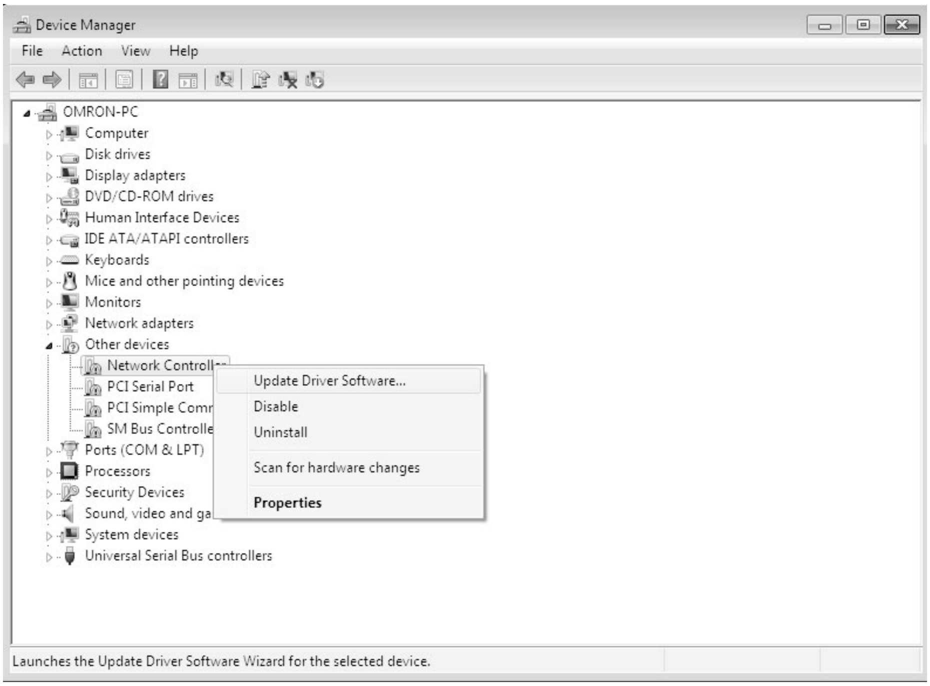
1,2,3... 1. After the Board is installed in the computer, start the Device Manager. New hardware will be detected automatically.

Open the Device Manager,[*1] and double-click **Other devices**.

[*1] To open the Device Manager, click the Windows Start Button and select **Control Panel, Hardware and Sound**, and **Device Manage**r in that order.

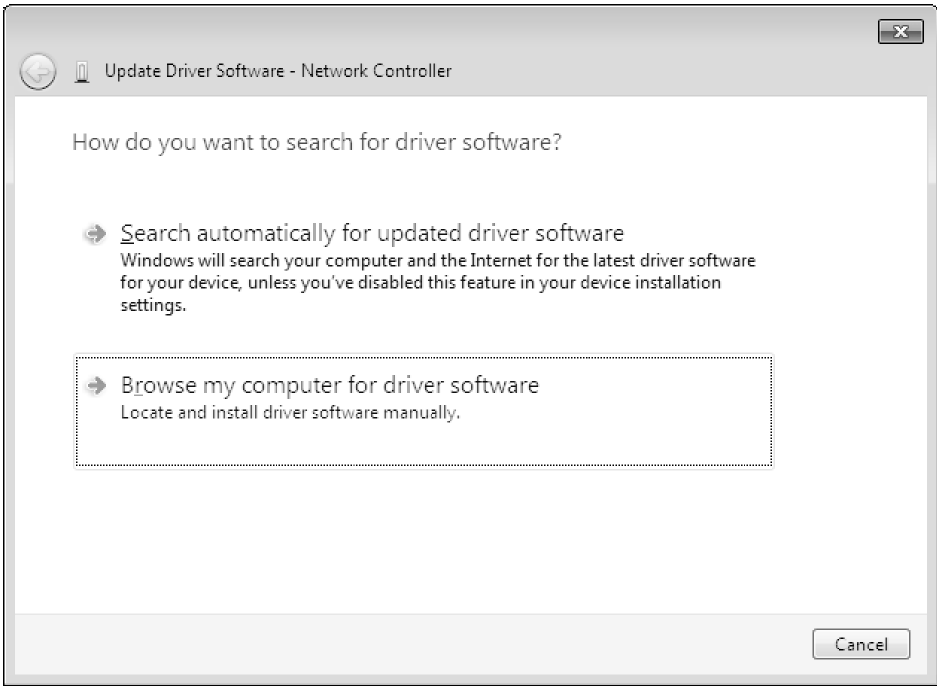


2. The **Network Controller** will appear under **Other devices**.
   Right-click **Network Controller** and then select ***Update Driver Software*** from the menu.



Launches the Update Driver Software Wizard for the selected device.

3. *How do you want to search for driver software?* will be displayed.
   Click **Browse my computer for driver software**.



How do you want to search for driver software?

Search automatically for updated driver software

Windows will search your computer and the Internet for the latest driver software for your device, unless you've disabled this feature in your device installation settings.

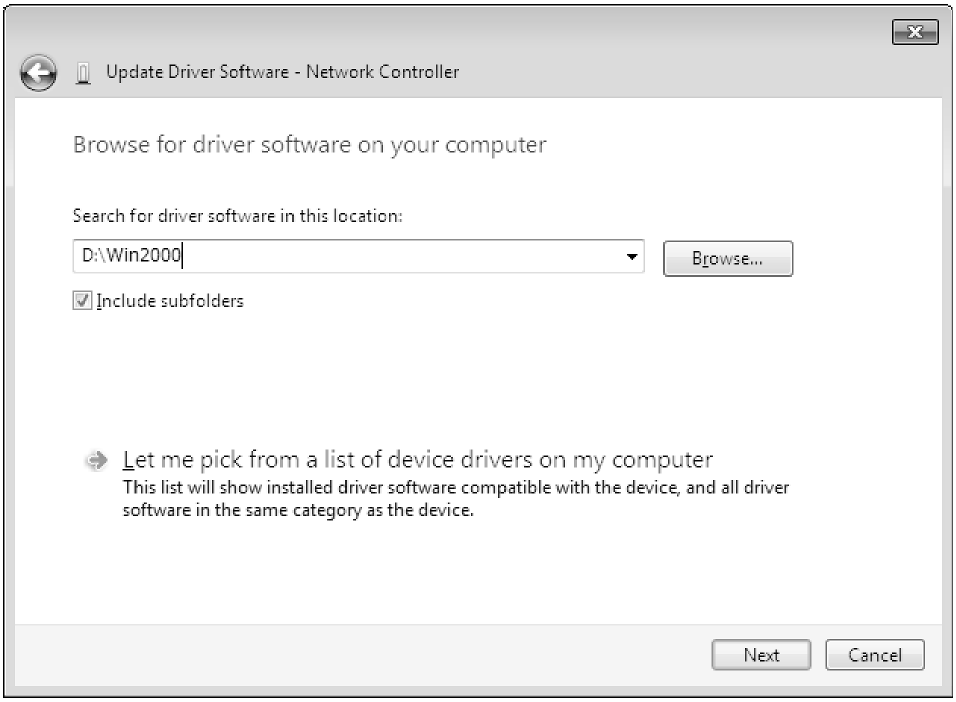
Browse my computer for driver software

Locate and install driver software manually.

4. Place the CD-ROM containing the Scanner SDK into the CD-ROM drive.

5. *Browse for driver software on your computer* will be displayed.
   Click the **Browse** Button, specify the Win2000 folder on the CD-ROM drive (see following figure), and click the **Next** Button

* The following figure shows an example for which drive D is the CD-ROM drive.



Update Driver Software - Network Controller

Browse for driver software on your computer

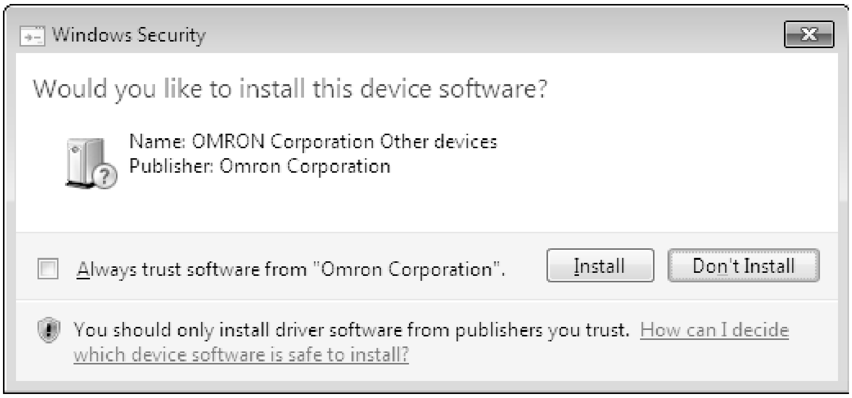
Search for driver software in this location:

D:\Win2000

☑ Include subfolders

Let me pick from a list of device drivers on my computer
This list will show installed driver software compatible with the device, and all driver software in the same category as the device.

6. The following dialog box will be displayed. Click **Install this driver software anyway** to start installation.



Windows Security

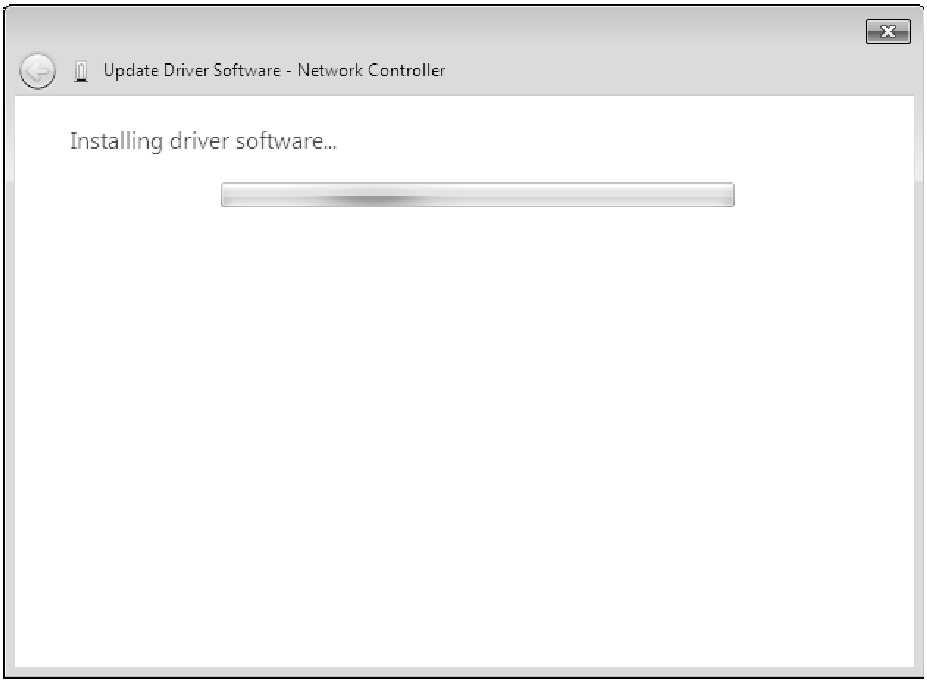
Would you like to install this device software?

Name: OMRON Corporation Other devices
Publisher: Omron Corporation

□ Always trust software from "Omron Corporation".

You should only install driver software from publishers you trust. How can I decide which device software is safe to install?

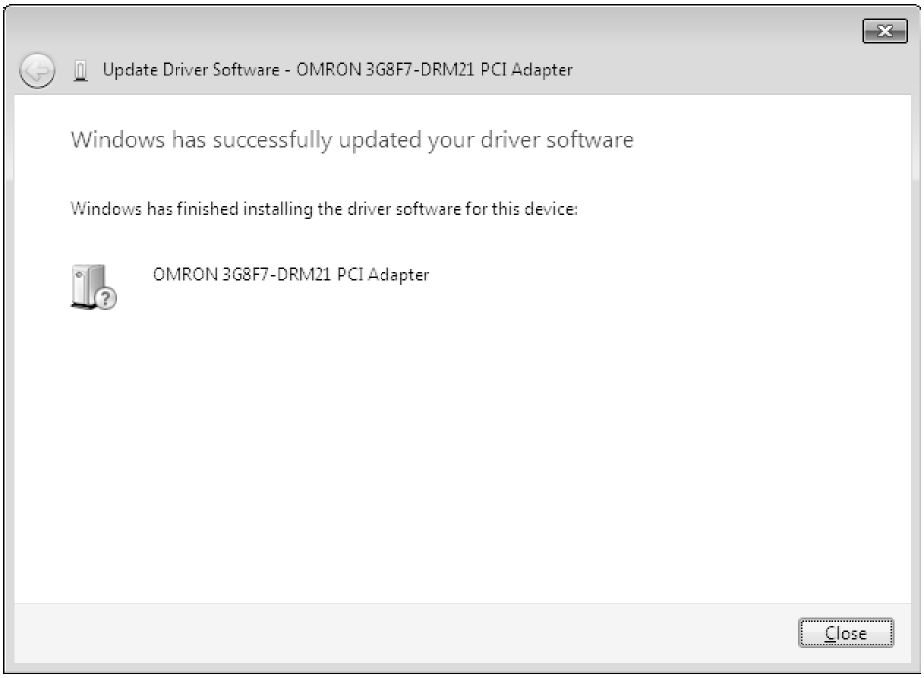
7. The installation will start.



Update Driver Software - Network Controller

Installing driver software...

8. A completion message (see the following figure) is displayed after the installation process is completed.
   Click the **Close** Button to complete driver installation.



Update Driver Software - OMRON 3G8F7-DRM21 PCI Adapter

Windows has successfully updated your driver software

Windows has finished installing the driver software for this device:

OMRON 3G8F7-DRM21 PCI Adapter

2-4 Installing the Scanner SDK

This section describes how to install the Scanner SDK.

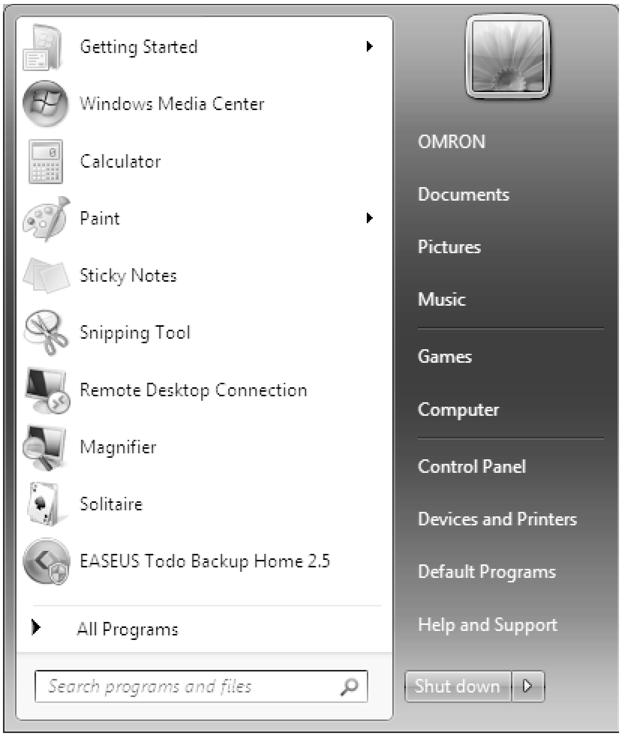
Note The procedure and displays will vary with the Windows OS that you are using.
This procedure is for Windows 7.

To install software on Windows 7, you must log in with administrator rights.

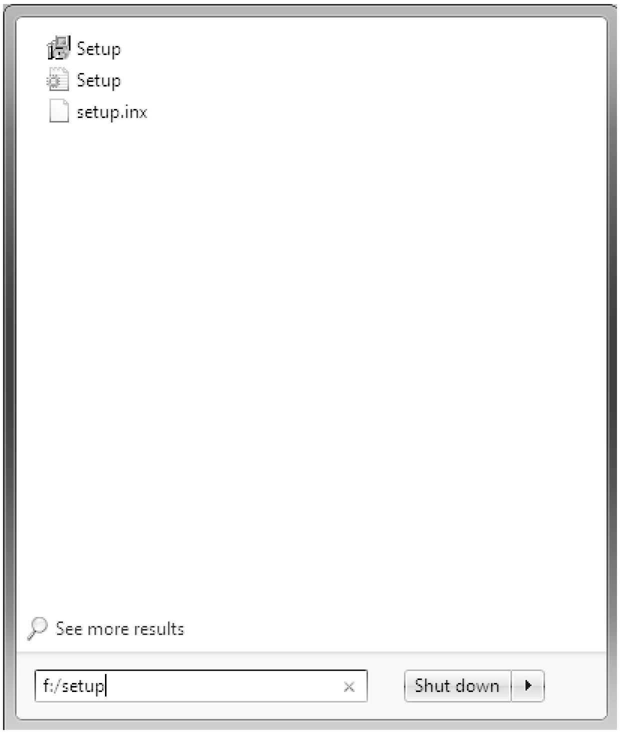
Installation Procedure

1,2,3...

1. End all applications that are currently running.
2. Place the CD-ROM containing the Scanner SDK into the CD-ROM drive.
3. Click the Windows **Start** Menu and select *Search for programs and files*.



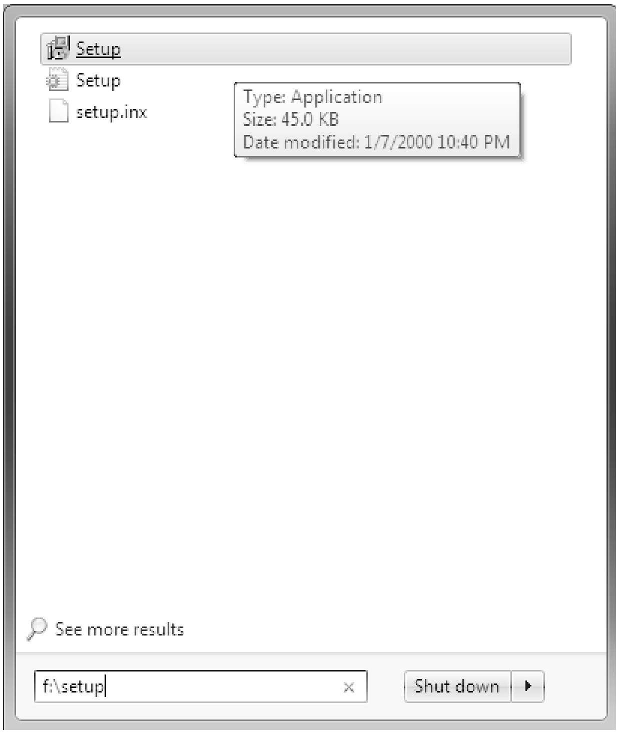
4. Enter *d:\setup* in the **Search for programs and files** box that is displayed. Specify the drive into which you inserted the CD-ROM drive. For example, if you inserted the CD-ROM to the F drive, enter *f:\setup.*



Setup
Setup
setup.inx

f:/setup

5. Click **setup** in the search results.



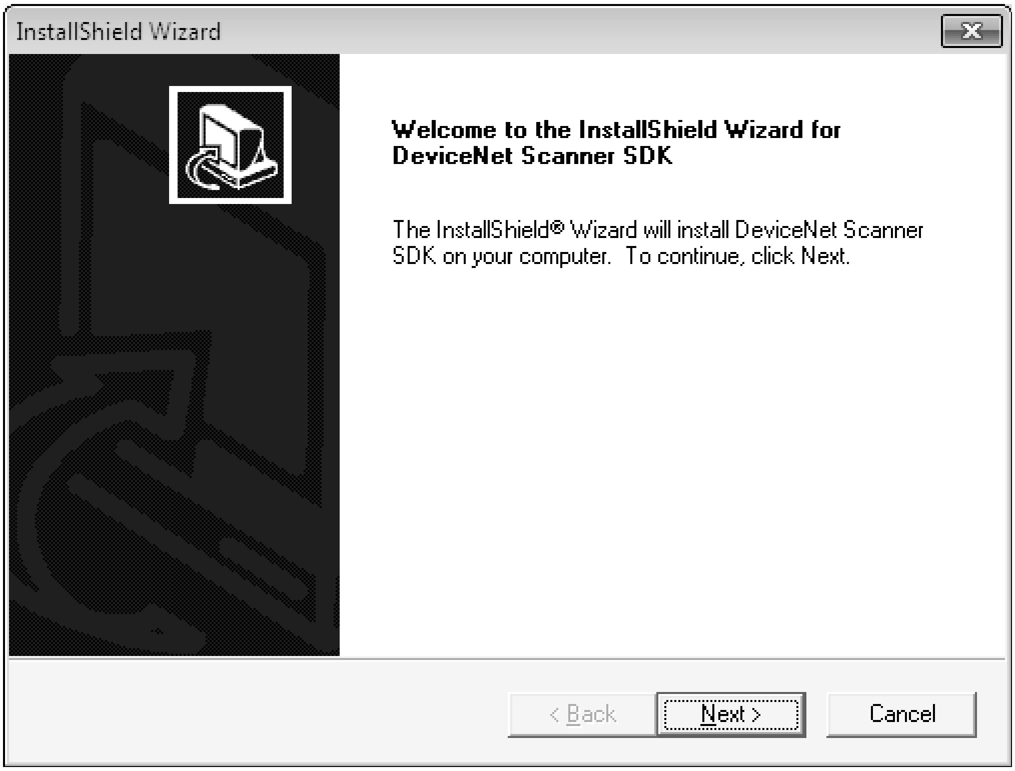
Setup
Setup
setup.inx

Type: Application
Size: 45.0 KB
Date modified: 1/7/2000 10:40 PM

f:\setup

6. The User Account Control Dialog Box will be displayed. Click the **Yes** Button to continue unless there is a problem.
7. Installation of the software will start.
   The InstallShield will start preparations for installation.

8. When the setup dialog box is displayed, click the **Next** Button.



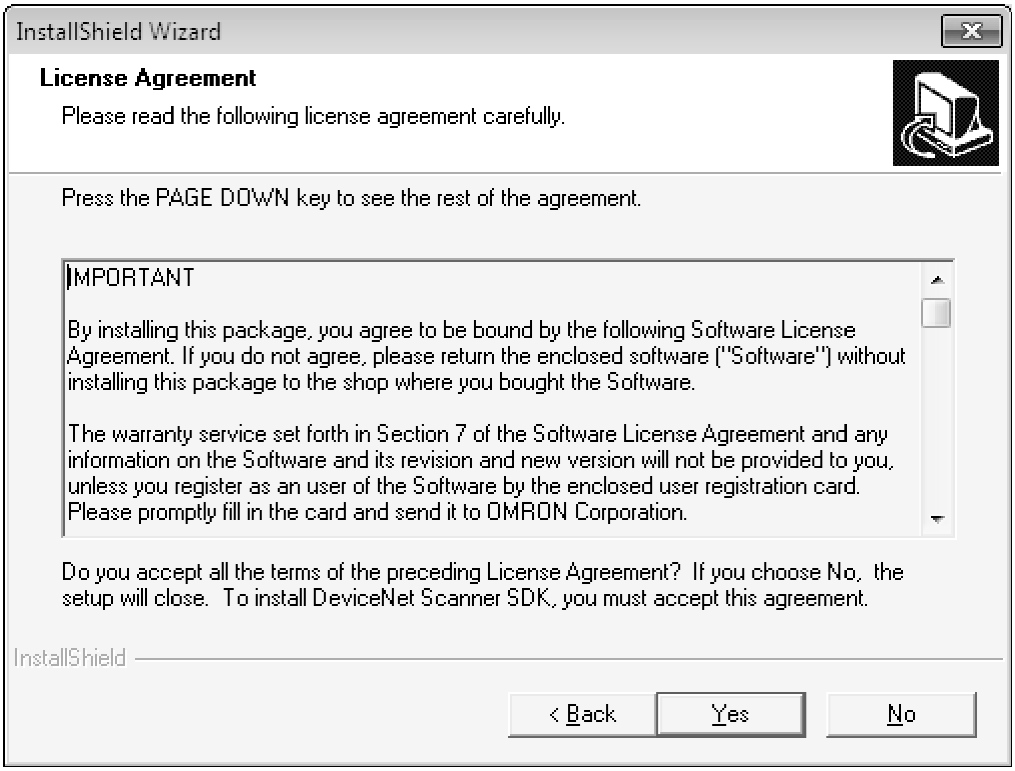
InstallShield Wizard

Welcome to the InstallShield Wizard for DeviceNet Scanner SDK

The InstallShield® Wizard will install DeviceNet Scanner SDK on your computer. To continue, click Next.

9. The software license agreement will be displayed.
   Check all of the items in the agreement and click the **Yes** Button if you agree to the license agreement.
   If you do not agree, click the **No** Button.

   Note If you do not agree, the installation will be stopped.



InstallShield Wizard

License Agreement

Please read the following license agreement carefully.

Press the PAGE DOWN key to see the rest of the agreement.

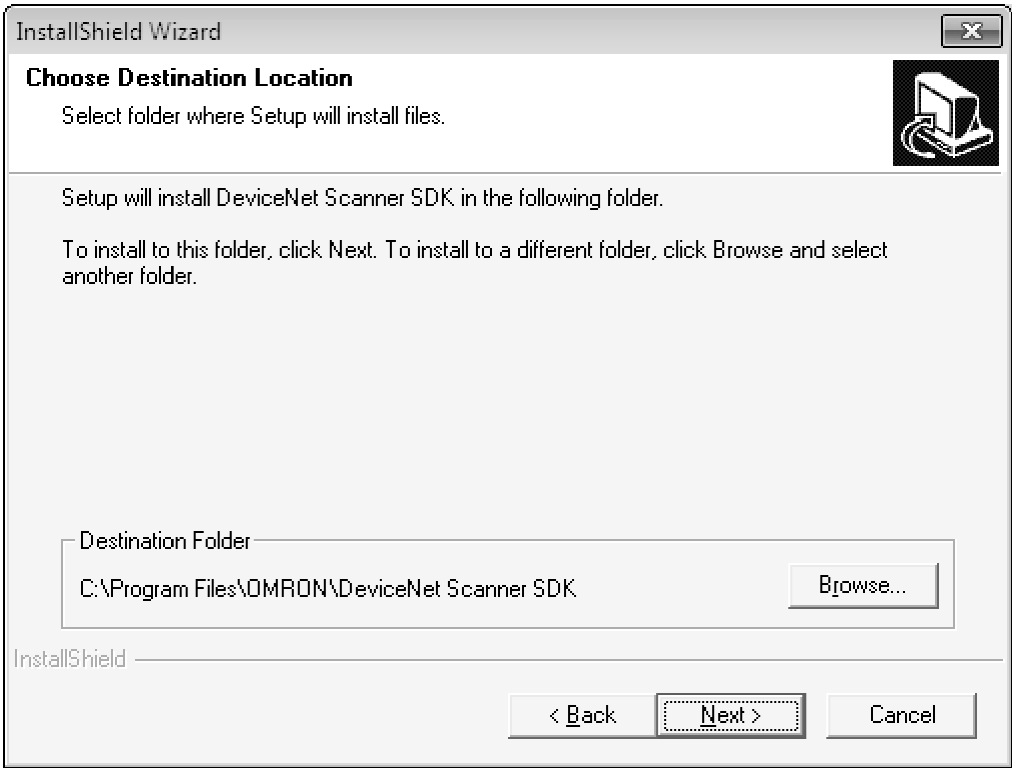
IMPORTANT

By installing this package, you agree to be bound by the following Software License Agreement. If you do not agree, please return the enclosed software ("Software") without installing this package to the shop where you bought the Software.

The warranty service set forth in Section 7 of the Software License Agreement and any information on the Software and its revision and new version will not be provided to you, unless you register as an user of the Software by the enclosed user registration card. Please promptly fill in the card and send it to OMRON Corporation.

Do you accept all the terms of the preceding License Agreement? If you choose No, the setup will close. To install DeviceNet Scanner SDK, you must accept this agreement.

10. Select the installation folder.
    The following folder is the default installation folder.
    C:\Program Files\OMRON\DeviceNet Scanner SDK
    To change the folder, click the **Browse** Button and specify the drive and folder. (If you specify a folder that does not exist, it will be created automatically.)
    After you specify the drive and folder, click the **Next** Button.
    A dialog box is displayed to specify the folder in which to register the program in the Windows Start Menu.



InstallShield Wizard

Choose Destination Location

Select folder where Setup will install files.

Setup will install DeviceNet Scanner SDK in the following folder.

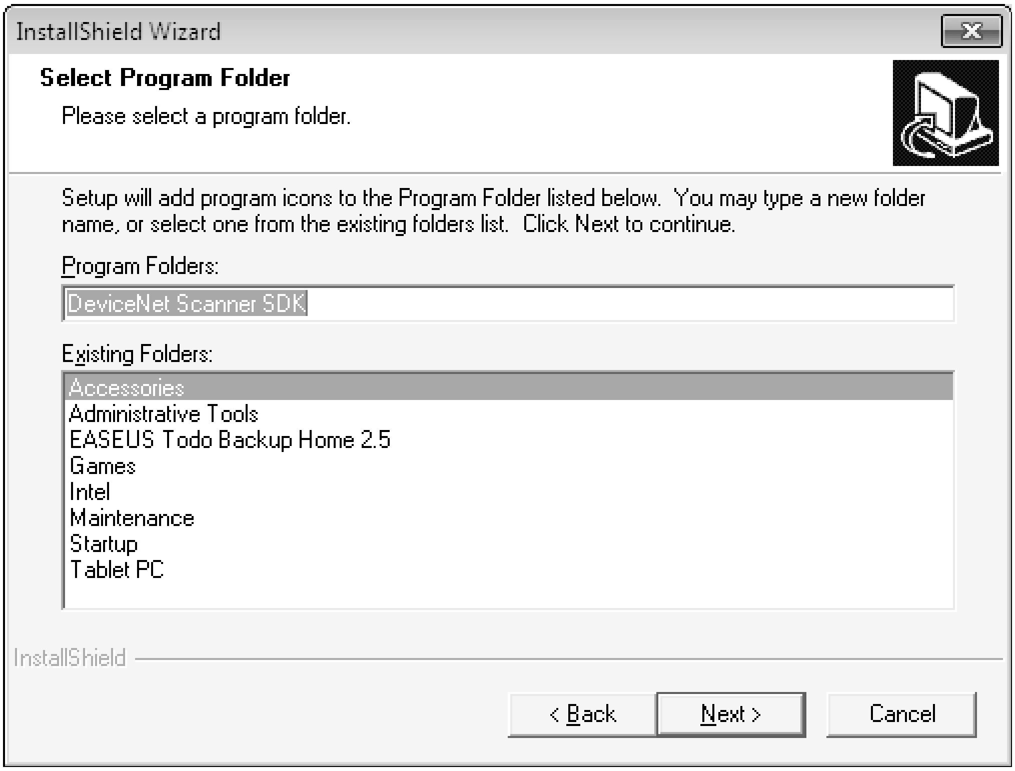
To install to this folder, click Next. To install to a different folder, click Browse and select another folder.

Destination Folder

C:\Program Files\OMRON\DeviceNet Scanner SDK

11. Specify the program folder in the Windows Start Menu in which to create a shortcut to the DeviceNet Scanner SDK.
    The default is to create shortcut in a program folder called *DeviceNet Scanner SDK*.
    To change the program folder, select from the existing folders or enter the folder name directly.

    (If the specified folder does not exist, it will be created automatically.)
    After you specify the drive and folder, click the **Next** Button.
    The installation will be executed and the files will be copied.
    The progress of the installation will be displayed as a percentage.



InstallShield Wizard

Select Program Folder

Please select a program folder.

Setup will add program icons to the Program Folder listed below. You may type a new folder name, or select one from the existing folders list. Click Next to continue.

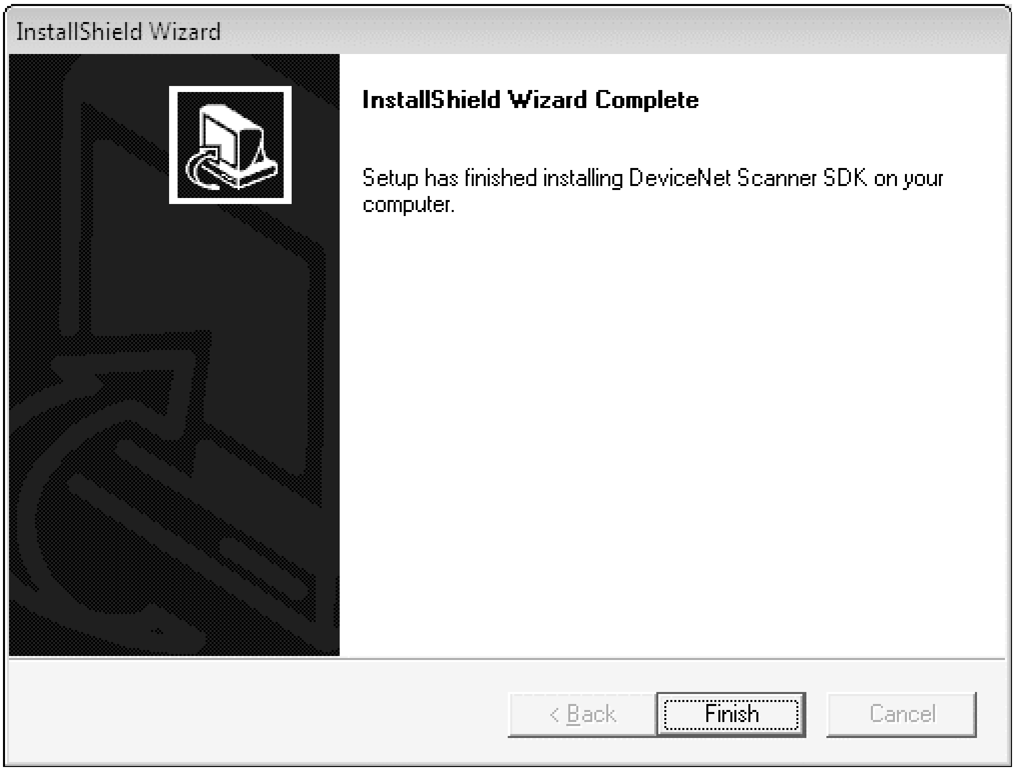
Program Folders:

DeviceNet Scanner SDK

Existing Folders:

Accessories
Administrative Tools
EASEUS Todo Backup Home 2.5
Games
Intel
Maintenance
Startup
Tablet PC

12. A completion message (see the following figure) is displayed after the installation process is completed.
    Click the **Finish** Button to complete the installation.



InstallShield Wizard

InstallShield Wizard Complete

Setup has finished installing DeviceNet Scanner SDK on your computer.

Installed Files

When the Scanner SDK is installed, the following folders are created and the required files are installed.

The following table shows the folders and files that are created and installed when the default folders are used.:

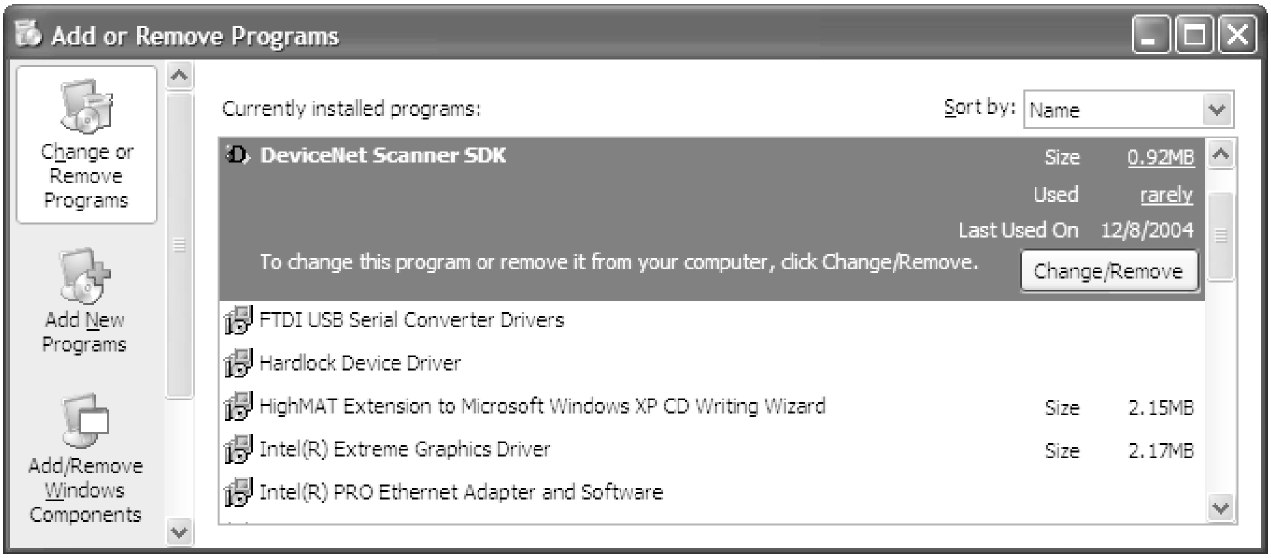
Default Folder	Contents
\Program Files\OMRON\DeviceNet Scanner SDK\Manual\	The PDF file of this manual is installed.
\Program Files\OMRON\DeviceNet Scanner SDK\Program\	Execution files for the sample program are installed.
\Program Files\OMRON\DeviceNet Scanner SDK\SDK\Include\	A sample Include file is installed.
\Program Files\OMRON\DeviceNet Scanner SDK\SDK\Lib\	The library files used for static links in Microsoft Visual C++ are installed.
\Program Files\OMRON\DeviceNet Scanner SDK\SDK\Sample\	A sample program is installed.

Removing the DeviceNet Scanner SDK Software

If the DeviceNet Scanner SDK Software is no longer needed, the files and other information can be removed with the following procedure. (Only the installed files and information will be removed. Files created later will not be removed.)

1,2,3...

1. Select ***Start/Settings/Control Panel*** from the Windows **Start** Button.
2. Double-click the **Add/Remove Programs** Icon in the Control Panel.
3. Select *DeviceNet Scanner SDK* from the list. Click the **Add/Remove** Button.



Currently installed programs:

DeviceNet Scanner SDK Size 0.92MB

Used rarely

Last Used On 12/8/2004

To change this program or remove it from your computer, click Change/Remove.

FTDI USB Serial Converter Drivers

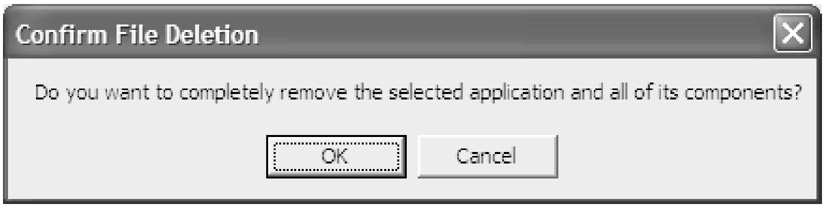
Hardlock Device Driver

HighMAT Extension to Microsoft Windows XP CD Writing Wizard Size 2.15MB

Intel(R) Extreme Graphics Driver Size 2.17MB

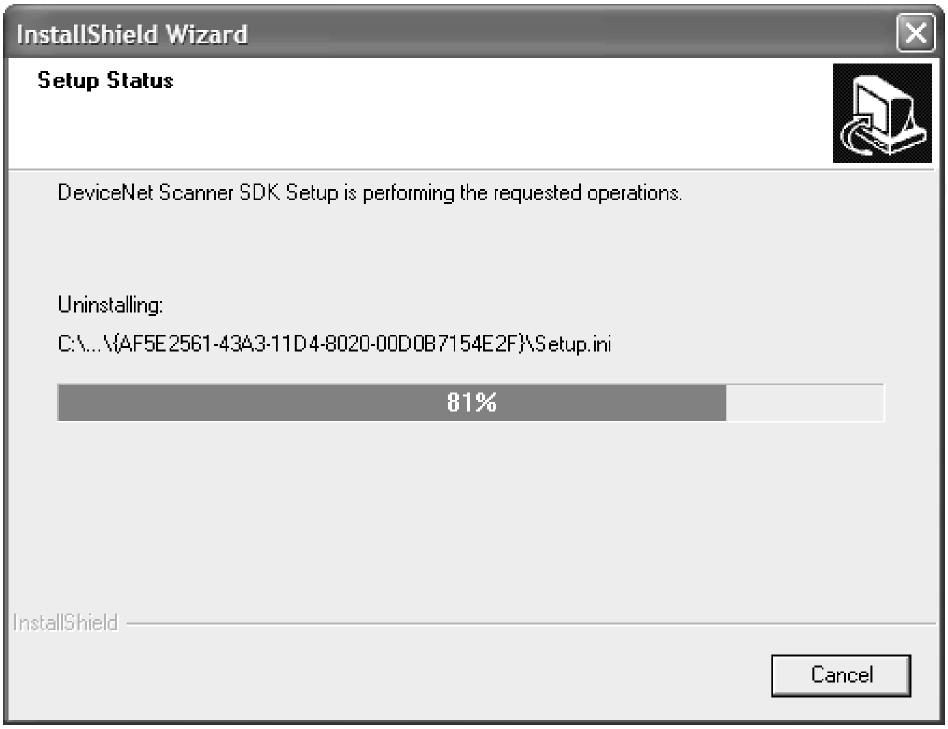
Intel(R) PRO Ethernet Adapter and Software

4. A confirmation message will be displayed. Click the **OK** Button to proceed.



Do you want to completely remove the selected application and all of its components?

5. Removal of the application will start.



Setup Status

DeviceNet Scanner SDK Setup is performing the requested operations.

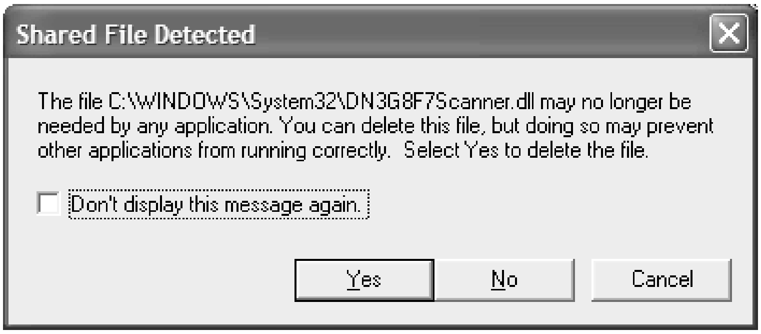
Uninstalling:

C:\...\{AF5E2561-43A3-11D4-8020-00D0B7154E2F}\Setup.ini

81%

6. During removal, messages asking if detected shared files are to be deleted may be displayed (see diagram below).
   - If other DeviceNet applications, such as the Configurator, NetXServer, or Analyzer, are installed, click the **No** Button. Do not delete the shared files.

- If no other DeviceNet applications are installed, click the **Yes** Button. The shared files can be deleted.

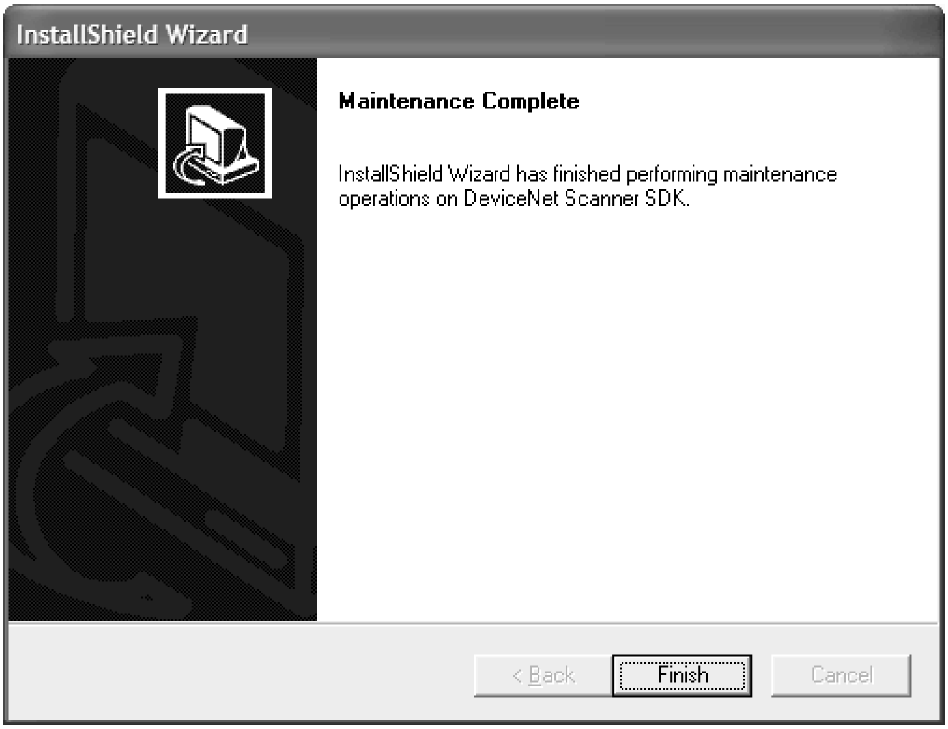


Shared File Detected

The file C:\WINDOWS\System32\DN3G8F7Scanner.dll may no longer be needed by any application. You can delete this file, but doing so may prevent other applications from running correctly. Select Yes to delete the file.

□ Don't display this message again.

7. A completion message will be displayed when removal of the driver has been completed (see diagram.) Click the **Finish** Button to complete the installation procedure.



InstallShield Wizard

Maintenance Complete

InstallShield Wizard has finished performing maintenance operations on DeviceNet Scanner SDK.

2-5 DeviceNet Connections

Connect the DeviceNet communications cables after installing the Board.

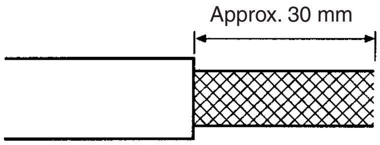
This section explains how to prepare and connect the communications cables to the DeviceNet PCI Board only. Refer to the *DeviceNet Operation Manual (W267)* for details on connecting cables to Slaves.

2-5-1 Attaching Connectors to the DeviceNet Cable

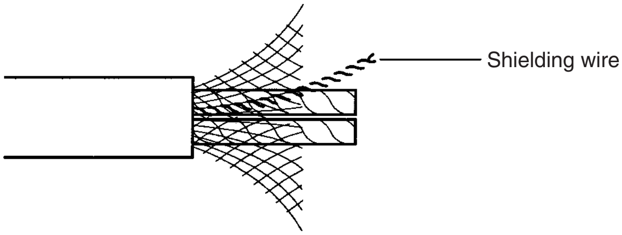
This section explains how to attach connectors to the network communications cables. Use the following procedures prepare the communications cables and attach connectors.

1,2,3...

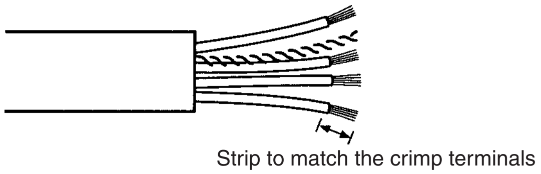
1. Remove about 30 mm of the cable sheathing, being careful not to damage the woven shielding underneath. Do not remove too much sheathing; removing too much of the sheathing can result in short circuits and increase the effect of noise.



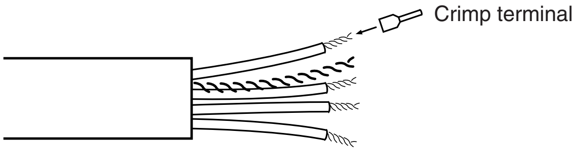
2. Carefully peel back the woven shielding. The cable contains a shielding wire along with the signal lines and power lines. The shielding wire is stiffer than the woven shielding, so it can be identified by touch.



3. Remove the exposed woven shielding, remove the aluminum tape from the signal and power lines, and strip the covering from the signal and power lines to the proper length for the crimp terminal connectors being used. Twist the wire strands on each of the signal and power lines so that there are no loose strands.



4. Attach the crimp terminals (solderless pin terminals) to the lines and use the proper Crimping Tool to crimp the terminal securely.

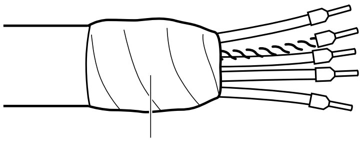


Note We recommend using the following crimp terminals and crimping tools.

- AI Series from Phoenix Contact

Connector **Cable type**		**XW4B-05C1-H1-D** **XW4B-05C1-V1R-D** **MSTB2.5/5-ST-5.08AU**	**XW4B-05C4-TF-D** **XW4B-05C4-T-D**	**XW4G-05C1-H1-D** **XW4G-05C4-TF-D**	**Crimping Tool**
Thin cable	Signal lines	AI 0.25-6YE	AI 0.25-8YE	AI 0.25-8YE	CRIMPFOX ZA3
	Power lines	AI 0.5-6WH	AI 0.5-10WH	AI 0.5-10WH	
Thick cable	Signal lines	AI -6	AI 10	AI 10	
	Power lines	AI 2.5-8BU	AI 2.5-10BU	AI 2.5-10BU	

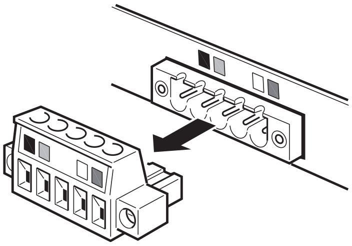
5. Cover the end of the cable with electrical tape or heat-shrink tubing as shown in the following diagram.



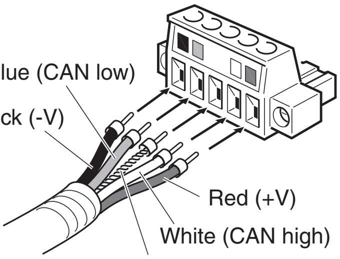
2-5-2 Connecting Communications Cables

1,2,3...

1. Remove the connector from the Board's DeviceNet Communications Connector. (It isn't necessary to remove the connector from the Board if it can be wired in place.)



2. Orient the connector properly and then insert the lines in order from left to right: black, blue, shield, white, and then red.



Note

(1) Loosen the screws for securing the connector wires before inserting signal lines, power lines, or the shield wire. If the screws are not loosened sufficiently, the wires cannot be inserted into the correct position. They will enter the gap at the back where they cannot be secured in place.

(2) The connector and Board have colored stickers that match the wire colors. The wire colors can be checked against the sticker color to check that the cables are wired correctly.

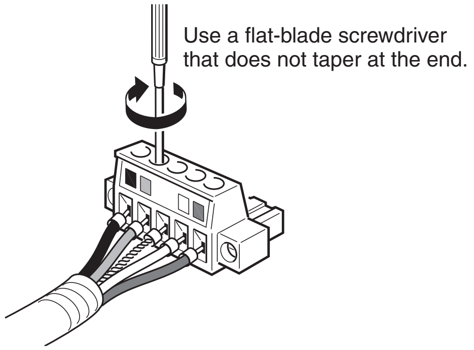
(3) The wire colors are listed in the following table.

Color	Signal	Symbol
Black	Communications power supply (negative)	V−
Blue	Signal LOW side	CAN L
---	Shield	S
White	Signal HIGH side	CAN H
Red	Communications power supply (positive)	V+

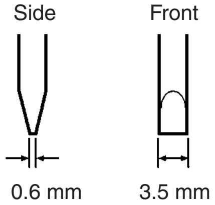
3. Tighten the line set screws for each line in the connector. Tighten the screws to a torque between 0.25 and 0.3 N·m.

You will not be able to tighten these screws with a normal screwdriver, which tapers at the end. You will need a screwdriver that does not taper at the end, such as a large precision screwdriver.

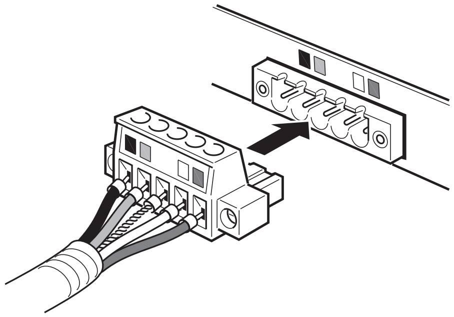
When using a Thick Cable, allow sufficient cable to ensure that the tension of the cable does not disconnect the connector.



Note The following diagram shows the dimensions of the OMRON XW4Z-00C screwdriver, which is ideal for these DeviceNet connectors.

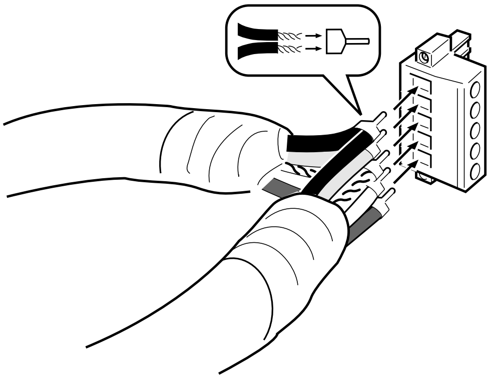


4. Align the connector to the Board's Connector and then insert the connector until it is securely set in place. Firmly tighten the screws at both ends of the connector. Tighten the connector mounting screws to a torque between 0.25 and 0.3 N·m.



Standard Connector (Thin Cables Only)

When thin cable is being used, a multi-drop connection can be made by inserting each pair of wires into a single same pin terminal and crimping them together.



Note We recommend using the following PHOENIX CONTACT terminal for this type of multi-drop connection.

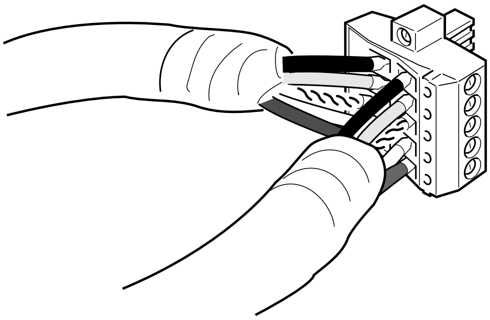
Connector	Crimp terminals	Crimping Tool
XW4B-05C1-H1-D MSTB2.5/5-ST-5.08 XW4B-05C1-V1R-D	AI-TWIN2×0.5-8WH from Phoenix Contact	CRIMPFOX UD6 or CRIMPFOX ZA3
XW4G-05C1-H1-D	H0.5/16.5 ZH from Weidmuller	

Multi-drop Connector

The following OMRON Multi-drop Connectors (sold separately) can be used to make a multi-drop connection with either thin or thick cable.

- XW4B-05C4-T-D Straight Multi-drop Connector without Attachment Screws
- XW4B-05C4-TF-D Straight Multi-drop Connector with Attachment Screws
- XW4G-05C4-TF-D Straight Multi-drop Clamp Connector with Attachment Screws

In some cases, the Multi-drop Connector cannot be used because there is not enough space and other Units or connectors get in the way.



Note 1. Before connecting the communications cables, turn OFF the power supply to all PCs, Slaves, and communications power supplies.

2. Use crimp terminals for wiring. Connecting bare twisted wires can cause the cables to come OFF, break, or short circuit, most likely resulting in incorrect operation and possibly damage to the Units.

3. Use suitable crimp tools and crimping methods when attaching crimp terminals. Consult the manufacturer of the tools and terminals you are using. Inappropriate tools or methods can result in broken wires.

4. Be extremely careful to wire all signal lines, power lines, and shielding wire correctly.

5. Tighten all set screws firmly. Tighten to a torque of between 0.25 and 0.3 N·m.
6. Wire the signal lines, power lines, and shielding wire so that they do not become disconnected during communications.
7. Do not pull on communications cables with excessive force. They may become disconnected or wires may break.
8. Allow leeway so that communications cables do not have to be bent further than natural. The Cables may become disconnected or wires may break if the cables are bent too far.
9. Never place heavy objects on communications cables. They may break.
10. Double-check all wiring before turning ON the power supply.

SECTION 3
Using API Functions

This section provides flowcharts showing how to use the API functions as well as precautions to observe when using the API functions. Refer to this section when actually writing the applications required to use the DeviceNet PCI Board.

3-1 Application Development Environments

Recommended Development Environment

Microsoft Visual C++ (Ver. 6.0 or later)

Precautions when Using Other Development Environments

Whether API can be used depends on the development environment.
Some examples are given below for development environments. Confirm details with the provider of the development environment.

Microsoft Visual C++ Version 6.0

- API functions can be used without declaring them.

Microsoft Visual Basic 6.0

- API functions other than explicit messages can be used, but they must be declared.

Microsoft Visual Basic 2005

- API functions cannot be used (even if you declare them).

3-2 API Functions

The PCI Board provides the following API functions.

API Functions for Managing the Board

Board Service API Functions

Board Service API Functions check the Scanner SDK and driver versions, open/close the Board, and connect/disconnect the Board to/from the network. The PCI Board must be open and connected to the network to be used for communications.

PCI Board Interrupt Service API Functions

Interrupt Service API functions detect interrupts from the PCI Board. Two kinds of interrupts can be detected.

Interrupt	Name	Function
BD_WDT	Board Watchdog Timer Timeout Interrupt	This interrupt occurs when the Board stops operation as a result of an error. The error can be cleared by detecting this interrupt.
SCAN	One-scan (Communications Cycle) Completed Interrupt	This interrupt occurs each time one communications cycle is completed during I/O communications. The latest input data (IN data) can be retrieved when this interrupt occurs.

For interrupts to be detected, the interrupt control register must be set for the Board to send interrupts to the computer. There are two ways the computer can detect interrupts. Refer to *3-3 Checking Events* for detection methods.

Master API Functions

Scan List Operation API Functions

Slave information must be registered in a scan list to use the Master function to perform I/O communications with Slaves. The Scan List Operation API functions are used to create scan lists. Refer to *Parameter Operation API Functions* on page 47 for information on the relationship between a scan list and the Scan List Operation API functions.

I/O Communications Service API Functions

I/O Communications Service API functions are used to start and stop I/O communications using the Master function, set the communications cycle time, obtain real Slave information, etc.

I/O Data Access Service API Functions

I/O Data Access Service API functions are used to read and write I/O data. The PCI Board has two memory areas: the I/O area which can be read from or written to using functions, and the Board's internal buffer. Only the internal buffer is used during I/O communications. Set to write all OUT data first, refresh data between the two areas, and then obtain all IN data when I/O data is to be used by user applications.

Slave Function API Functions

Slave Scan List Operation API Functions

To use the Slave function, information to enable operation as a slave must be registered in a slave scan list. The Slave Scan List Operation API functions are used to register a slave scan list. Refer to *Parameter Operation API Functions* on page 47 for information on the relationship between slave scan lists and Slave Scan List Operation API function.

I/O Communications Service API Functions

I/O Communications Service API functions are used to temporarily stop/start I/O communications with Masters when using the Slave function.

I/O Data Access Service API Functions

These functions are the same as the I/O Data Access Service API functions for the Master function, but are used with the Slave function.

Explicit Message API Functions

Message Monitoring Timer Service API Functions

The message monitoring timer monitors the time from when a explicit message request is sent until an explicit message response is received from the remote node. If the response is not be received within the response time, the request is timed out. The Message Monitoring Timer Service API functions are used to set the message monitoring timer for each remote device.

The default message time is 2 s (2,000 ms). Set a longer time if the response from the remote nodes will take longer. The next request to the same device cannot be sent while waiting for a response.

Client Message Service API Functions

Client Message Service API functions are used to sent explicit request messages and receive responses, to set parameters for any device, and to monitor information for that device.

When the Scanner SDK receives a response for the sent request, it automatically stores it in the response queue in the DLL file. There is a response queue for each remote device and the response is queued for as long as there is free memory on the computer. The response should be removed from the queue as soon as possible after it is received.

Noise or other interference may cause the explicit message to be lost. Also, the message may not be received, depending on the remote device. Always enable retries when sending explicit requests.

There are two methods for detecting reception of explicit response messages. Refer to *3-3 Checking Events* for information on detection methods.

Server Message Service API Functions

Server Message Service API functions receive explicit request messages and send responses to implement any device profile.

When Scanner SDK receives a request addressed to a registered object, that request is automatically stored in the response queue in the DLL file. There is a response queue for each registered object and the response is queued for as long as there is free memory on the computer. The response should be removed from the queue as soon as possible after it is received.

There are two methods for detecting reception of explicit request messages. Refer to *3-3 Checking Events* for information on detection methods.

Maintenance API Functions

Status Service API Functions

Status Service API functions are used to monitor errors during I/O or explicit message communications and monitor the communications cycle time for the Master function. Monitor error status when creating user applications and perform error processing as required.

Error Log Access Service API Functions

Error Log Access Service API functions read and clear the error log registered in the Board. Refer to *3-10 Error Log Functions* for details.

PC Watchdog Timer Service API Functions

PC Watchdog Timer Service API functions monitor operation of user applications by the Board. Refer to *3-11 PC Watchdog Timer Management Function* for details.

3-3 Checking Events

3-3-1 Board Events

The following table shows the kinds of events that occur in the Scanner SDK.

Event	Description
Interrupt from Board	This event occurs when an interrupt is sent from the Board to the computer. (An interrupt control register must be set.)
Explicit client response received	This event occurs when a response is received from the server after an explicit client request message is sent.
Explicit server request received	This event occurs when an explicit server request message is received for a registered object.

3-3-2 Checking for Events

There are two ways to check events in the Board. Use the method that is easiest for your application and system conditions.

- Detecting events with Windows messaging
- Checking events by polling

Detecting Events with Windows Messaging

A Windows message is sent automatically when an event occurs. A thread or window can be specified as the destination for the event notification. The notification can include event specific information such as the interrupt status or reception data length.

When the user application receives the event notification, it will read (receive) the event after preparing a data buffer to hold the event data.

When you use Windows messaging for event notification, the notification message settings must be made in advance with the
SCAN_RegIrqEvtNotifyMessage(), SCAN_RegClientEvtNotifyMessage(),
and SCAN_RegServerEvtNotifyMessage() functions.

Once event notification has been received by the user application, execute the corresponding event processing. For example, when an explicit client response is received, prepare a data buffer to hold the received service data and read the response.

Checking Events by Polling

The event queue can be checked from a user application with the SCAN_PeekIrqEvent(), SCAN_PeekClientEvent(), and SCAN_PeekServerEvent() functions.

If the check shows an event in the event queue, execute the corresponding event processing. For example, for an explicit client response event, check the received data size using the SCAN_ClientEventLength() function, prepare a data buffer to hold the received service data, and read the response.

3-4 Checking for Errors

Checking for Errors with Function Return Values

The value returned by the function shows whether an error occurred. The Scanner SDK's API functions are all bool type functions and the result of the function's execution is returned as one of the following boolean values:

FALSE: The function was completed with an error.
TRUE: The function was completed normally.

If an error occurred, detailed error information can be obtained with the GetLastError() function. Refer to *7-2 Identifying Errors Detected by Functions* for the meaning of the error codes obtained with the GetLastError() function and appropriate error processing.

Note GetLastError is a Windows API function. Detailed error information can be obtained by calling GetLastError immediately after executing the DeviceNet PCI Board API function, as shown below.

Example: Using GetLastError() Function with Board Open API Call

```
DWORD dwErrCode;
//DeviceNet PCI Board Open API function call
if(SCAN_Open(DeviceNo,Handle)==false){
    //Gets the detailed error code
    //when the function returns an error.
    dwErrCode=GetLastError();
}
```

3-5 Parameters

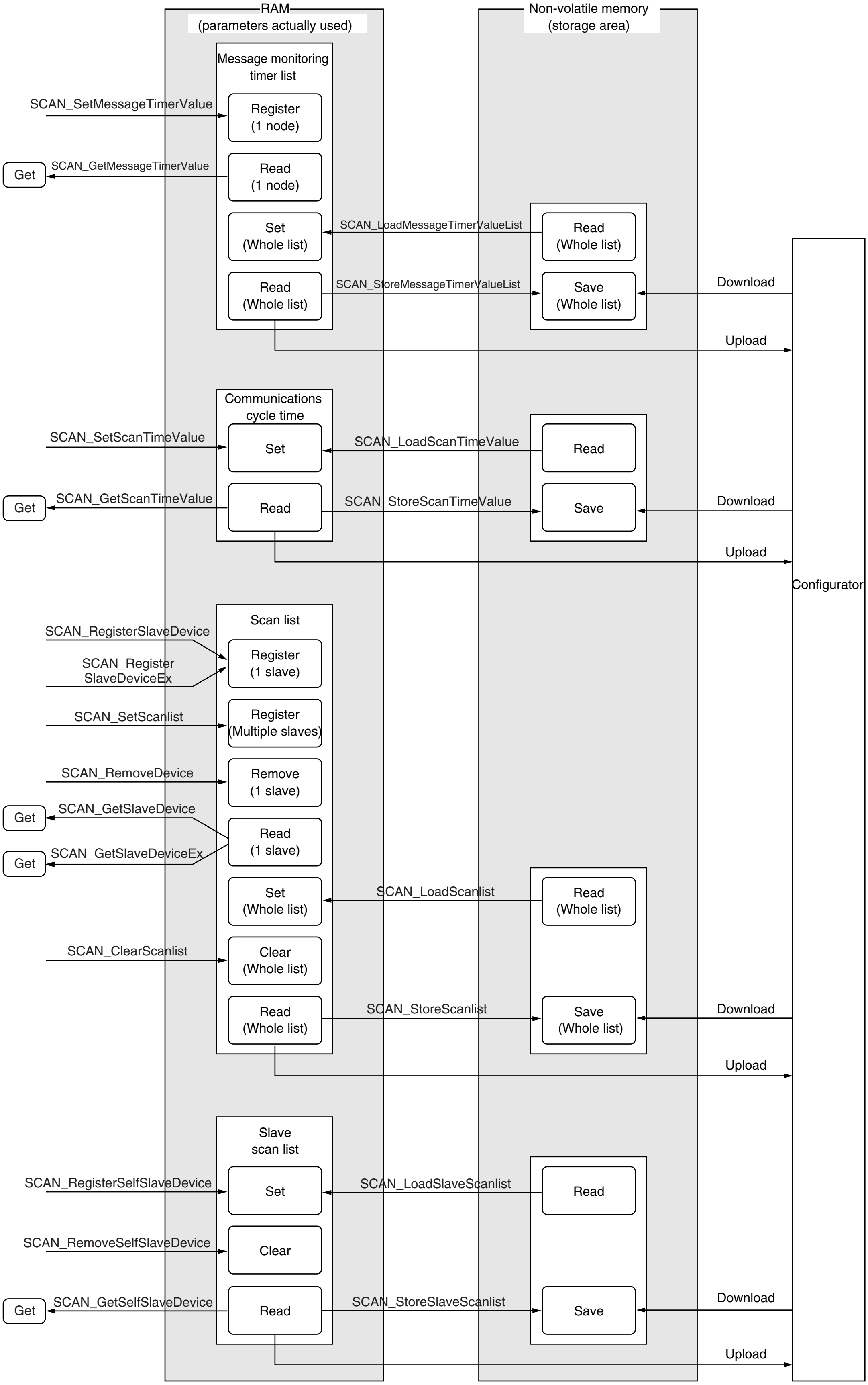
Parameter Types

The PCI Board has the parameters listed in the following table.

Name	Meaning	Initial Value
Message Monitoring Timer List	A list of message monitoring timers for all node addresses. The monitoring time is from when the explicit request message is sent until the response is received.	0 (2 s) is set for all node addresses as the default.
Communications Cycle Time	Used with the Master function. The communications cycle time is the cycle for I/O communications.	0 (automatic) is set as the default.

Name	Meaning	Initial Value
(Master) Scan List	Used with the Master function. The scan list is the list of slave node addresses for I/O communications. The list also registers the parameters required for I/O communications for each node.	The default is no scan list. API functions are required to register a scan list.
Slave Scan List	Used with the Slave function. The slave scan list registers the parameters required for nodes to operate as a slave.	The default is no slave scan list. API functions are required to register a slave scan list.

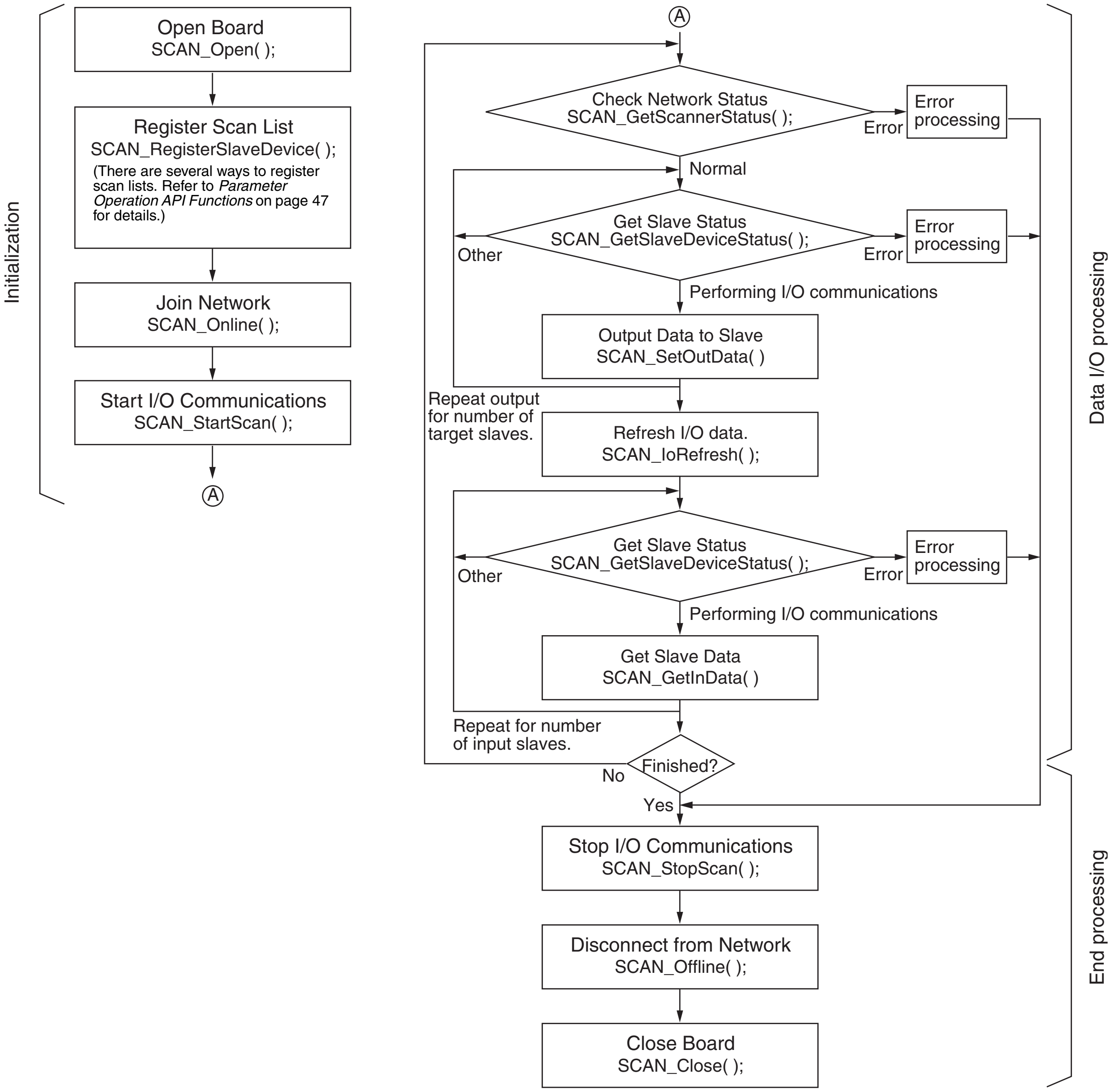
Parameter Operation API Functions



3-6 Using I/O Communications Functions

Procedure for Using Master Function

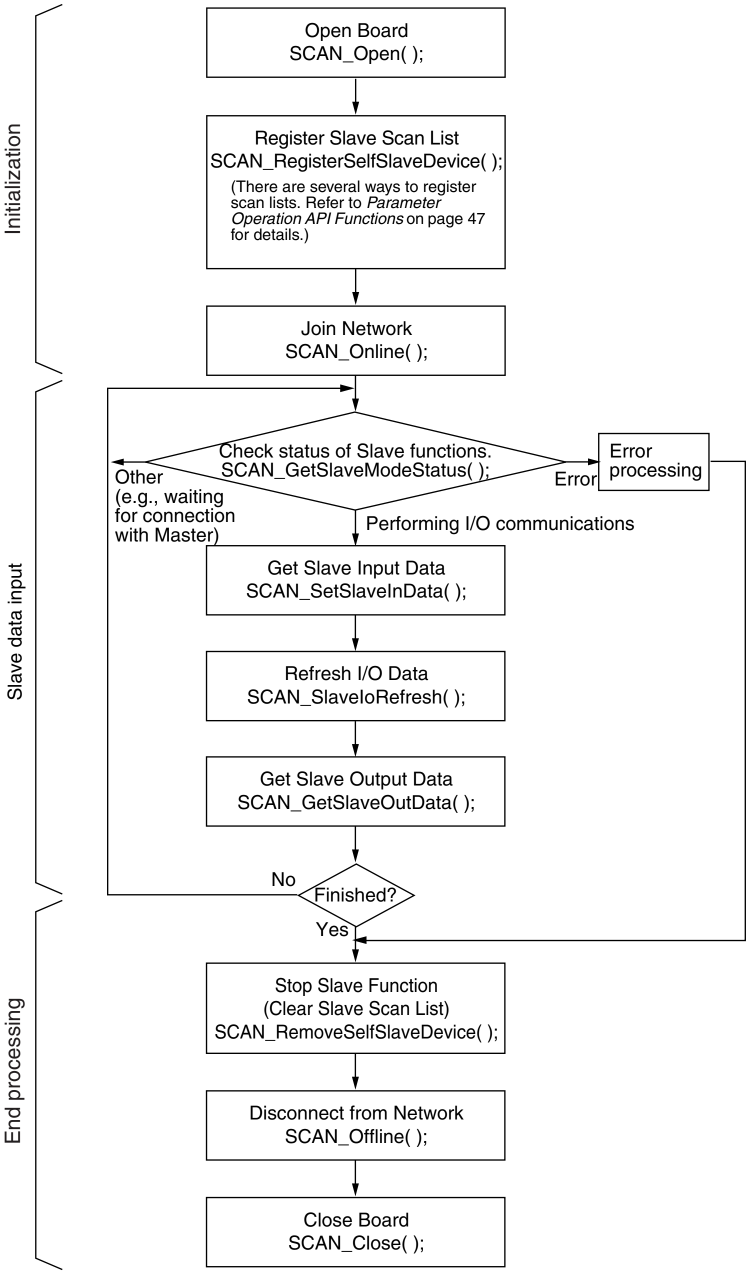
Use the procedure shown in the following diagram to use API functions when using the Master function of the Scanner SDK. Refer to *SECTION 4 API Function Reference* for details on using API functions.



Note Several seconds are required after SCAN_StartScan() is called until I/O communications actually start.

Procedure for Using Slave Function

Use the procedure shown in the following diagram to use API functions when using the Slave function of the Scanner SDK. Refer to *SECTION 4 API Function Reference* for details on how to use API functions.

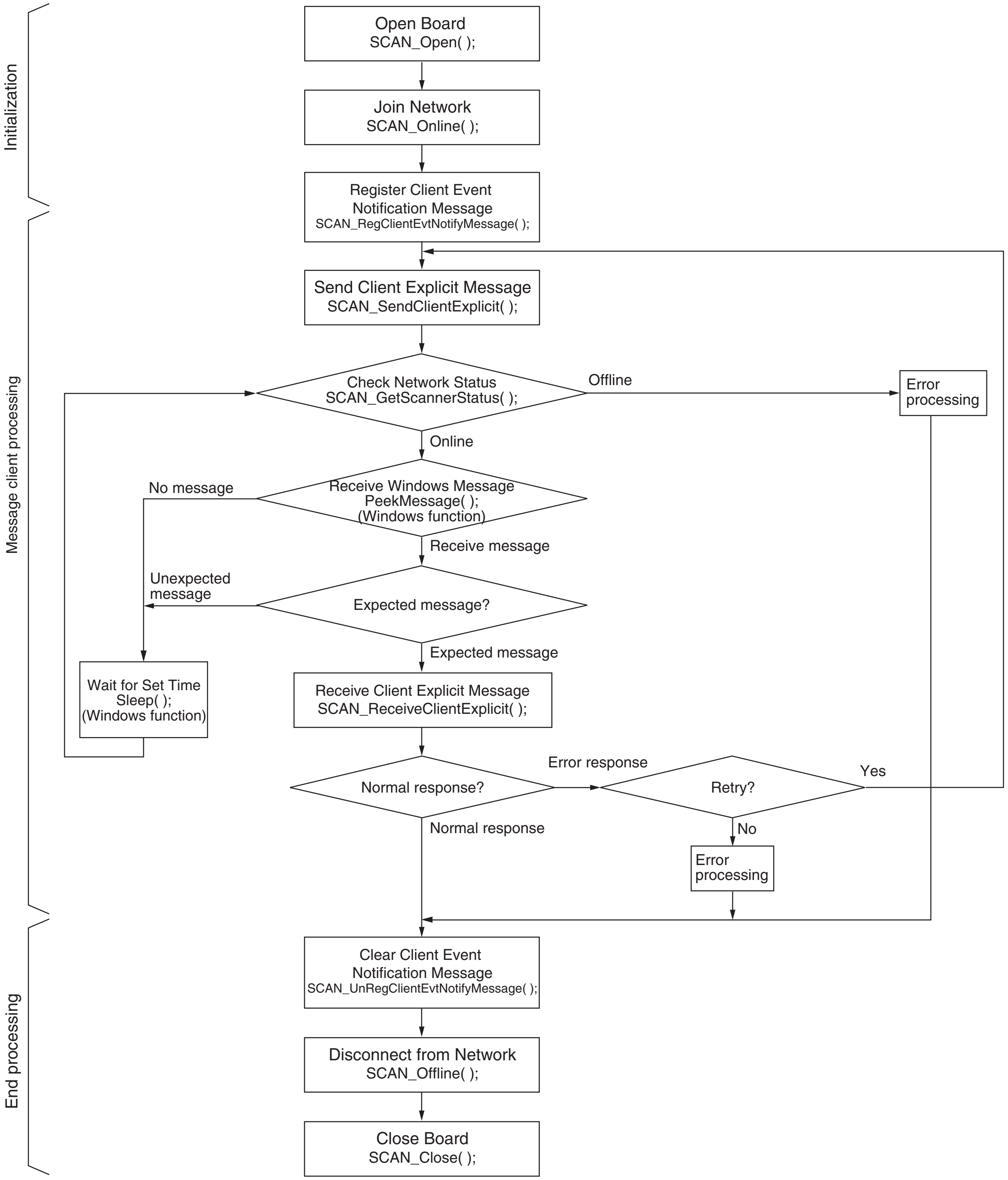


3-7 Using the Explicit Message Client Function

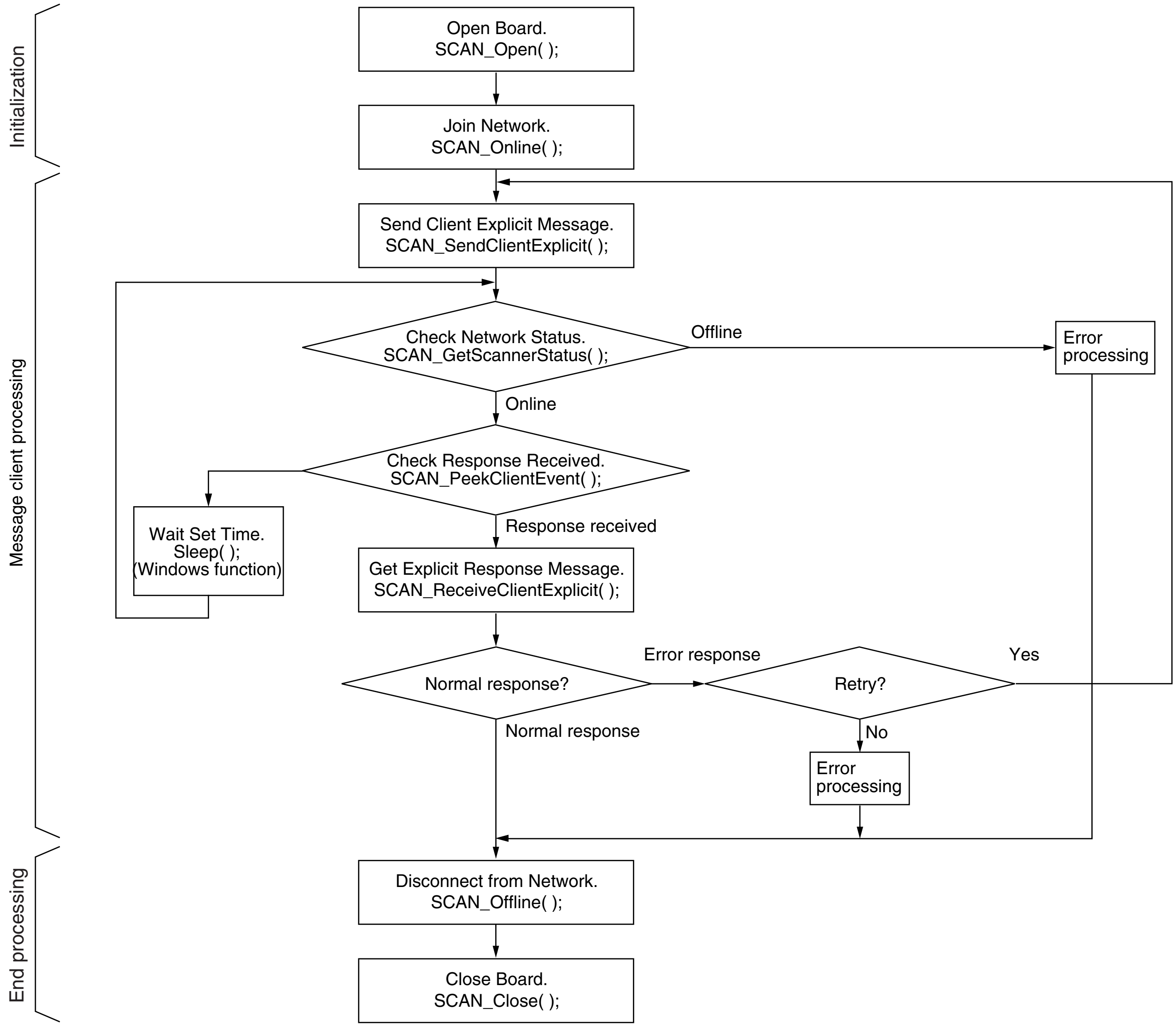
There are two ways to detect response reception when using the explicit message client function of the Scanner SDK. Each method uses API functions in the procedures outlined in the following diagrams. Refer to *SECTION 4 API Function Reference* for details.

Note Always execute retries with explicit message communications. Sometimes messages cannot be received due to the status of the remote node or an error response is returned.

Using Windows Messages for Event Notification



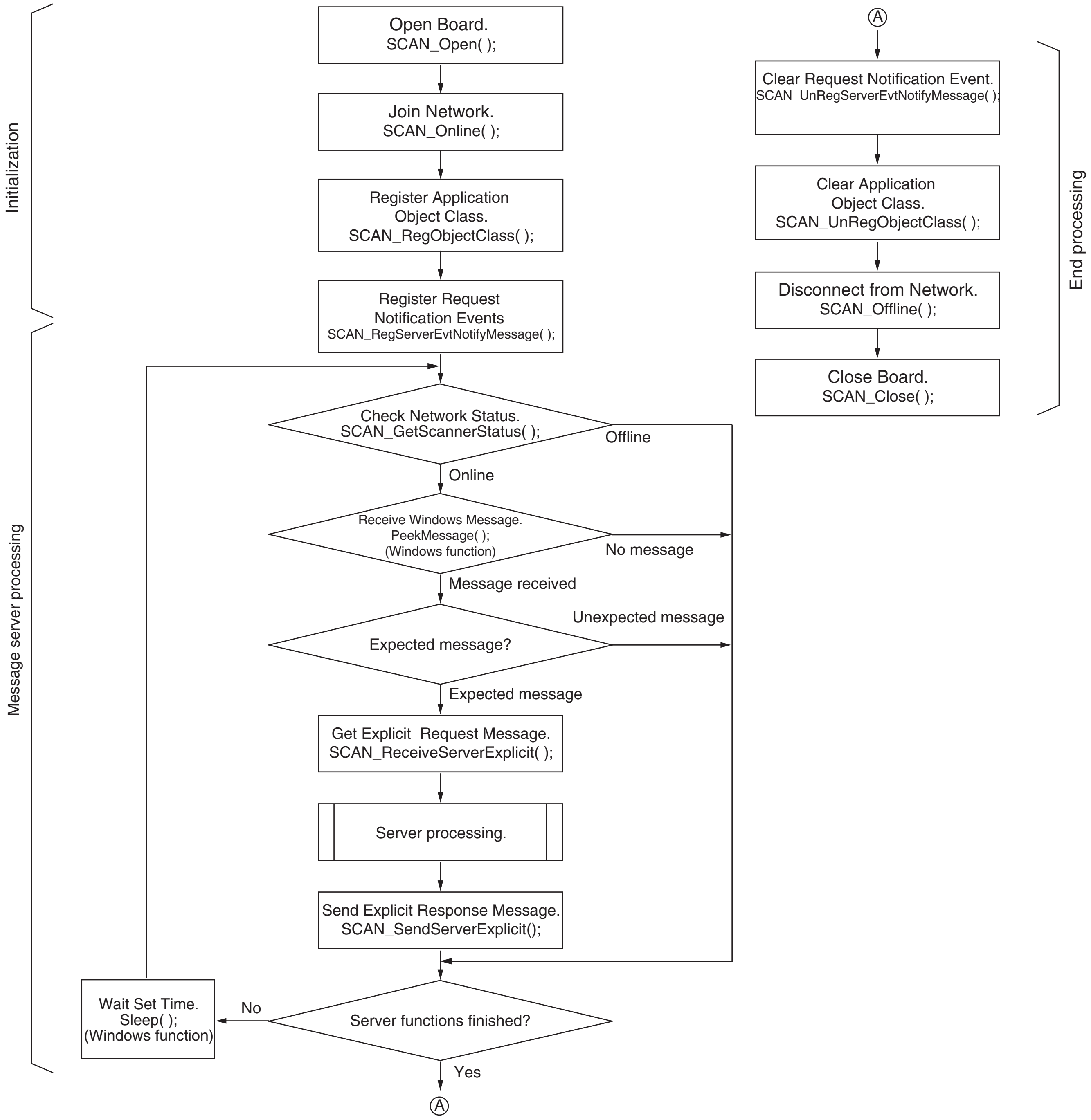
Checking Events by Polling the Event Queue



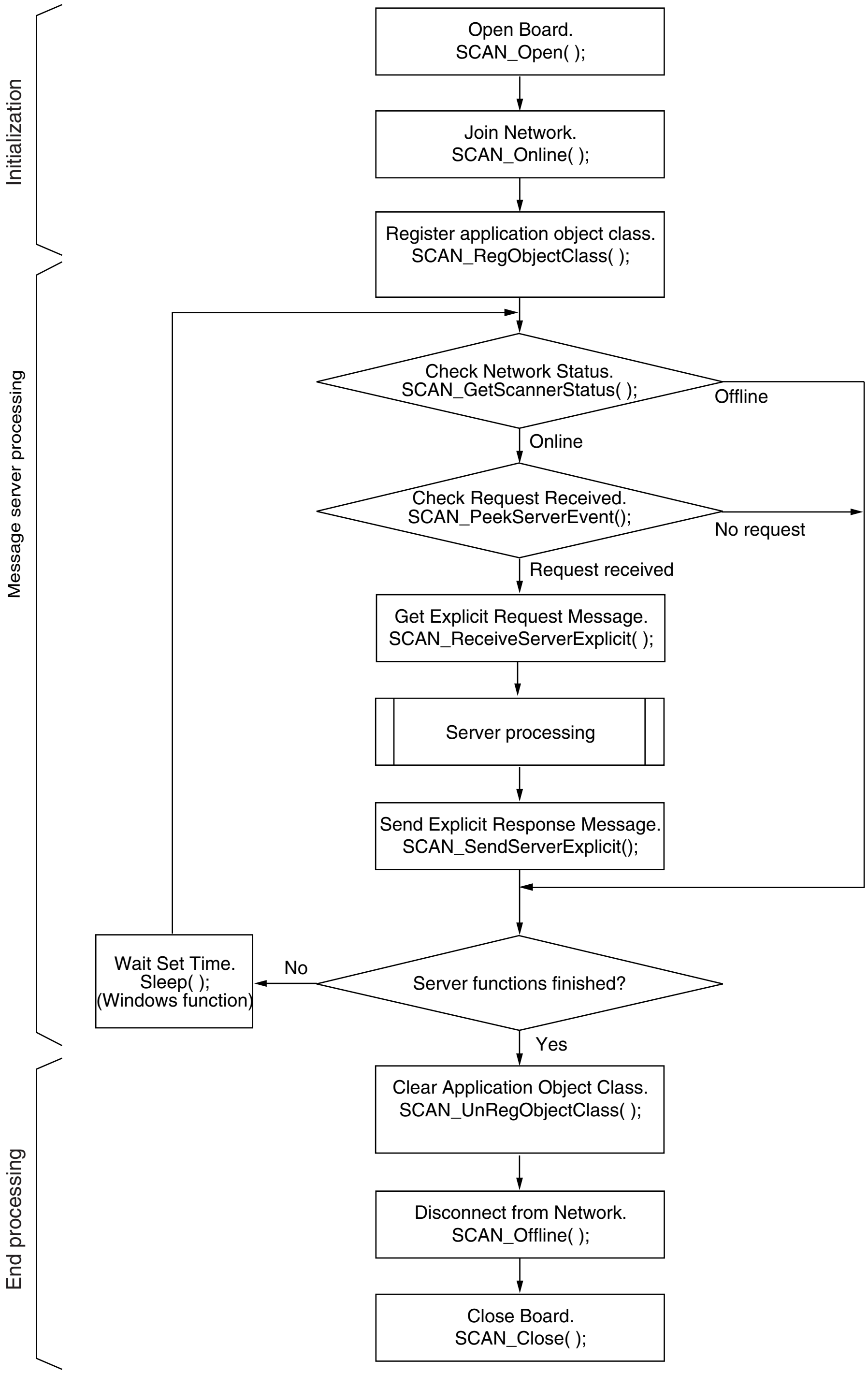
3-8 Using the Explicit Message Server Function

There are two methods for detecting request messages when using the explicit message server function with Scanner SDK. Each method uses API functions in the procedures shown in the following diagram. Refer to *SECTION 4 API Function Reference* for details.

Using Windows Messages for Event Notification



Checking Events by Polling the Event Queue



3-9 Reset Function

Use the SCAN_Reset() function to reset the Board and initialize the Board to the same status it had at startup. When the Board is reset, all of the resources set aside for the Board will be cleared and all of the information set from user applications will be lost. All required information must be set again after executing SCAN_Reset().

Note The Board reset will be completed when the TRUE return value is returned from SCAN_Reset(). The next function can be executed immediately after the TRUE return value is recognized.

3-10 Error Log Functions

The PCI Board provides an error log function that records and holds error information. When an error occurs, one record per error is stored in the Board's internal RAM error log table, up to a maximum of 64 records. When the maximum number of 64 records has been stored in the error log table, the oldest record will be discarded when another error occurs and the new error data will be recorded in the table.

The following information is recorded in the error log table.

- Error code (Refer to *Error Log Data* on page 126.)
- Details code (Refer to *Error Log Data* on page 126.)
- Data and time of error (The computer time and data information is used.)

Saving Error Logs

When an error is detected, the error log and the date and time of the error are recorded in the Board's internal RAM.

Serious errors are also recorded in EEPROM. (Refer to *Error Code Table* on page 126.) Error logs stored in EEPROM are held even if the computer power is turned OFF or the Board is reset. When the Board is started, the error log recorded in EEPROM is copied to RAM.

API functions can be used to read the error logs held in RAM. When error logs are cleared, the error logs held in both RAM and EEPROM are cleared.

Reading and Clearing Error Logs

The functions listed in the following table are used to read and clear error logs stored in the Board.

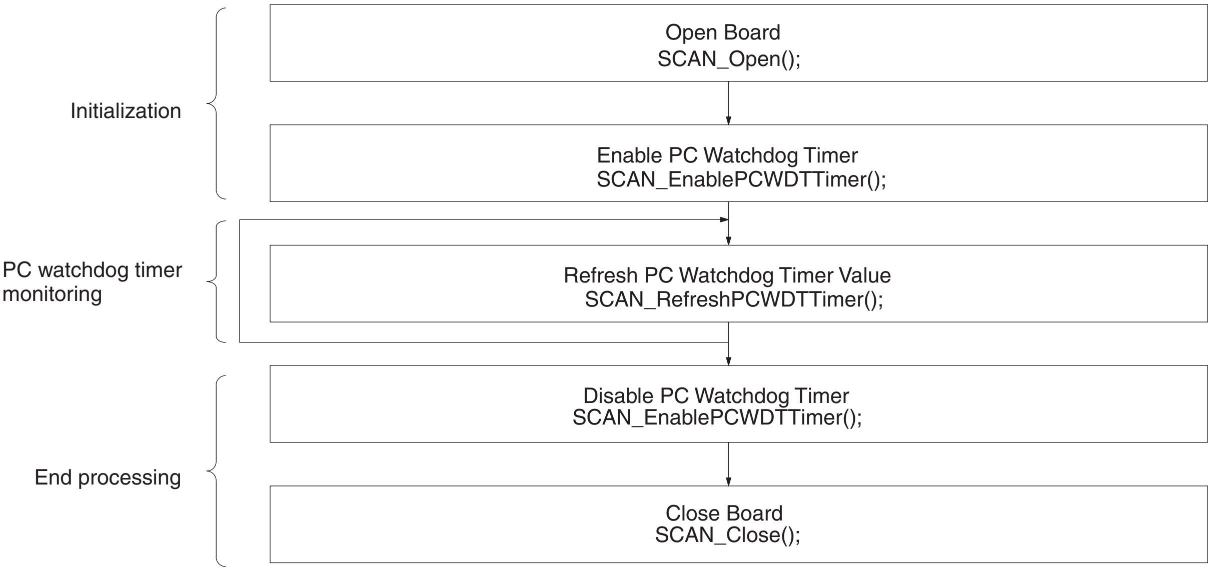
Function	Name	Operation	Page
SCAN_GetErrorLog()	Read Error Log	Reads all of the error records stored in the Board.	101
SCAN_ClearErrorLog()	Clear Error Log	Clears all of the error records stored in the Board.	102

3-11 PC Watchdog Timer Management Function

The DeviceNet PCI Board is equipped with a PC watchdog timer function that can stop remote I/O communications automatically if the application that controls the Board stops for some reason. While the PC watchdog timer function is running, remote I/O communications will be stopped automatically if the PC watchdog timer value is not refreshed from the user application within the set timeout detection period.

The Board's PC watchdog timer is refreshed regularly from the computer (application) to notify the Board that the application is operating normally. The timeout detection period is 2 × (the monitoring time + 10 ms).

Use the API functions as outlined in the following flowchart when using the PC watchdog timer function.



Note The Board's PC watchdog timer function is disabled when the Board starts operation. It is also disabled by the SCAN_Reset() function.

When using the PC watchdog timer function, execute the SCAN_Enable PCWDTTimer() to enable the PC watchdog timer and refresh the timer value. If the timer isn't refreshed, the function will determine that the application has stopped and will stop remote I/O communications if the set time has elapsed since the last refresh.

SECTION 4
API Function Reference

This section provides details on the various API functions in the DN 3G8F7 Scanner.DLL that are used with the DeviceNet PCI Board.

4-1 Function Lists

Board Management API Functions

Board Service API Functions

Use the following API functions for initialization and end processes such as reading the DLL version, reading the driver version, opening the Board, or closing the Board.

API function	Operation
SCAN_GetVersion	Reads the Scanner Software's DLL version.
SCAN_GetDriverVersion	Reads the Board's driver version.
SCAN_IsExistCard	Checks whether a Board is installed.
SCAN_Open	Opens the Board.
SCAN_Close	Closes the Board.
SCAN_Online	Adds the Board to the network (online).
SCAN_Offline	Removes the Board from the network (offline).
SCAN_Reset	Resets the Board.

PC Board Interrupt Service API Functions

Use the following API functions to set interrupts from the Board to the PC, set the interrupt notification message, and clear the interrupt notification message.

API function	Operation
SCAN_GetIrqControl	Reads the PC interrupt control register.
SCAN_SetIrqControl	Sets the PC interrupt control register.
SCAN_RegIrqEvtNotifyMessage	Registers the interrupt notification message.
SCAN_UnRegIrqEvtNotifyMessage	Clears the interrupt notification message.
SCAN_PeekIrqEvent	Checks the PC interrupt status
SCAN_ClearIrqEvent	Clears the PC interrupt status.

Master Function API Functions

Scan List Operations API Functions

Use the following API functions for scan list operations when the Master function is operating.

API function	Operation
SCAN_RemoveDevice	Removes a Slave from the scan list.
SCAN_StoreScanlist	Saves the scan list to non-volatile memory.
SCAN_LoadScanlist	Loads the scan list from non-volatile memory.
SCAN_SetScanlist	Registers multiple slaves to the scan list.
SCAN_ClearScanlist	Deletes the scan list.
SCAN_RegisterSlaveDevice	Registers a Slave in the scan list.
SCAN_GetSlaveDevice	Reads Slave information from the scan list.
SCAN_RegisterSlaveDeviceEx	Registers a Slave in the scan list (detailed information.)
SCAN_GetSlaveDeviceEx	Reads Slave information from the scan list (detailed information.)

I/O Communications Service API Functions

Use the following API functions for operations such as starting remote I/O communications (Master function), stopping remote I/O communications (Master function), or establishing a connection.

API function	Operation
SCAN_StartScan	Starts remote I/O.
SCAN_StopScan	Stops remote I/O.

API function	Operation
SCAN_GetActualSlaveDevice	Reads existing Slave information.
SCAN_ConnectSlaveDevice	Starts I/O communications with a specified Slave.
SCAN_DisconnectSlaveDevice	Stops I/O communications with a specified Slave.
SCAN_SetScanTimeValue	Sets the communications cycle time.
SCAN_GetScanTimeValue	Reads the communications cycle time.
SCAN_StoreScanTimeValue	Writes the communications cycle time to non-volatile memory.
SCAN_LoadScanTimeValue	Loads the communications cycle time from non-volatile memory.

I/O Data Access Service API Functions

Use the following API functions to refresh, set, and read I/O data when using the Master function.

API function	Operation
SCAN_IoRefresh	Executes I/O refreshing.
SCAN_GetInData	Reads Slave input data.
SCAN_SetOutData	Sets Slave output data.
SCAN_SendMasterCosToSlave	Immediately sends output data to Slaves from the Master performing COS communications.

Slave Function API Functions

Slave Scan List Operations API Functions

Use the following API functions for scan list operations when the Slave function is operating.

API function	Operation
SCAN_RegisterSelfSlaveDevice	Registers the Slave scan list.
SCAN_RemoveSelfSlaveDevice	Deletes the Slave scan list.
SCAN_GetSelfSlaveDevice	Reads the Slave scan list information.
SCAN_StoreSlaveScanlist	Saves the Slave scan list to non-volatile memory.
SCAN_LoadSlaveScanlist	Loads the Slave scan list from non-volatile memory.

I/O Communications Service API Functions

Use the following API functions for operations such as starting remote I/O communications (Slave function), stopping remote I/O communications (Slave function), or establishing a connection.

API function	Operation
SCAN_ConnectMasterDevice	Starts I/O communications with a Master.
SCAN_DisconnectMasterDevice	Stops I/O communications with a Master.

I/O Data Access Service API Functions

Use the following API functions to refresh, set, and read I/O data.

API function	Operation
SCAN_SlaveIoRefresh	Executes Slave I/O refreshing.
SCAN_GetSlaveOutData	Reads Master output data.
SCAN_SetSlaveInData	Sets Master input data.
SCAN_SendSlaveCosToMaster	Immediately sends input data to the Master from Slaves performing COS communications.

Explicit Message API Functions

Message Monitoring Timer Service API Functions

Use the following API functions to manage the explicit message monitoring timer.

API function	Operation
SCAN_SetMessageTimerValue	Sets the message monitoring timer.
SCAN_GetMessageTimerValue	Reads the message monitoring timer.
SCAN_StoreMessageTimerValueList	Saves the message monitoring timer list to non-volatile memory.
SCAN_LoadMessageTimerValueList	Loads the message monitoring timer list from non-volatile memory.

Client Message Service API Functions

Use the following API functions when the Board is operating as an explicit message client.

API function	Operation
SCAN_RegClientEvtNotifyMessage	Registers the client response event notification message.
SCAN_UnRegClientEvtNotifyMessage	Clears the client response event notification message.
SCAN_SendClientExplicit	Sends an explicit client message.
SCAN_ReceiveClientExplicit	Receives an explicit client message.
SCAN_PeekClientEvent	Checks the client response event.
SCAN_GetClientEventLength	Reads the size of the client response.

Server Message Service API Functions

Use the following API functions when the Board is operating as an explicit message server.

API function	Operation
SCAN_RegObjectClass	Registers the object class ID.
SCAN_UnRegObjectClass	Clears the object class ID.
SCAN_RegServerEvtNotifyMessage	Sets the server request event notification message.
SCAN_UnRegServerEvtNotifyMessage	Clears the server request event notification message.
SCAN_SendServerExplicit	Sends an explicit server message.
SCAN_ReceiveServerExplicit	Gets an explicit server message.
SCAN_PeekServerEvent	Checks the server request event.
SCAN_GetServerEventLength	Reads the size of the server request.

Maintenance API Functions

Status Service API Functions

Use the following API functions to read various kinds of status information.

API function	Operation
SCAN_GetNetworkStatus	Reads the network status.
SCAN_GetScannerStatus	Reads the scanner status.
SCAN_GetMasterModeStatus	Reads the Master function status.
SCAN_GetSlaveModeStatus	Reads the Slave function status.
SCAN_IsScanlistSlaveDeviceRegist	Checks if a Slave is registered in the scan list.
SCAN_IsDeviceConnection	Checks that the connection with a Slave is open.
SCAN_GetSlaveDeviceStatus	Reads the Slave’s status.

API function	Operation
SCAN_GetCycleTime	Reads the present value of the communications cycle time.
SCAN_GetMaxCycleTime	Reads the maximum value of the communications cycle time.
SCAN_GetMinCycleTime	Reads the minimum value of the communications cycle time.
SCAN_ClearCycleTime	Clears the maximum and minimum communications cycle times.

Error Log Access Service API Functions

Use the following API functions to read and clear the error log.

API function	Operation
SCAN_GetErrorLog	Reads the error log.
SCAN_ClearErrorLog	Clears the error log.

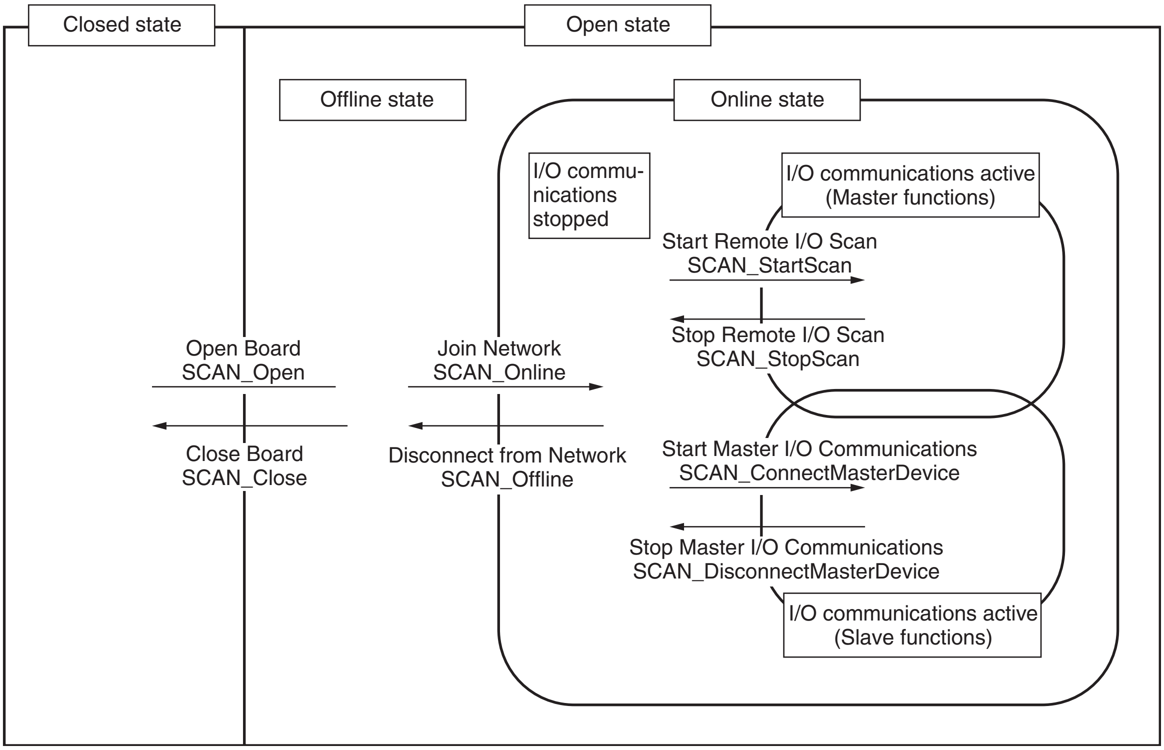
PC Watchdog Timer Service API Functions

Use the following API functions to control the PC watchdog timer function.

API function	Operation
SCAN_EnablePCWDTTimer	Enables or disables the PC watchdog timer.
SCAN_RefreshPCWDTTimer	Refreshes the PC watchdog timer.

4-2 Board Status

The following diagram shows how the Board status is changed by API functions. There are restrictions on what API functions can be used, depending on the Board status. Refer to *Application Range* under each function description for details.


Closed state
Open state
Offline state
Online state
I/O communications stopped
I/O communications active (Master functions)
Start Remote I/O Scan SCAN_StartScan
Stop Remote I/O Scan SCAN_StopScan
Open Board SCAN_Open
Close Board SCAN_Close
Join Network SCAN_Online
Disconnect from Network SCAN_Offline
Start Master I/O Communications SCAN_ConnectMasterDevice
Stop Master I/O Communications SCAN_DisconnectMasterDevice
I/O communications active (Slave functions)

Closed State

All of the Board's resources are disengaged in this status. Control operations cannot be performed on the Board in this status.

The Board will be in this status after it is started or reset.

Open State

All of the Board's resources are engaged in this status. Control operations can be performed on the Board in this status.

The Board's "device handle" is determined when the Board is switched to open status by the SCAN_Open() function. Specify the Board with the device handle to use the API functions while the Board is open.

Offline State

The Board is open but is not participating in the DeviceNet network.

Online State

The Board is open and participating in the DeviceNet network.

Remote I/O Stopped State

The Board is open and participating in the DeviceNet network but remote I/O communications are stopped.

Remote I/O Active State

The Board is open, participating in the DeviceNet network, and performing remote I/O communications.

4-3 Board Management API Functions

4-3-1 Board Service API Functions

Reading the DLL Version: SCAN_GetVersion()

Application Range

Unlimited (Can be executed in closed status.)

Function

Reads the version information for the API (DN 3G8F7 Scanner.DLL) that is being used.

Call Format

BOOL SCAN_GetVersion(DWORD **Version*)

Arguments

Type	Name	Contents
DWORD*	Version	Buffer address for obtaining version information.

Return Value

TRUE is returned when the function is completed normally and FALSE is returned when an error occurs during processing. Detailed error information can be read with the GetLastError() function.

Description

Use this function when the DLL version needs to be checked.

The DLL version is stored in BCD in the following format:

31 16	15 0
Major version	Minor version

For example, DLL version 1.13 will be represented as 0x00010013.

Reading the Driver Version: SCAN_GetDriverVersion()

Application Range

Open status

Function

Reads the version information for the driver being used for the DeviceNet PCI Board.

Call Format

BOOL SCAN_GetDriverVersion(DWORD *Handle,* DWORD **Version*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
DWORD*	Version	Buffer address for obtaining version information.

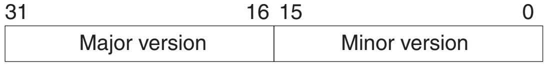
Return Value

TRUE is returned when the function is completed normally and FALSE is returned when an error occurs during processing. Detailed error information can be read with the GetLastError() function.

Description

Use this function when the driver version needs to be checked.

The driver version is stored in BCD in the following format:



For example, DLL version 1.13 will be represented as 0x00010013.

Checking the Board: SCAN_IsExistCard()

Application Range

Unlimited (Can be executed in closed status.)

Function

Checks whether or not there is a DeviceNet PCI Board with the specified board ID.

Call Format

BOOL SCAN_IsExistCard(DWORD *BoardId*)

Arguments

Type	Name	Contents
DWORD	BoardId	Board ID of the Board that you want to check Setting range: 0x0 to 0x7 (0 to 7)

Return Value

TRUE is returned if there is a DeviceNet PCI Board with the specified board ID and FALSE is returned if there is not. Detailed error information can be read with the GetLastError() function.

Description

Use to check the ID of the mounted Board.

Set the board ID with the Board's rotary switch.

Opening the Board: SCAN_Open()

Application Range

Closed status

Function

Opens the DeviceNet PCI Board with the specified board ID and makes it usable.

Call Format

BOOL SCAN_Open(DWORD BoardId, DWORD **Handle*)

Arguments

Type	Name	Contents
DWORD	BoardId	Board ID of the Board that you want to open Setting range: 0x0 to 0x7 (0 to 7)
DWORD*	Handle	Buffer address for obtaining device handle.

Return Value

TRUE is returned if the DeviceNet PCI Board with the specified board ID was opened successfully. FALSE is returned if an error occurred or there is not a Board with the specified board ID. Detailed error information can be read with the GetLastError() function.

Description

The Board must be open before it can be used.

Each Board can be opened from one application (process) only.

The Board ID is the value set using the rotary switch on the Board.

Closing the Board: SCAN_Close()

Application Range

Open status

Function

Closes the specified device handle and makes the Board unusable.

Call Format

BOOL SCAN_Close(DWORD *Handle*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()

Return Value

TRUE is returned if the specified device handle was closed successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

When this function is executed, the device handle is released, the Board is reset, and all set information is cleared. Perform any end processing for the application before executing this function.

Joining the Network (Online): SCAN_Online()

Application Range

Offline status

Function

Adds the specified DeviceNet PCI Board to the network and puts it in online status.

Call Format

BOOL SCAN_Online(DWORD *Handle*, WORD *MacId*, WORD *BaudRate*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	MacId	Node address setting for the Board Setting range: 0x00 to 0x3F (0 to 63)
WORD	BaudRate	Communications speed setting for the Board Setting range: ONLINE125K(0): 125 kbps ONLINE250K(1): 250 kbps ONLINE500K(2): 500 kbps

Return Value

TRUE is returned if the specified Board was successfully switched to online status. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

It takes approximately 3 seconds for online processing to be completed.

FALSE is returned and the Board is not switched to online status in the following cases. Use GetLastError() to find the source of the error, clear the error, and execute the function again.

- Bus OFF error
- Node address duplication
- Network power supply error

- Send timeout

A network error may occur if the other nodes in the network have a different baud rate setting.

When connecting to slaves with an automatic baud rate recognition function, a transmission timeout may occur. If this error occurs, perform SCAN_Online() retry processing.

Leaving the Network (Offline): SCAN_Offline()

Application Range

Online status

Function

Removes the specified DeviceNet PCI Board from the network and puts it in offline status.

Call Format

BOOL SCAN_Offline(DWORD *Handle*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()

Return Value

TRUE is returned if the specified Board was successfully switched to offline status. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Resetting the Board: SCAN_Reset()

Application Range

Open status

Function

Resets the specified DeviceNet PCI Board.

Call Format

BOOL SCAN_Reset(DWORD *Handle*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()

Return Value

TRUE is returned if the processing was completed properly. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

When this function is executed the Board will be reset and all of the information that has been set will be cleared. It will be necessary to redo all of the procedures that were executed since the Board was opened.

The TRUE return value is returned when initialization of the Board is completed, so other functions can be executed immediately after TRUE is returned.

If remote I/O functions are being used and the Board is communicating with other nodes as a client, a communications timeout will occur at the remote node and the NS indicator will flash red when the Board is reset.

4-3-2 PC Board Interrupt Service API Functions

Reading the Interrupt Control Register: SCAN_GetIrqControl()

Application Range

Open status

Function

Reads the value of the interrupt control register that determines whether or not the computer is notified of interrupts that occur in the specified DeviceNet PCI Board.

Call Format

BOOL SCAN_GetIrqControl(DWORD *Handle*, BYTE **IrqReg*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
BYTE*	IrqReg	Buffer address for obtaining interrupt control register.

Return Value

TRUE is returned if the register value was read from the specified Board successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

Use this function to check the value set in the interrupt control register.

The interrupt control register value is stored in IrqReg with the following format:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit name	Reserved						BD_ WDT	SCAN

BD_WDT: Board watchdog timer interrupt
SCAN: One-scan completed interrupt

Writing the Interrupt Control Register: SCAN_SetIrqControl()

Application Range

Open status

Function

Sets the value of the interrupt control register that determines whether or not the computer is notified of interrupts that occur in the specified DeviceNet PCI Board.

Call Format

BOOL SCAN_SetIrqControl(DWORD *Handle*, BYTE *IrqReg*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
BYTE	IrqReg	New register value setting (The data format is the same as it is in SCAN_GetIrqControl().)

Return Value

TRUE is returned if the register value was written to the specified Board successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

Once the interrupt control register has been set, the computer will be notified when the corresponding interrupts occur.

Registering an Interrupt Notification Message: SCAN_RegIrqEvtNotifyMessage()

Application Range

Open status

Function

Registers the Windows message that notifies that an interrupt from the Board occurred.

Call Format

BOOL SCAN_RegIrqEvtNotifyMessage(DWORD *Handle*, DWORD *ThreadId*, HWND *hWnd*, UNIT *Msg*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
DWORD	ThreadId	The thread ID to notify. (No setting = NULL)
HWND	hWnd	Specifies the window handle to notify. (No setting = NULL)
UNIT	Msg	Notification message Range: WM_USER + 0x100 to WM_USER + 0x7FFF

Return Value TRUE is returned if registration of the notification message was completed successfully. FALSE is returned if an error occurred such as null values for both the thread ID and window handle. Detailed error information can be read with the GetLastError() function.

Description Specifies the thread ID or window handle for the event notification.

The interrupt status and notification message are sent to WPARAM and the notification message is sent to LPARAM.

Clearing an Interrupt Notification Message: SCAN_UnRegIrqEvtNotifyMessage()

Application Range Open status

Function Clears the notification message.

Call Format BOOL SCAN_UnRegIrqEvtNotifyMessage(DWORD *Handle*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()

Return Value TRUE is returned if the registered notification message was cleared successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Reading Interrupt Event: SCAN_PeekIrqEvent()

Application Range Open status

Function Reads the cause of the interrupt that occurred in the specified Board.

Call Format BOOL SCAN_PeekIrqEvent(DWORD *Handle*, BYTE **IrqStatus*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
BYTE*	IrqStatus	Buffer address for obtaining the cause of the interrupt. (The data format is the same as it is in SCAN_GetIrqControl().)

Return Value TRUE is returned if the status value was read successfully from the specified Board. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description To check the interrupt status of a specified Board, use the SCAN_SetIrqControl() function and specify notification to the computer of the interrupt that occurred in the Board.

Clearing Interrupt Status: SCAN_ClearIrqEvent()

Application Range

Open status

Function

Clears the cause of the interrupt that occurred in the specified Board.

Call Format

BOOL SCAN_ClearIrqEvent(DWORD *Handle*, BYTE *IrqClrMask*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
BYTE	IrqClrMask	Interrupt mask to clear Clearing is specified when the corresponding interrupt bit position is ON. (The data format is the same as it is in SCAN_GetIrqControl().)

Return Value

TRUE is returned when the value was successfully written in the specified Board's register. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

Use this function to clear the cause of an interrupt after an interrupt has been detected by event notification or by SCAN_PeekIrqEvent() and the corresponding processing has been executed.

4-4 Master Function API Functions

4-4-1 Scan List Operation API Functions

Registering a Slave in the Scan List: SCAN_RegisterSlaveDevice()

Application Range

Open status

Function

Registers information in the scan list for a specified Slave.

Call Format

BOOL SCAN_RegisterSlaveDevice(DWORD *Handle*, WORD, *MacId*, WORD *Outsize*, WORD *Insize*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	MacId	Address to register in the scan list. Setting range: 0x00 to 0x3F (0 to 63)
WORD	Outsize	Output data size (bytes) Setting range: 0x00 to 0xC8 (0 to 200)
WORD	Insize	Input data size (bytes) Setting range: 0x00 to 0xC8 (0 to 200)

Return Value

TRUE is returned if the specified Slave was successfully registered in the scan list. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

I/O communications can be performed with Slaves registered in the scan list. Slaves do not need to be registered in the scan list for message communications unless they are involved in I/O communications.

Previously registered Slaves will be overwritten when re-registered.

The scan type for Slaves registered using this function will be automatically selected. Use SCAN_RegisterSlaveDeviceEx() to specify the scan type.

The I/O data for Slaves registered using this function will be output data 1 and input data 1.

Reading a Scan List Slave: SCAN_GetSlaveDevice()

Application Range Open status

Function Reads Slave information registered in the scan list.

Call Format BOOL SCAN_GetSlaveDevice(DWORD *Handle*, WORD, *MacId*, WORD* *Outsize*, WORD* *Insize*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	MacId	Node address for obtaining Slave information. Setting range: 0x00 to 0x3F (0 to 63)
WORD	Outsize	Buffer address for receiving output data size.
WORD	Insize	Buffer address for receiving input data size.

Return Value TRUE is returned if the specified Slave information was successfully read from the scan list. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description When this function is completed, the output data size and input data size (both in bytes, between 0 and 200) will be stored in the buffer at the specified addresses.

Removing a Slave from the Scan List: SCAN_RemoveDevice()

Application Range Open status

Function Deletes the specified Slave’s registration information from the scan list.

Call Format BOOL SCAN_RemoveDevice(DWORD *Handle*, WORD *MacId*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	MacId	Slave’s node address Setting range: 0x00 to 0x3F (0 to 63)

Return Value TRUE is returned if the specified Slave was successfully removed from the scan list. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Clearing the Scan List: SCAN_ClearScanlist()

Application Range Open status

Function Deletes the scan list as a group.

Call Format BOOL SCAN_ClearScanlist(DWORD *Handle*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()

Return Value TRUE is returned if the specified Board’s scan list was deleted successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Storing the Scan List: SCAN_StoreScanlist()

Application Range

Open status

Function

Saves the scan list information to non-volatile memory.

Call Format

BOOL SCAN_StoreScanlist(DWORD *Handle*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()

Return Value

TRUE is returned if the scan list was successfully written to non-volatile memory. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

The scan list information saved to non-volatile memory can be read using SCAN_LoadScanlist() and registered as the scan list.

Loading the Scan List: SCAN_LoadScanlist()

Application Range

Open status

Function

Loads the scan list information from non-volatile memory and registers it as the scan list.

Call Format

BOOL SCAN_LoadScanlist(DWORD *Handle*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()

Return Value

TRUE is returned if the scan list was successfully loaded from non-volatile memory. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

Use SCAN_StoreScanlist() to save the scan list information to non-volatile memory.

When scan list information is loaded, the previous scan list is overwritten and the newly read scan list is enabled.

SCAN_DEV Structure

This structure defines the format of the registration information used when registering a node (Master) in the scan list. This structure is used in the SCAN_RegisterSlaveDeviceEx() and SCAN_GetSlaveDeviceEx() functions. When nodes are registered in the scan list as a group, the FSCAN_DEV and FILE_SCAN_DEV structures are used.

Type	Name	Contents
WORD	VendorID	Vendor ID Scan list disabled mode:0xFFFF (65535) Specify the vendor ID to be verified when performing a vender ID verification check.
WORD	ProductType	Product type Scan list disabled mode:0xFFFF (65535) Specify the product type to be verified when performing a product type verification check.
WORD	ProductCode	Product code Scan list disabled mode:0xFFFF (65535) Specify the product code to be verified when performing a product code verification check

<table>
<tr><th>Type</th><th>Name</th><th>Contents</th></tr>
<tr><td>WORD</td><td>ScanType</td><td>Scan type
Specifies the type of connection to be used.
Up to two can be selected. However, COS and cyclic connections cannot be selected at the same time.
Specify 0×8000 (32768) for automatic selection.<table><tr><td>Bit 5</td><td>Bit 4</td><td>Bit 3</td><td>Bit 2</td><td>Bit 1</td><td>Bit 0</td></tr><tr><td>CY</td><td>COS</td><td>Reserved</td><td>BS</td><td>Poll</td><td>Reserved</td></tr></table>BS: Bit Strobe
CY: Cyclic</td></tr>
<tr><td>WORD</td><td>Output1</td><td>Size of output data 1 in bytes 0x00 to 0xC8 (0 to 200)</td></tr>
<tr><td>WORD</td><td>Input1</td><td>Size of input data 1 in bytes 0x00 to 0xC8 (0 to 200)</td></tr>
<tr><td>WORD</td><td>Output2</td><td>Size of output data 2 in bytes 0x00 to 0xC8 (0 to 200)
Set this area to 0 if specifying automatic selection for the scan type or specifying only one connection.</td></tr>
<tr><td>WORD</td><td>Input2</td><td>Size of input data 2 in bytes 0x00 to 0xC8 (0 to 200)
Set this area to 0 if specifying automatic selection for the scan type or specifying only one connection.</td></tr>
<tr><td>WORD</td><td>ConnectAccept</td><td>Connected/Not connected
Not connected:0xFFFF (65535)
Connected: Some other value
Normally set to 0 (when connected).
Select 0xFFFF (when not connected) if Slaves are to be added later or reserved in the scan list. When 0xFFFF is selected, I/O communications do not occur even if the Slave is registered in the scan list. No communications errors will occur either. Once the Slave has been added use SCAN_ConnectSlaveDevice() to start I/O communications.</td></tr>
<tr><td>WORD</td><td>HeatBeatTime</td><td>COS/cyclic send interval time (ms)
Setting range:
0x0000 (0): Default (1 s)
0x000A to 0xFFFF (10 to 65535): 10 to 65,535 ms
The actual HeatBeatTime will be the greater of the following values:
• The value set here
• Communications cycle time x 2
• 10 ms</td></tr>
<tr><td>WORD</td><td>Output1PathLen</td><td>Output 1 connection path size: 0x00 to 0x10 (0 to 16)</td></tr>
<tr><td>WORD</td><td>Output1Path[16]</td><td>Output 1 connection path</td></tr>
<tr><td>WORD</td><td>Input1PathLen</td><td>Input 1 connection path size: 0x00 to 0x10 (0 to 16)</td></tr>
<tr><td>WORD</td><td>Input1Path[16]</td><td>Input 1 connection path</td></tr>
<tr><td>WORD</td><td>Output2PathLen</td><td>Output 2 connection path size: 0x00 to 0x10 (0 to 16) Set this area to 0 if specifying automatic selection for the scan type or specifying only one connection.</td></tr>
<tr><td>WORD</td><td>Output2Path[16]</td><td>Output 2 connection path</td></tr>
<tr><td>WORD</td><td>Input2PathLen</td><td>Input 2 connection path size: 0x00 to 0x10 (0 to 16) Set this area to 0 if specifying automatic selection for the scan type or specifying only one connection.</td></tr>
<tr><td>WORD</td><td>Input2Path[16]</td><td>Input 2 connection path</td></tr>
<tr><td>WORD</td><td>Reserve</td><td>Reserved area, so set to 0.</td></tr>
<tr><td>WORD</td><td>Reserve2</td><td>Reserved area, so set to 0.</td></tr>
</table>

Note When not specifying a connection path, set the connection path size to 0 and the connection path to NULL.

Registering a Slave in the Scan List (Detailed): SCAN_RegisterSlaveDeviceEx()

Application Range Open status

Function Registers detailed information in the scan list for a Slave.

Call Format BOOL SCAN_RegisterSlaveDeviceEx(DWORD *Handle*, WORD, *MacId*, SCAN_DEV **DeviceInfo*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	MacId	Node address to register in the scan list. Setting range: 0x00 to 0x3F (0 to 63)
SCAN_DEV *	Device	Buffer address where detailed Slave information is stored.

Return Value TRUE is returned if the specified Slave was successfully registered in the scan list. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description Slave information that is already registered in the scan list will be overwritten.

Reading a Scan List Slave (Detailed): SCAN_GetSlaveDeviceEx()

Application Range Open status

Function Reads detailed Slave information registered in the scan list.

Call Format BOOL SCAN_GetSlaveDeviceEx(DWORD *Handle*, WORD, *MacId*, SCAN_DEV **DeviceInfo*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	MacId	Node address to register Slave information. Setting range: 0x00 to 0x3F (0 to 63)
SCAN_DEV *	Device	Buffer address where detailed Slave information is stored.

Return Value TRUE is returned if the specified Slave information was successfully read from the scan list. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

FSCAN_DEV Structure

This structure defines the format of the individual Slave information used when nodes are registered in the scan list as a group.
This structure is used within the FILE_SCAN_DEV structure and in the SCAN_SetScanList() function.

Type	Name	Contents
WORD	MacId	Node address to register
SCAN_DEV	ScanDevice	Slave information

FILE_SCAN_DEV

This structure defines the format of the grouped Slave information used when nodes are registered in the scan list as a group.
This structure is used in the SCAN_SetScanList() function.

Type	Name	Contents
WORD	DeviceCount	Number of Slaves set
FSCAN_DEV	Device[DEVICE_MAX]	Scan Slave information

Registering Multiple Slaves in the Scan List: SCAN_SetScanlist()

Application Range

Open status

Function

Registers multiple Slaves in the scan list as a group.

Call Format

BOOL SCAN_SetScanlist(DWORD *Handle*, LPCTSTR *FilePath*,)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
LPCTSTR	LPCTSTR	Buffer address of the scan list file path (FILE_SCAN_DEV structure format file)

Return Value

TRUE is returned if the scan list was registered successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

4-4-2 I/O Communications Service API Functions

Starting Remote I/O Communications: SCAN_StartScan()

Application Range

Online status

Function

Starts remote I/O communications.

Call Format

BOOL SCAN_StartScan(DWORD *Handle*, BOOL *ErrStop*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
BOOL	ErrStop	Specifies whether to continue or stop remote I/O when a communications error occurs. TRUE: Stop after a communications error FALSE: Continue after a communications error

Return Value

TRUE is returned if remote I/O communications started normally in the specified Board. FALSE is returned if an error occurred, such as the lack of a corresponding event. Detailed error information can be read with the GetLastError() function.

Description

Performs remote I/O communications with the Slaves registered in the scan list.

Several seconds are required after this function is called before I/O communications will actually start.

Scan list operation API Functions must be used to create a scan list before this function is called.

When TRUE has been specified for the ErrStop variable, remote I/O communications with all of the Slaves will be stopped if a communications error occurs during remote I/O communications.

Stopping Remote I/O Communications: SCAN_StopScan()

Application Range I/O communications execution status

Function Stops remote I/O communications.

Call Format BOOL SCAN_StopScan(DWORD *Handle*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()

Return Value TRUE is returned if remote I/O communications stopped normally in the specified Board. FALSE is returned if an error occurred, such as the lack of a corresponding event. Detailed error information can be read with the GetLastError() function.

Writing the Communications Cycle Time: SCAN_SetScanTimeValue()

Application Range Open status

Function Sets the communications cycle time.

Call Format BOOL SCAN_SetScanTimeValue(DWORD *Handle*, WORD *ScanTime*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	ScanTime	Communications cycle time (ms) Setting range: 0x000 (0): Automatic (Always executed at the fastest speed.) 0x0001 to 0x01F4 (1 to 500): 1 to 500 ms

Return Value TRUE is returned if the communications cycle time was set successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description The default communications cycle time is automatic (maximum). The communications cycle time for the automatic setting is calculated using the formula listed in *6-1-1 Communications Cycle Time*.

If a value shorter than the communications cycle time used under the automatic setting is set, the set value changes to that specified value but the actual communications use the communications cycle time from the automatic setting.

Reading the Communications Cycle Time: SCAN_GetScanTimeValue()

Application Range Open status

Function Reads the set communications cycle time.

Call Format BOOL SCAN_GetScanTimeValue(DWORD *Handle*, WORD**ScanTime*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	ScanTime	Buffer address for receiving the communications cycle time Receiving range: 0x0000 (0): Automatic (Always executed at the fastest speed.) 0x0001 to 0x01F4 (1 to 500): 1 to 500 ms

Return Value TRUE is returned if the communications cycle time was read successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description This function gets the set value (in ms).

Use SCAN_GetCycleTime(), SCAN_GetMaxCycleTime(), or SCAN_GetMin CycleTime() to get the real cycle time.

Storing the Communications Cycle Time: SCAN_StoreScanTimeValue()

Application Range Open status

Function Saves the set communications cycle time to non-volatile memory.

Call Format BOOL SCAN_StoreScanTimeValue(DWORD *Handle*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()

Return Value TRUE is returned if the communications cycle time was successfully written to non-volatile memory. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description The value for the communications cycle time saved to non-volatile memory is enabled when SCAN_LoadScanTimeValue() is used.

Loading the Communications Cycle Time: SCAN_LoadScanTimeValue()

Application Range Open status

Function Loads the communications cycle time from non-volatile memory and uses it as the set value.

Call Format BOOL SCAN_LoadScanTimeValue(DWORD Handle)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()

Return Value TRUE is returned if the communications cycle time was successfully loaded from non-volatile memory. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description If this function is used without executing SCAN_SetScanTimeValue(), the communications cycle time will be the default (automatic).

ACTUAL_SLAVE_INFO Structure

This structure stores information obtained when reading information about the Slaves connected to the network. This structure is used in the SCAN_Get ActualSlaveDevice() function.

<table>
<tr><th>Type</th><th>Name</th><th>Contents</th></tr>
<tr><td>WORD</td><td>VendorID</td><td>Vendor ID
Scan list disabled mode:0xFFFF (65535)
No Slaves connected:0xFF00 (65280)</td></tr>
<tr><td>WORD</td><td>ProductType</td><td>Product type
Scan list disabled mode:0xFFFF (65535)
No Slaves connected:0xFF00 (65280)</td></tr>
<tr><td>WORD</td><td>ProductCode</td><td>Product code
Scan list disabled mode:0xFFFF (65535)
No Slaves connected:0xFF00 (65280)</td></tr>
<tr><td>WORD</td><td>ScanType</td><td>Scan type
0x8000 (32768) for automatic selection
<table>
<tr><td>Bit 5</td><td>Bit 4</td><td>Bit 3</td><td>Bit 2</td><td>Bit 1</td><td>Bit 0</td></tr>
<tr><td>CY</td><td>COS</td><td>Reserved</td><td>BS</td><td>Poll</td><td>Reserved</td></tr>
</table>
BS: Bit Strobe
CY: Cyclic</td></tr>
<tr><td>WORD</td><td>Output1</td><td>Size of output data 1 in bytes
No Slaves connected: 0xFF00 (65280)</td></tr>
<tr><td>WORD</td><td>Input1</td><td>Size of input data 1 in bytes
No Slaves connected: 0xFF00 (65280)</td></tr>
<tr><td>WORD</td><td>Output2</td><td>Size of output data 2 in bytes
Two I/O connections not set: 0×000 (0)
No slaves connected: 0×FF00 (65280)</td></tr>
<tr><td>WORD</td><td>Input2</td><td>Size of input data 2 in bytes
Two I/O connections not set: 0×000 (0)
No slaves connected: 0×FF00 (65280)</td></tr>
</table>

Reading Slave Information: SCAN_GetActualSlaveDevice()

Application Range

I/O communications execution status

Function

Reads information about the specified Slave.

Call Format

BOOL SCAN_GetActualSlaveDevice(DWORD *Handle*, WORD *MacId*, ACTUAL_SLAVE_INFO**SlaveInfo*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	MacId	Node address of the Slave whose information is being read
ACTUAL_SLAVE_INFO*	SlaveInfo	Buffer address for receiving slave information Slave information is stored to the buffer address specified here when the function is completed.

Return Value

TRUE is returned if the Slave information was read successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

This function gets information from Slaves connected to the network.

Information can even be obtained for Slaves with verification errors.

Use SCAN_GetSlaveDevice() or SCAN_GetSlaveDeviceEx() to get Slave information for Slaves registered on the scan list.

Starting Slave I/O Communications: SCAN_ConnectSlaveDevice()

Application Range I/O communications execution status

Function Starts I/O communications with the specified Slave.

Call Format BOOL SCAN_ConnectSlaveDevice(DWORD *Handle*, WORD *MacId*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	MacId	Slave's node address

Return Value TRUE is returned if the I/O communications with the specified Slave were started successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description This function can be used in I/O communications execution status.

During execution of I/O communications, this function clears I/O communications stop status (withdrawal status) for a specific Slave and starts I/O communications.

Use this function with SCAN_DisconnectSlaveDevice() when replacing Slaves or to make reservations for expected additional Slaves (when registering to the scan list but not performing I/O communications).

Stopping Slave I/O Communications: SCAN_DisconnectSlaveDevice()

Application Range I/O communications execution status

Function Stops I/O communications with the specified Slave.

Call Format BOOL SCAN_DisconnectSlaveDevice(DWORD *Handle*, WORD *MacId*);

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	MacId	Slave's node address

Return Value TRUE is returned if the I/O communications with the specified Slave were stopped successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description This function can be used in I/O communications execution status.

During execution of I/O communications, this function clears I/O communications stop status (withdrawal status) for a specific Slave and stops I/O communications.

Use this function with SCAN_ConnectSlaveDevice() when replacing Slaves or to make reservations for expected additional Slaves (when registering to the scan list but not performing I/O communications).

Slaves for which I/O communications were stopped (withdrawn) using this function will not have verification or communications errors.

4-4-3 I/O Data Access Service API Functions

Refreshing Master I/O Data: SCAN_IoRefresh()

Application Range

Open status

Function

Refreshes the all Slave data in the I/O area of the Board's Master function.

Call Format

BOOL SCAN_IoRefresh(DWORD *Handle*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()

Return Value

TRUE is returned if the input and output areas of the specified Board's Master function were successfully refreshed. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

This function executes data exchange between the I/O areas accessible using API functions and I/O areas on the Board used for actual communications, and refreshes these I/O areas.

This function refreshes all Slave I/O areas.

Use the SCAN_GetInData() after this function is completed to get the input data.

For output data, first execute SCAN_SetOutData() for the required Slaves, and then execute this function.

IO_DATA_CTL Structure

This structure defines the data format used when accessing the Board's I/O areas. This structure is used in the SCAN_GetInData(), SCAN_SetOutData(), SCAN_GetSlaveOutData(), and SCAN_SetSlaveInData() functions.

Type	Name	Contents
DWORD	DataSize	Amount of data in bytes Setting range: 0x00 to 0xC8 (0 to 200 bytes)
BYTE	Data[200]	Buffer where data is stored

Specify the read data size or write data size under DataSize.

Reading Slave Input Data: SCAN_GetInData()

Application Range

Open status

Function

Reads the specified Slave's input data from the Board's input area.

Call Format

BOOL SCAN_GetInData(DWORD *Handle*, WORD *MacId*, IO_DATA_CTL **InData1*, IO_DATA_CTL **InData2*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	MacId	Node address to read Setting range: 0x00 to 0x3F (0 to 63)
IO_DATA_CTL*	InData1	Buffer address where input data 1 is stored
IO_DATA_CTL*	InData2	Buffer address where input data 2 is stored (NULL when not used.)

Return Value

TRUE is returned if the specified input data was read successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

The most recent Slave inputs will not be reflected in the Board's input area unless this function is executed after SCAN_IoRefresh().

Specify NULL as the input data 2 storage buffer address if only one I/O connection is used with Slaves (e.g., when using SCAN_RegisterSlaveDevice()).

Specify the input data size for the specified node address as the data size for the IO_DATA_CTL structure. The read I/O data is stored in the data storage buffer of the IO_DATA_CTL structure.

Writing Slave Output Data: SCAN_SetOutData()

Application Range

Open status

Function

Writes the specified Slave's output data to the Board's output area.

Call Format

BOOL SCAN_SetOutData(DWORD *Handle*, WORD *MacId*, IO_DATA_CTL **OutData1*, IO_DATA_CTL **OutData2*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	MacId	Node address to write Setting range: 0x00 to 0x3F (0 to 63)
IO_DATA_CTL*	OutData1	Buffer address where output data 1 is stored
IO_DATA_CTL*	OutData2	Buffer address where output data 2 is stored (NULL when not used.)

Return Value

TRUE is returned if the specified output data was written successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

The Board's output area will not be reflected in the most recent Slave outputs unless SCAN_IoRefresh() is executed after this function.

Specify NULL as the input data 2 storage buffer address if only one I/O connection is used with Slaves (e.g., when using SCAN_RegisterSlaveDevice()).

Specify the output data size for the specified node address as the data size for the IO_DATA_CTL structure. The output data to Slaves is stored in the data storage buffer of the IO_DATA_CTL structure.

Sending COS Output Data: SCAN_SendMasterCosToSlave()

Application Range

I/O communications executing status

Function

Sends output data to a Slave communicating through a COS connection.

Call Format

BOOL SCAN_SendMasterCosToSlave(DWORD *Handle* DWORD *MacId*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
DWORD	MacId	Node address to write output data Designation range: 0X0 to 0X3F (0 to 63)

Return Value

TRUE is returned if the transmission of output data to the Slave proceeded normally. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

Before output data is sent to Slaves through a COS connection, the SCAN_SetOutData() function must be used to set the output data.

COS connections do not always produce immediate communications. Transmission may be delayed when several nodes are using COS communications.

4-5 Slave Function API Functions

4-5-1 Slave Scan List Operation API Functions

SELF_DEV Structure

This structure defines the format of the registration information used when registering I/O information in the Slave's scan list. This structure is used in the SCAN_RegisterSelfSlaveDevice() and SCAN_GetSelfSlaveDevice() functions. When nodes are registered in the scan list as a group, the FSCAN_DEV and FILE_SCAN_DEV structures are used.

<table>
<tr><th>Type</th><th>Name</th><th>Contents</th></tr>
<tr><td>WORD</td><td>ScanType</td><td>Scan type
Specifies the connection type used.
Up to 2 scan types can be selected, although COS and cyclic cannot be selected simultaneously.
0x8000 (32768) for automatic selection
<table>
<tr><td>Bit 5</td><td>Bit 4</td><td>Bit 3</td><td>Bit 2</td><td>Bit 1</td><td>Bit 0</td></tr>
<tr><td>CY</td><td>COS</td><td>Reserved</td><td>BS</td><td>Poll</td><td>Reserved</td></tr>
</table>
BS: Bit Strobe
CY: Cyclic</td></tr>
<tr><td>WORD</td><td>Output1</td><td>Output data 1 (Master to Slave)
Size (bytes): 0x00 to 0xC8 (0 to 200)</td></tr>
<tr><td>WORD</td><td>Input1</td><td>Input data 1 (Slave to Master)
Size (bytes): 0x00 to 0xC8 (0 to 200)</td></tr>
<tr><td>WORD</td><td>Output2</td><td>Output data 2 (Master to Slave)
Size (bytes): 0x00 to 0xC8 (0 to 200)
Set this area to 0 if specifying automatic selection for the scan type or specifying only one connection.</td></tr>
<tr><td>WORD</td><td>Input2</td><td>Input data 2 (Slave to Master)
Size (bytes): 0x00 to 0xC8 (0 to 200)
Set this area to 0 if specifying automatic selection for the scan type or specifying only one connection.</td></tr>
<tr><td>WORD</td><td>ConnectAccept</td><td>Connected/Not connected
Not connected:0xFFFF (65535)
Connected:Some other value
Normally set to 0 (when connected).
Set to 0xFFFF (when not connected) if not performing I/O communications even though the slave is registered in the scan list. A "No Slave" error will occur on the Master.</td></tr>
<tr><td>WORD</td><td>ErrorOutData</td><td>Indicates whether output data is retained or cleared when a communications error occurs.
Retain output data: 0xFFFF (65535)
Clear output data: Some other value
Specifies the output data (Master to Slave) status for when a communications error occurs with the Master.</td></tr>
</table>

Registering a Slave Scan List: SCAN_RegisterSelfSlaveDevice()

Application Range Open status

Function Registers Slave function information in the Slave scan list.

Call Format BOOL SCAN_RegisterSelfSlaveDevice(DWORD *Handle*, SELF_DEV* *DeviceInfo*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
SELF_DEV*	DeviceInfo	Buffer address where Slave function information is stored.

Return Value TRUE is returned if the specified Slave information was successfully registered in the scan list. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description If connection is specified, I/O communications will automatically start if a request to establish a connection is received from the Master after this function is completed.

Deleting a Slave Scan List: SCAN_RemoveSelfSlaveDevice()

Application Range Open status

Function Deletes the Slave scan list information.

Call Format BOOL SCAN_RemoveSelfSlaveDevice(DWORD *Handle*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()

Return Value TRUE is returned if the Slave scan list was successfully deleted. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Reading a Slave Scan List: SCAN_GetSelfSlaveDevice()

Application Range Open status

Function Reads the Slave scan list information.

Call Format BOOL SCAN_GetSelfSlaveDevice(DWORD *Handle*, SELF_DEV* *DeviceInfo*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
SELF_DEV*	DeviceInfo	Buffer address for receiving the Slave function information.

Return Value TRUE is returned if the Slave scan list was read successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description The Slave scan list information is stored in the SELF_DEV structure when this function is completed.

Storing a Slave Scan List: SCAN_StoreSlaveScanlist()

Application Range Open status

Function Saves the Slave scan list information to non-volatile memory.

Call Format BOOL SCAN_StoreSlaveScanlist(DWORD *Handle*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()

Return Value TRUE is returned if the Slave scan list was successfully written to non-volatile memory. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description The Slave scan list information saved to non-volatile memory can be registered as the Slave scan list by using SCAN_LoadSlaveScanlist().

Loading a Slave Scan List: SCAN_LoadSlaveScanlist()

Application Range Open status

Function Loads the Slave scan list information from non-volatile memory and registers the information as the Slave scan list.

Call Format BOOL SCAN_LoadSlaveScanlist(DWORD Handle)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()

Return Value TRUE is returned if the Slave scan list was successfully loaded from non-volatile memory. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description Use SCAN_StoreSlaveScanlist() to save the Slave scan list to non-volatile memory.

4-5-2 I/O Communications Service API Functions

Starting Master I/O Communications: SCAN_ConnectMasterDevice()

Application Range Online status

Function Starts I/O communications with the specified Master.

Call Format BOOL SCAN_ConnectMasterDevice(DWORD *Handle*, WORD *ErrorOutData*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	ErrorOutData	Holds/clears output data at communications errors Hold output data: 0×FFFF (65535) Clear: Any other value

Return Value TRUE is returned if the I/O communications with the specified Master were started successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description Starts I/O communications with Masters that have stopped communications.

Slave information must be registered using SCAN_RegisterSelfSlaveDevice() to use this function.

Stopping Master I/O Communications: SCAN_DisconnectMasterDevice()

Application Range

I/O communications execution status

Function

Stops I/O communications with the specified Master.

Call Format

BOOL SCAN_DisconnectMasterDevice(DWORD *Handle*);

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()

Return Value

TRUE is returned if the I/O communications with the specified Master were stopped successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

Stops I/O communications with Masters. The Master will have a communications error.

The stopped I/O communications can be restarted using SCAN_Connect MasterDevice().

4-5-3 I/O Data Access Service API Functions

Refreshing Slave I/O Data: SCAN_SlaveIoRefresh()

Application Range

Open status

Function

Refreshes the input and output areas of the Board's Slave function with the most recent data from remote I/O communications. Refreshes the data in the I/O areas of the Slave function.

Call Format

BOOL SCAN_SlaveIoRefresh(DWORD *Handle*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()

Return Value

TRUE is returned if the input and output areas of the specified Board's Slave function were successfully refreshed. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

This function executes data exchange between the Slave function's I/O areas accessible using API functions and Slave function's I/O areas on the Board used for actual communications, and refreshes these I/O areas.

Use the SCAN_GetSlaveOutData() after this function is completed to get the input data.

For output data, first execute SCAN_SetSlaveInData() for the required Slaves, and then execute this function.

IO_DATA_CTL Structure

This structure defines the data format used when accessing the Board's I/O areas. This structure is used in the SCAN_GetInData(), SCAN_SetOutData(), SCAN_GetSlaveOutData(), and SCAN_SetSlaveInData() functions.

Type	Name	Contents
DWORD	DataSize	Amount of data in bytes Setting range: 0x00 to 0xC8 (0 to 200 bytes)
BYTE	Data[256]	Buffer where data is stored

Specify the read data size or write data size under DataSize.

Reading Master Output Data: SCAN_GetSlaveOutData()

Application Range Open status

Function Reads output data from the Master from the Board's Slave function output area.

Call Format BOOL SCAN_GetSlaveOutData(DWORD *Handle*, IO_DATA_CTL **OutData1*, IO_DATA_CTL **OutData2*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
IO_DATA_CTL*	OutData1	Buffer address where output data 1 is stored
IO_DATA_CTL*	OutData2	Buffer address where output data 2 is stored (NULL when not used.)

Return Value TRUE is returned if the specified output data was read successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description The Board's Slave function output area will not reflect in the most recent outputs from the Master unless this function is executed after SCAN_SlaveIoRefresh().

Specify NULL as the output data 2 storage buffer address if only one I/O connection is used with the Slave function.

Specify the output data size from the Master when using the Slave function (OUT size) as the data size for the IO_DATA_CTL structure. The output data read from the Master is stored in the data storage buffer of the IO_DATA_CTL structure.

Writing Master Input Data: SCAN_SetSlaveInData()

Application Range Open status

Function Writes input data to the Master to the Board's Slave function input area.

Call Format BOOL SCAN_SetSlaveInData(DWORD *Handle*, WORD *MacId*, IO_DATA_CTL **InData1*, IO_DATA_CTL **InData2*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
IO_DATA_CTL*	InData1	Buffer address where input data 1 is stored
IO_DATA_CTL*	InData2	Buffer address where input data 2 is stored (NULL when not used.)

Return Value

TRUE is returned if the specified input data was written successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

The Board's Slave function input area will not reflect in the most recent Master inputs unless SCAN_IoRefresh() is executed after this function.

Specify NULL as the input data 2 storage buffer address if only one I/O connection is used with the Slave function.

Specify the input size (IN size) for the Master using the Slave function as the data size for the IO_DATA_CTL structure. The input data to the Master is stored in the data storage buffer of the IO_DATA_CTL structure.

Sending COS Input Data: SCAN_SendSlaveCosToMaster()

Application Range

I/O communications executing status

Function

Sends input data to a Master communicating through a COS connection.

Call Format

BOOL SCAN_SendSlaveCosToMaster(DWORD *Handle*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()

Return Value

TRUE is returned if the transmission of input data to the Master proceeded normally. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

The SCAN_SetSlaveInData() function must be used to set the input data before the input data is sent to the Master through a COS connection.

COS connections do not guarantee immediate communications. Transmission may be delayed when several nodes are using COS communications.

4-6 Explicit Message API Functions

4-6-1 Message Monitoring Timer Service API Functions

Registering the Message Monitoring Timer: SCAN_SetMessageTimerValue()

Application Range

Open status

Function

Registers the message monitoring timer value of the explicit client connection with the specified node address.

Call Format

BOOL SCAN_SetMessageTimerValue(DWORD *Handle*, WORD *MacId*, WORD *Epr*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	MacId	Desired node address Setting range: 0x00 to 0x3F (0 to 63)
WORD	Epr	Message monitoring timer value to be registered (ms) Setting range: 0x1F4 to 0x7530 (500 to 30,000)

Return Value

TRUE is returned when the message monitoring timer value was successfully registered. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

Use this function to change the message monitoring timer value.

The default value (2 s, or 2,000 ms) will be used when 0 is set as the message monitoring timer value.

Reading the Message Monitoring Timer: SCAN_GetMessageTimerValue()

Application Range

Open status

Function

Reads the message monitoring timer value of the explicit client connection with the specified node address.

Call Format

BOOL SCAN_GetMessageTimerValue(DWORD *Handle*, WORD *MacId*, WORD **Epr*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	MacId	Desired node address Setting range: 0x00 to 0x3F (0 to 63)
WORD*	Epr	Buffer address for obtaining the message monitoring timer value (ms).

Return Value

TRUE is returned when the message monitoring timer value was successfully read. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

Use this function to check the current message monitoring timer value.

The default message monitoring timer value, 0 (2 s), will be read when the value is not set using SCAN_SetMessageTimerValue().

Storing the Message Monitoring Timer List: SCAN_StoreMessageTimerValueList()

Application Range

Open status

Function

Saves the explicit client connection's message monitoring timer list to non-volatile memory.

Call Format

BOOL SCAN_StoreMessageTimerValueList(DWORD *Handle*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()

Return Value

TRUE is returned when the message monitoring timer list was successfully saved to non-volatile memory. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

Message monitoring timer values for all node addresses are saved.

The message monitoring timer list saved to non-volatile memory is enabled when SCAN_LoadMessageTimerValueList() is used.

Loading the Message Monitoring Timer List: SCAN_LoadMessageTimerValueList()

Application Range

Open status

Function

Reads the explicit client connection monitoring timer list from non-volatile memory and registers it as the message monitoring timers.

Call Format

BOOL SCAN_LoadMessageTimerValueList(DWORD *Handle*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()

Return Value

TRUE is returned when the message monitoring timer list was successfully loaded from non-volatile memory. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

The message monitoring timer values for all node addresses are refreshed using the saved values.

If this function is used without SCAN_StoreMessageTimerValueList() being executed, the default value, 0 (2 s), will be used as the message monitoring timer value for all node addresses.

4-6-2 Client Message Service API Functions

Registering a Client Response Message: SCAN_RegClientEvtNotifyMessage()

Application Range

Open status

Function

Registers the Windows message sent when a client response reception event occurs from the specified node address.

Call Format

BOOL SCAN_RegClientEvtNotifyMessage(DWORD *Handle*, WORD *MacId*, DWORD *ThreadId*, HWND *hWnd*, UINT *Msg*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	MacId	Node address to check for events Setting range: 0x00 to 0x3F (0 to 63)
DWORD	ThreadId	The thread ID to notify. (No setting = NULL)
HWND	HWnd	Specifies the window handle to notify. (No setting = NULL)
UINT	Msg	Notification message (event ID) Range: WM_USER + 0x100 to WM_USER + 0x7FFF

Return Value

TRUE is returned if the message registration was completed successfully. FALSE is returned if an error occurred such as null values for both the thread ID and window handle. Detailed error information can be read with the GetLastError() function.

Description

The client response source (remote) node address is sent to WPARAM with the notification message. The reception response size (in bytes) is sent to LPARAM with the notification message.

Clearing a Client Response Message: SCAN_UnRegClientEvtNotifyMessage()

Application Range

Open status

Function

Clears the registered notification message of the specified node address.

Call Format

BOOL SCAN_UnRegClientEvtNotifyMessage(DWORD *Handle*, WORD *MacId*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	MacID	Desired node address

Return Value TRUE is returned if the message registration was cleared successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Checking for Client Response Events: SCAN_PeekClientEvent()

Application Range Online status

Function Checks whether there is a client response event.

Call Format BOOL SCAN_PeekClientEvent(DWORD *Handle*, WORD *MacId*,)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	MacId	Node address being checked for response destination (remote) Setting range: 0x00 to 0x3F (0 to 63)

Return Value TRUE is returned if there is a client response event in the event queue. FALSE is returned if an error occurred or there is not a client response event in the event queue. Detailed error information can be read with the GetLastError() function.

Reading the Length of a Client Response: SCAN_GetClientEventLength()

Application Range Online status

Function Determines the length of the client response.

Call Format BOOL SCAN_GetClientEventLength(DWORD *Handle*, WORD *MacId*, DWORD* *Length*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	MacId	Destination (remote) node address for response for which response size is being read.
DWORD*	Length	Buffer address where response size is received.

Return Value TRUE is returned if the size of the event was read successfully. FALSE is returned if an error occurred, such as the lack of a client event. Detailed error information can be read with the GetLastError() function.

Description Before getting the response message using SCAN_ReceiveClientExplicit(), set aside a buffer for storing service data that is at least as large as the response read using this function.

CLIENT_REQ Structure

This structure defines the format of client request explicit messages. It is used with the SCAN_SendClientExplicit() function.

Type	Name	Contents
DWORD	MessageID	Message ID Set a value to enable the application to identify the message. Set 0 if no identification is required.
WORD	MacId	Destination (remote) node address
WORD	ServiceCode	Service code
WORD	ClassID	Class ID
WORD	InstanceID	Instance ID
WORD	DataLength	The amount of service data in bytes Setting range: 0x0 to 0x228 (0 to 552)
BYTE*	ServiceData	Buffer address where the service data is stored. The buffer stores the number of bytes of service data specified under Data Length.

Sending a Client Explicit Message: SCAN_SendClientExplicit()

Application Range Online status

Function Sends a client request message.

Call Format BOOL SCAN_SendClientExplicit(DWORD *Handle*, CLIENT_REQ**Msg*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
CLIENT_REQ*	Msg	Buffer address where the request message is stored

Return Value TRUE is returned if the send event registration was completed successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description The message ID, destination node address, service data size, the data for the service being sent as a request (service code, class ID, instance ID, and service data) needs to be saved in the CLIENT_REQ structure.

Set a value to the message ID in the CLIENT_REQ structure when the message needs to be identified by the application. Set 0 if the message doesn't need to be identified.
The set message ID is saved in the CLIENT_RES structure's message ID after SCAN_ReceiveClientExplicit() has been completed.

Note Always enable retries when sending explicit messages because explicit messages may not be received, depending on the type of destination Slave.

CLIENT_RES Structure

This structure defines the format of client response explicit messages. This structure is used in the SCAN_ReceiveClientExplicit() function.

Type	Name	Contents
DWORD	MessageID	Message ID The message ID set when the request was sent is stored when the function is completed.
WORD	MacId	Destination (remote) node address Specifies the remote node address for the client response to be obtained before calling the function.

Type	Name	Contents
WORD	ServiceCode	Service code Stores the service code when function processing is completed.
WORD	DataLength	The size (in bytes) of the buffer containing the service data Setting range: 0x0 to 0x228 (0 to 552) Specifies the size of the buffer for storing the service data before calling the function. A buffer for storing service data must be reserved that is the size of the reception response buffer obtained using the client response communications message or SCAN_GetClientEventLength(). When the function is completed, the number of bytes of service data that was actually stored will be set.
BYTE*	ServiceData	Buffer address where service data is stored. A buffer of the number of bytes specified under DataLength must be set aside before the function is called. When the function is completed, the received service data is stored to the buffer address specified here.

Reading a Client Explicit Message: SCAN_ReceiveClientExplicit()

Application Range Online status

Function Gets a client response message from the reception queue.

Call Format BOOL SCAN_ReceiveClientExplicit(DWORD *Handle*, CLIENT_RES**Msg*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
CLIENT_RES*	Msg	Buffer address where the response message is stored

Return Value TRUE is returned if the response message was read successfully from the specified node address. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description The CLIENT_RES structure's destination (remote) node address, service data storage buffer size, and service data storage buffer address must be set when calling this function.

Reserve a buffer large enough to store the response message. Check the event data length with the notification message or SCAN_GetClient EventLength() function.

If this function is completed normally, the message ID, service code, and size of the buffer containing service data are saved in the CLIENT_RES structure and the received service data (response) is stored in the specified service data storage buffer.

4-6-3 Server Message Service API Functions

Registering an Object Class ID: SCAN_RegObjectClass()

Application Range Open status

Function Registers the object class implemented in the application program.

Call Format BOOL SCAN_RegObjectClass(DWORD *Handle*, WORD *ClassId*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	ClassId	Registers object class ID.

Return Value TRUE is returned if the object class ID was successfully registered to the specified Board. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description To register an object, a server request message addressed to that object must be received and a response returned.

The following list shows the object class IDs that can be registered. Up to 16 can be registered.

0x0000 to 0x0063 (excluding 0x0002, 0x0003, 0x0005, 0x002B, and 0x002F), 0x0064 to 0x007F, 0x0088 to 0x008F, 0x0098 to 0x009F, 0x00A8 to 0x00AF, 0x00B8 to 0x00BF, 0x00C0 to 0x00C7

When Identity Object Class (ID=1) is registered, the service code must support Reset processing.

Clearing an Object Class ID: SCAN_UnRegObjectClass()

Application Range Open status

Function Clears registration of an object class.

Call Format BOOL SCAN_UnRegObjectClass(DWORD *Handle*, WORD *ClassId*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	ClassId	Object class ID of the object class being deleted.

Return Value TRUE is returned if the object class ID was successfully deleted from the specified Board. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Registering a Server Notification Message: SCAN_RegServerEvtNotifyMessage()

Application Range Open status

Function Registers the Windows message that notifies when a server request reception event for the registered object has occurred.

Call Format BOOL SCAN_RegServerEvtNotifyMessage(DWORD *Handle*, WORD *ClassId*, DWORD *ThreadId*, HWND *hWnd*, UINT *Msg*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	ClassId	Object class ID used in server request reception event.
DWORD	ThreadId	The thread ID to notify. (No setting = NULL)

Type	Name	Contents
HWND	hWnd	Specifies the window handle to notify. (No setting = NULL)
UINT	Msg	Notification message Range: WM_USER + 0x100 to WM_USER + 0x7FFF

Return Value

TRUE is returned if registration of the notification message was completed successfully. FALSE is returned if an error occurred such as null values for both the thread ID and window handle. Detailed error information can be read with the GetLastError() function.

Description

Along with the notification message, notification of the object class ID of the server request recipient is sent as WPARAM and notification of the reception request size (in bytes) is sent as LPARAM.

Clearing a Server Notification Message: SCAN_UnRegServerEvtNotifyMessage()

Application Range

Open status

Function

Clears registration of the notification message for the specified object class ID.

Call Format

BOOL SCAN_UnRegServerEvtNotifyMessage(DWORD *Handle*, WORD *ClassId*,)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	ClassId	Object class ID being cleared

Return Value

TRUE is returned if registration of the notification message was cleared successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Checking Server Request Events: SCAN_PeekServerEvent()

Application Range

Open status

Function

Checks whether there is a server request event addressed to the specified class ID.

Call Format

BOOL SCAN_PeekServerEvent(DWORD *Handle*, WORD *ClassId*,)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	ClassId	Class ID being checked for event

Return Value

TRUE is returned if there is a server request event addressed to the specified class ID; FALSE is returned if an error occurred or there is not. Detailed error information can be read with the GetLastError() function.

Reading the Server Request Size: SCAN_GetServerEventLength()

Application Range

Open status

Function

Gets the server request size (in bytes) for the specified object class ID.

Call Format

BOOL SCAN_GetServerEventLength(DWORD *Handle*, WORD *ClassId*, DWORD**Length*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	ClassId	Class ID of the destination for the request for which the request size is being read.
DWORD*	Length	Buffer address for receiving the request size.

Return Value TRUE is returned if the size of the event was read successfully. FALSE is returned if an error occurred, such as the lack of a corresponding event. Detailed error information can be read with the GetLastError() function.

Description Set aside a buffer for storing service data that is at least as large as the request obtained using this function, before using SCAN_Receive ServerExplicit() to get the request message.

SERVER_REQ Structure

This structure defines the format of server request explicit messages. This structure is used in the SCAN_ReceiveServerExplicit() function.

Type	Name	Contents
DWORD	MessageID	Message ID Stores the server request message ID when the function is completed.
WORD	ServiceCode	Service code Stores the service code for the server request when the function is completed.
WORD	ClassID	Class ID Specifies the class ID for the server request to be obtained, before the function is called.
WORD	InstanceID	Instance ID Stores the instance ID for the server request when the function is completed.
WORD	DataLength	The size (in bytes) of the buffer containing the service data Setting range: 0x0 to 0x228 (0 to 552) Specifies the size of the buffer reserved for storing the service data before calling the function. A buffer for storing service data must be reserved that is the size of the reception request buffer obtained using the server notification message or SCAN_GetServerEventLength(). When the function is completed, the number of bytes of service data that was actually stored will be set.
BYTE*	ServiceData	Service data storage buffer address The number of bytes of service data specified under DataLength must be set aside. When the function is completed, the received service data is stored in the buffer at the address specified here.

Reading a Server Explicit Message: SCAN_ReceiveServerExplicit()

Application Range Online status

Function Gets a server request message from the reception queue.

Call Format BOOL SCAN_ReceiveServerExplicit(DWORD *Handle*, SERVER_REQ**Msg*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
SERVER_REQ*	Msg	Buffer address where the request message is stored

Return Value

TRUE is returned if the request message was read successfully from the specified Board. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

The SERVER_REQ structure's ClassID, size of the service data storage buffer, and address of the service data storage buffer must be set before calling this function.

Reserve a buffer large enough to store the request message. Check the event data length with the notification message or SCAN_GetServerEventLength() function.

When this function is completed normally, the message ID automatically set by the Board and the service code, class ID, instance ID, and service data size sent from the client will be stored in the SERVER_REQ structure. The received service data (request) will be stored in the specified service data storage buffer.

Specify the obtained message ID as the message ID for the SERVER_RES structure when sending the server response corresponding to the server request.

SERVER_RES Structure

This structure defines the format of server response explicit messages. This structure is used in the SCAN_SendServerExplicit() function.

Type	Name	Contents
DWORD	MessageID	Message ID Specifies the request message ID obtained by SCAN_ReceiveServerExplicit().
WORD	ServiceCode	Service code Specifies the service code sent as the response. For normal responses: Turns ON the leftmost bit (bit 15) of the service code obtained using SCAN_ReceiveServerExplicit and uses it as the response service code. For error responses: Specifies 0×94(148) as the service code for the error response.
WORD	DataLength	The amount of service data in bytes Setting range: 0x0 to 0x228 (0 to 552)
BYTE*	ServiceData	Provide the number of bytes of service data indicated by the DataLength parameter.

Sending a Server Explicit Message: SCAN_SendServerExplicit()

Application Range Online status

Function Sends a server response message.

Call Format BOOL SCAN_SendServerExplicit(DWORD *Handle*, SERVER_RES**Msg*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
SERVER_RES*	Msg	Buffer address where the response message is stored

Return Value

TRUE is returned if the send event registration was completed successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

The message ID, service code, service data size, and buffer address where the service data must be saved in the SERVER_RES structure. The data corresponding to the service to be sent as the response (service data) must be stored in the specified service data storage buffer.

Make sure that the value saved as MessageID in the SERVER_REQ structure when SCAN_ReceiveServerExplicit() is completed is saved as the message ID for the SERVER_RES structure.

4-7 Maintenance API Functions

4-7-1 Status Service API Functions

Reading Network Status: SCAN_GetNetworkStatus()

Application Range

Open status

Function

Gets the network (CAN) status.

Call Format

BOOL SCAN_GetNetworkStatus(DWORD *Handle*, WORD **Status*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD *	Status	Buffer address for receiving the status.

Return Value

TRUE is returned if the specified Board's network status was read successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

The following table shows the location and meaning of the network status flags. The flags are contained within 16 bits.

Bit	Code	Meaning
0	B00	CAN active
1	B01	Network frame detected
8	B08	Network power supply error
9	B09	Transmission timeout error
10	B10	Reception overwrite occurred
11	B11	Reception overload warning
12	B12	Transmission overload warning
14	B14	Bus off error
15	---	Not used

The status obtained using this function is not required for control, but is useful for determining whether or not a communications error originated in the network.

Reading Scanner Status: SCAN_GetScannerStatus()

Application Range Open status

Function Gets the scanner status.

Call Format BOOL SCAN_GetScannerStatus(DWORD *Handle*, DWORD **Status*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
DWORD *	Status	Buffer address for receiving the status.

Return Value TRUE is returned if the specified Board's scanner status was read successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description The following table shows the location and meaning of the scanner status flags. The flags are contained within 32 bits.

Bit	Code	Meaning
0	B00	Scanner function error occurred
1	B01	Master function error occurred
2	---	Not used
3	B03	Slave function error occurred
4	B04	Memory error
5	B05	Bus off error
6	B06	MAC ID duplication error
7	B07	Network power supply error
8	B08	Transmission timeout error
9 to 12	---	Not used
13	B13	Invalid message monitoring timer
14 and 15	---	Not used
16	B16	Online
17	B17	Scanning (I/O communications being executed)
18 to 20	---	Not used
21	B21	Slave function enabled mode
22	B22	Automatic setting of Slave scan type
23 to 30	---	Not used
31	B31	Error log in use

Monitor scanner status bit 0 to check if errors have occurred when using Master or Slave functions to perform I/O communications.

Bits 0 to 13 notify if errors have occurred. Bit 0 is an OR of bits 1 to 13.

Bit 1 turns ON if an error occurs with the Master function. Bit 1 is an OR of bits 0 and 2 of the Master function status obtained using SCAN_Get MasterModeStatus().

Bit 3 turns ON if an error occurs with the Slave function. Bit 3 is an OR of bits 2 and 3 of the Slave function status obtained using SCAN_Get SlaveModeStatus().

Reading Master Function Status: SCAN_GetMasterModeStatus()

Application Range Open status

Function Gets the Master function status.

Call Format

BOOL SCAN_GetMasterModeStatus(DWORD *Handle*, WORD **Status*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD *	Status	Buffer address for receiving the status.

Return Value

TRUE is returned if the specified Board's Master function status was read successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

The following table shows the location and meaning of the Master function status flags. The flags are contained in 16 bits.

Bit	Code	Meaning
0	B00	Verification error
1	---	Not used
2	B02	I/O communications error
3 to 14	---	Not used
15	B15	I/O communications in progress

Bit 15 indicates that the node has started I/O communications as the Master. Execute I/O data I/O processing with Slaves only after checking the status of each Slave using SCAN_GetSlaveDeviceStatus().

Check verification error details for each Slave using SCAN_Get SlaveDeviceStatus().

Bits 0 and 2 indicate Master function errors. When either bit 0 or 2 turn ON, bit 1 of the scanner status obtained using SCAN_GetScannerStatus() will turn ON.

Reading Slave Detailed Status: SCAN_GetSlaveDeviceStatus()

Application Range

Open status

Function

Gets the specified Slave's detailed status.

Call Format

BOOL SCAN_GetSlaveDeviceStatus(DWORD *Handle*, WORD *MacId*, WORD **Status*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	MacId	Node address for reading detailed Slave status. Setting range: 0x00 to 0x3F (0 to 63)
WORD *	Status	Buffer address for receiving status.

Return Value

TRUE is returned if the specified Slave's device status was read successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

The following table shows the location and meaning of the Slave device status flags.

Bit	Code	Meaning
0	B00	Error occurred
1	B01	Verification error
2	B02	Invalid Slave
3	B03	Vendor ID discrepancy

Bit	Code	Meaning
4	B04	Product type discrepancy
5	B05	Product code discrepancy
6	B06	Unsupported connection
7	B07	I/O size discrepancy
8	B08	Connection path discrepancy
10	B10	I/O communications error
15	B15	I/O communications in progress

Bit 0 is an OR of bits 1 and 10.

Bit 1 is an OR of bits 2 to 8. The details of the verification error are indicated by bits 2 to 8.

Check that bit 15 is ON before performing I/O processing with Slaves using the Master function.

Checking Slave Registration: SCAN_IsScanlistSlaveDeviceRegist()

Application Range Open status

Function Checks whether or not a Slave is registered in the scan list.

Call Format BOOL SCAN_IsScanlistSlaveDeviceRegist(DWORD *Handle*, WORD *MacId*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	MacId	Node address to check Setting range: 0x00 to 0x3F (0 to 63)

Return Value TRUE is returned if the specified Slave is registered in the scan list; FALSE is returned if an error occurred or the Slave is not registered in the scan list. Detailed error information can be read with the GetLastError() function.

Checking Slave Connection Established: SCAN_IsDeviceConnection()

Application Range Open status

Function Checks whether or not a connection has been established with a Slave.

Call Format BOOL SCAN_IsDeviceConnection(DWORD *Handle*, WORD *MacId*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD	MacId	Node address for which the connection is to be checked. Setting range: 0x00 to 0x3F (0 to 63)

Return Value TRUE is returned if a connection has been established successfully with the specified device. FALSE is returned if a connection has not been opened normally. Detailed error information can be read with the GetLastError() function.

Reading the Current Communications Cycle Time: SCAN_GetCycleTime()

Application Range Open status

Function Gets the present value of the communications cycle time.

Call Format BOOL SCAN_GetCycleTime(DWORD *Handle*, WORD **CycleTime*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD *	CycleTime	Buffer address for receiving the present value for the communications cycle time.

Return Value TRUE is returned if the communications cycle time was read successfully from the specified Board. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description The present value of the communications cycle time will be stored in ms.

Reading the Maximum Communications Cycle Time: SCAN_GetMaxCycleTime()

Application Range Open status

Function Gets the maximum value of the communications cycle time.

Call Format BOOL SCAN_GetMaxCycleTime(DWORD *Handle*, WORD **CycleTime*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD *	CycleTime	Buffer address for receiving the maximum communications cycle time.

Return Value TRUE is returned if the maximum communications cycle time was read successfully from the specified Board. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description The maximum value after the SCAN-ClearCycleTime() function is executed, power is turned ON, or the Board is reset is held as the maximum communications cycle time.

Reading the Minimum Communications Cycle Time: SCAN_GetMinCycleTime()

Application Range Open status

Function Gets the minimum value of the communications cycle time.

Call Format BOOL SCAN_GetMinCycleTime(DWORD *Handle*, WORD **CycleTime*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD *	CycleTime	Buffer address for receiving the minimum communications cycle time.

Return Value TRUE is returned if the minimum communications cycle time was read successfully from the specified Board. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description The minimum value after the SCAN_CycleTime function is executed, the power is turned ON, or the Board is reset is held in the minimum communications cycle time.

Clearing the Minimum and Maximum Communications Cycle Time: SCAN_ClearCycleTime()

Application Range Open status

Function

Clears the maximum and minimum communications cycle time values held in the Board.

Call Format

BOOL SCAN_ClearCycleTime(DWORD *Handle*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()

Return Value

TRUE is returned if the maximum and minimum communications cycle times were cleared successfully from the specified Board. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Reading Slave Function Status: SCAN_GetSlaveModeStatus()

Application Range

Open status

Function

Gets the Slave function status.

Call Format

BOOL SCAN_GetSlaveModeStatus(DWORD *Handle*, WORD **Status*, WORD **MacId*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
WORD*	Status	Buffer address for receiving the status.
WORD*	MacId	Buffer address for receiving the Master node address.

Return Value

TRUE is returned if the specified Board's Slave function status was read successfully. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

The following table shows the location and meaning of the Slave function status flags. The flags are contained in 16 bits.

Bit	Code	Meaning
0 and 1	---	Not used
2	B02	I/O communications error (Output data 1 and input data 1)
3	B03	I/O communications error (Output data 2 and input data 2)
4 to 11	---	Not used
12	B12	Connection 2 established
13	B13	Connection 1 established
14	B14	I/O communications in progress (Output data 2 and input data 2)
15	B15	I/O communications in progress (Output data 1 and input data 1)

The correct value for the Master node address is stored when bit 12 or bit 13 turns ON.

Bits 2 and 3 indicate Slave function errors. When either bit 2 or 3 turns ON, bit 3 of the scanner status obtained using SCAN_GetScannerStatus() turns ON.

Check that bits 14 or 15 is ON before performing I/O data I/O processing using the Slave function.

4-7-2 Error Log Access Service API Functions

ERROR_TIME Structure

This structure defines the format of the error time stamp data.

This structure is used within the ERROR_INFO structure and in the SCAN_GetErrorLog() function.

Type	Name	Contents (BCD)
BYTE	Second	Second when error occurred
BYTE	Minute	Minute when error occurred
BYTE	Hour	Hour when error occurred
BYTE	Day	Day when error occurred
BYTE	Month	Month when error occurred
BYTE	Year	Year when error occurred

ERROR_INFO Structure

This structure defines the format of the error log data.

This structure is used within the ERROR_LOG structure and in the SCAN_GetErrorLog() function.

Type	Name	Contents
WORD	ErrorCode	Error code
WORD	DetailCode	Detail code
ERROR_TIME	Time	Time when error occurred

ERROR_LOG Structure

This structure consolidates all of the error log data. It is used with the SCAN_GetErrorLog() function.

Type	Name	Contents
WORD	ErrorCount	Number of error records contained in the error log
ERROR_INFO	ErrorInfo[64]	Error log information

Reading an Error Log: SCAN_GetErrorLog()

Application Range

Open status

Function

Reads the error log from the specified Board.

Call Format

BOOL SCAN_GetErrorLog(DWORD *Handle*, ERROR_LOG **Log*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
ERROR_LOG *	Log	Buffer address for receiving the error log data.

Return Value

TRUE is returned if the error log was read successfully from the specified Board. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

Refer to *7-3 Error Log Function* for details on the error log function such as the meaning of error codes.

Error logs not saved to non-volatile memory are held only in open status. The error log is cleared when the Board is closed, the power turned OFF, or the Board is reset. Error logs saved to non-volatile memory can be read again even after the Board is re-opened or restarted.

Clearing an Error Log: SCAN_ClearErrorLog()

Application Range Open status

Function Clears the error log from the specified Board.

Call Format BOOL SCAN_ClearErrorLog(DWORD *Handle*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()

Return Value TRUE is returned if the error log data was successfully cleared from the specified Board. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description Clears the error log stored in non-volatile memory.

4-7-3 PC Watchdog Timer Service API Functions

Setting the PC WDT: SCAN_EnablePCWDTTimer()

Application Range Open status

Function Enables or disables the Board's PC watchdog timer management function.

Call Format BOOL SCAN_EnablePCWDTTimer(DWORD *Handle*, BOOL *Enable*, WORD *Timer*)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()
BOOL	Enable	PC watchdog timer setting TRUE: Enabled (PC-WDT is used.) FALSE: Disabled (PC-WDT is stopped.)
WORD	Timer	Monitoring time setting 0x0 to 0xFFFF (0 to 65535) (Monitoring time = Set value × 10 ms)

Return Value TRUE is returned if the processing was completed normally. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description Refer to *3-11 PC Watchdog Timer Management Function* for details on the PC watchdog timer management function.

Refreshing the PC WDT: SCAN_RefreshPCWDTTimer()

Application Range Open status

Function Refreshes the PC watchdog timer's timer value in the specified Board.

Call Format BOOL SCAN_RefreshPCWDTTimer(DWORD Handle)

Arguments

Type	Name	Contents
DWORD	Handle	Device handle obtained by SCAN_Open()

Return Value TRUE is returned if the processing was completed normally. FALSE is returned if an error occurred. Detailed error information can be read with the GetLastError() function.

Description

When using the PC watchdog timer management function, be sure to refresh the timer value with this function within the PC-WDT timer value. Remote I/O communications will stop if the set timer value elapses before the timer value is refreshed.

Refer to *3-11 PC Watchdog Timer Management Function* for details on the PC watchdog timer management function.

SECTION 5
Sample Programs

This section describes the sample programs that have been provided as reference when writing programs for the DeviceNet PCI Board.

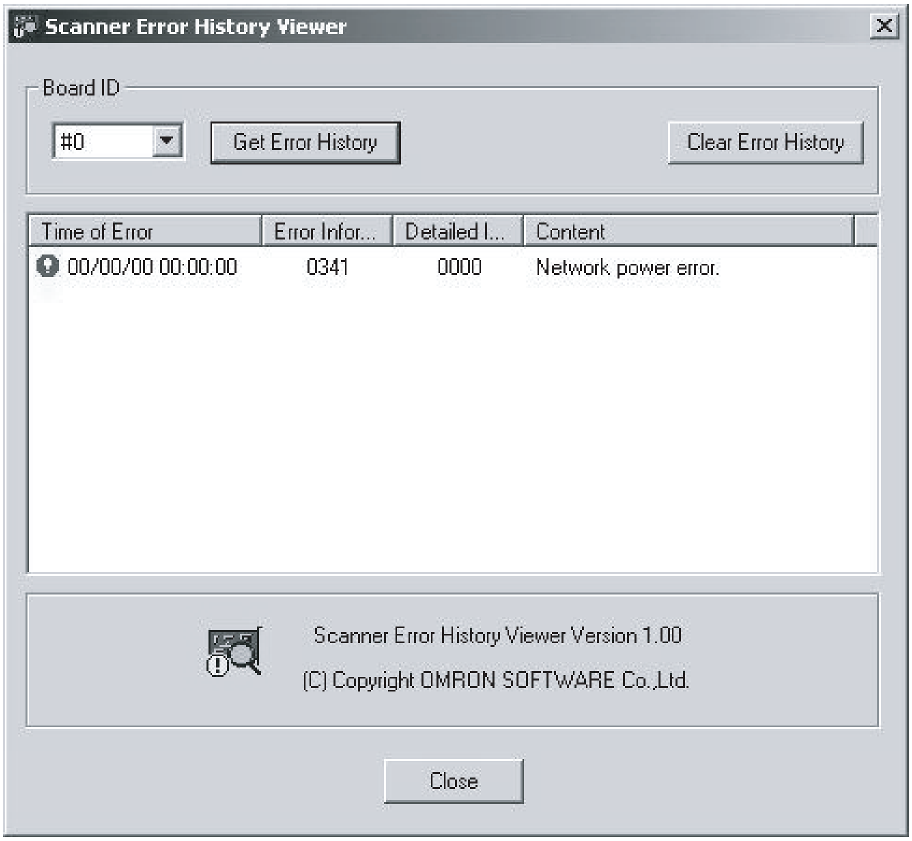
5-1 Sample

The following sample programs are extracted into the “Sample” directory when the Scanner SDK is installed.

Path	Function
Sample\BusDELog	This sample program displays the Scanner’s error log.
Sample\DemoToo	This sample program operates the Scanner function and executes remote I/O communications and message communications.

The sample programs can be started from the program folder specified when the Scanner SDK Software was installed.

The following diagram shows the starting window of the Scanner Error History Viewer.



Time of Error	Error Infor...	Detailed I...	Content
00/00/00 00:00:00	0341	0000	Network power error.

Scanner Error History Viewer Version 1.00

(C) Copyright OMRON SOFTWARE Co.,Ltd.

MicroSoft Visual C++ version 6.0 or higher is required to build the sample program.

The sample programs are provided to show how to use the API functions; they are not intended to be used as-is.

5-2 Using DeviceNet Scanner Demo

5-2-1 DeviceNet Scanner Demo Functions

The DeviceNet Scanner Demo program performs the following functions:

- Opens the Board.
- Closes the Board.
- Connects to network (goes online).

- Disconnects from the network (offline).
- Creates scan lists.
- Registers scan lists.
- Refreshes and monitors output and input data.
- Starts I/O communications.
- Stops I/O communications.
- Sends explicit client request messages and receives responses.

5-2-2 Usage Example for I/O Communications

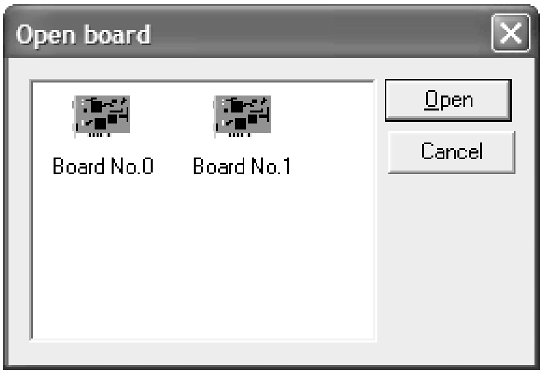
This section provides an example of performing I/O communications with Slaves using the DeviceNet Scanner Demo program.

Opening the Board

Use the following procedure to open the Board and prepare it for use.

1,2,3...

1. Select ***Open*** from the Board menu.
2. The following Open Board Window will be displayed.



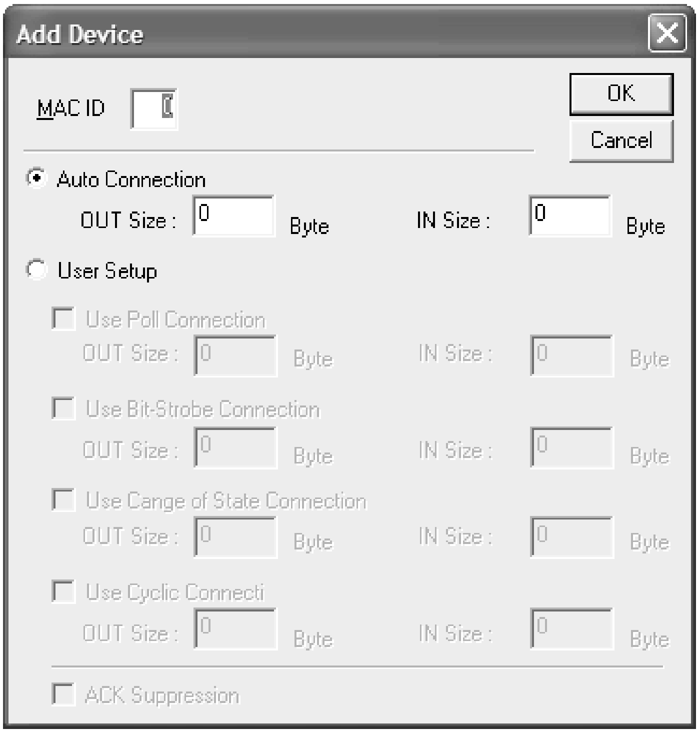
3. Select the Board to be used and click **Open**.
4. The specified Board will be opened and ready for use.
   The Board number (Board ID) will be displayed on the status bar.

Creating Scan Lists

Use the following procedure to create a scan list in the DeviceNet Scanner Demo program.

1,2,3...

1. Select ***Register to ScanList*** from the Edit menu.
2. The following Add Device Window for registering Slaves will be displayed.



MAC ID:

● Auto Connection
OUT Size : 0 Byte IN Size : 0 Byte

○ User Setup

□ Use Poll Connection
OUT Size : 0 Byte IN Size : 0 Byte

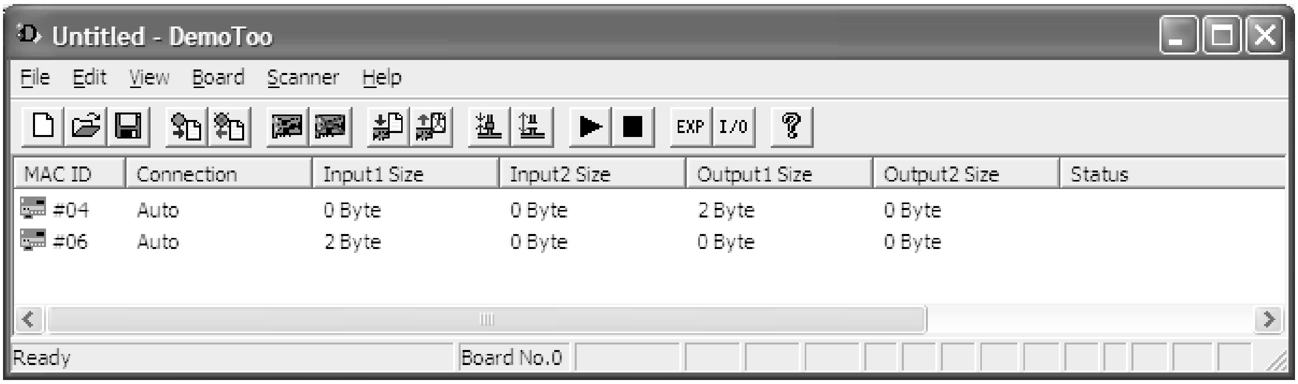
□ Use Bit-Strobe Connection
OUT Size : 0 Byte IN Size : 0 Byte

□ Use Cange of State Connection
OUT Size : 0 Byte IN Size : 0 Byte

□ Use Cyclic Connecti
OUT Size : 0 Byte IN Size : 0 Byte

□ ACK Suppression

3. Set the Slave node address (MAC ID) and I/O size (OUT Size and IN Size). Select ***User Setup*** to specify the I/O connection, and set the connection to be used and the I/O size.
4. Click **OK** to register the Slave in the main window.
5. Once all the Slaves have been registered, click **Cancel**, to display the created scan list in the main window, as shown below.



MAC ID	Connection	Input1 Size	Input2 Size	Output1 Size	Output2 Size	Status
#04	Auto	0 Byte	0 Byte	2 Byte	0 Byte	
#06	Auto	2 Byte	0 Byte	0 Byte	0 Byte	

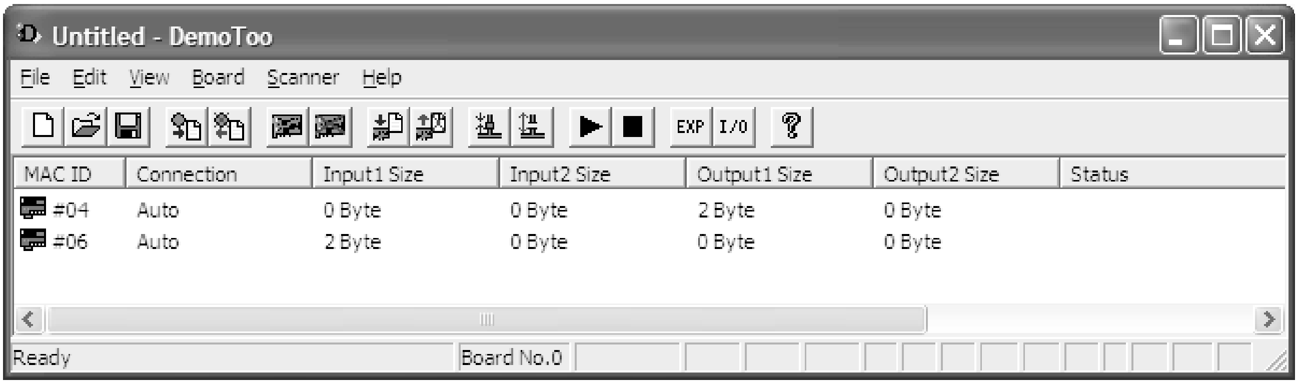
Ready Board No.0

Note The icons in the MAC ID column will be displayed in gray until the scan list is registered to the Board.

Registering Scan Lists Use the following procedure to register the created scan list to the Board.

1,2,3...

1. Select ***Set ScanList*** from the Board menu.
2. The icons in the MAC ID column will change from gray, as shown below, when the scan list is registered correctly.



MAC ID	Connection	Input1 Size	Input2 Size	Output1 Size	Output2 Size	Status
#04	Auto	0 Byte	0 Byte	2 Byte	0 Byte	
#06	Auto	2 Byte	0 Byte	0 Byte	0 Byte	

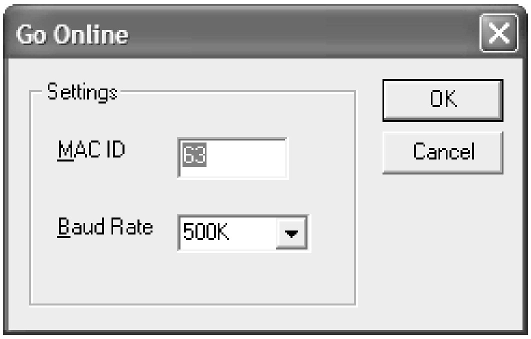
Ready Board No.0

Connecting to the Network

Use the following procedure to connect the Board to the DeviceNet network (i.e., go online).

1,2,3...

1. Select ***Online*** from the Scanner menu.
2. The following Go Online Window will be displayed.



MAC ID: 63

Baud Rate: 500K

3. Set the Board's node address (MAC ID) and baud rate, and click **OK**.
4. The Board will go online, i.e., will be connected to the DeviceNet network. The status bar will indicate the online status ("OL" at the far right) and the node address will be displayed next to the Board number.

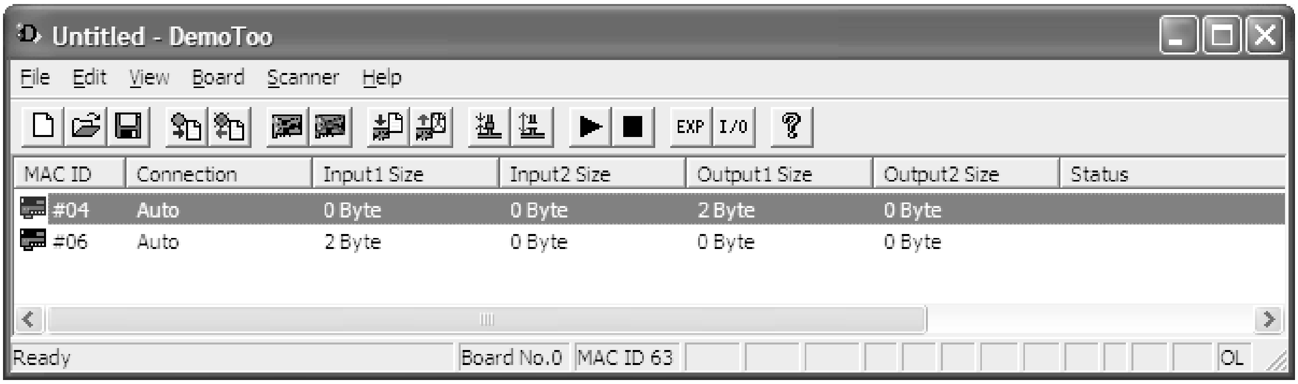
Note I/O communications have not started yet, even though the Board is online.

Setting Initial Output Data

Use the following procedure to set the initial output data before starting I/O communications.

1,2,3...

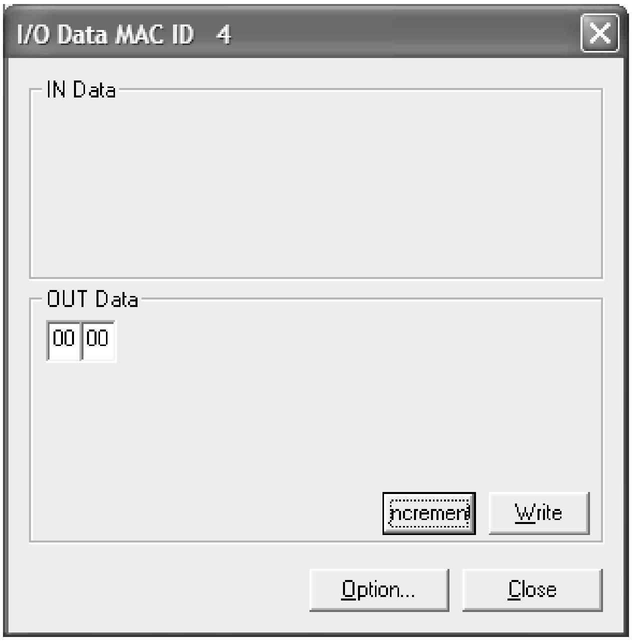
1. Select the Slave with the output data.



MAC ID	Connection	Input1 Size	Input2 Size	Output1 Size	Output2 Size	Status
#04	Auto	0 Byte	0 Byte	2 Byte	0 Byte	
#06	Auto	2 Byte	0 Byte	0 Byte	0 Byte	

Ready Board No.0 MAC ID 63 OL

2. Select ***I/O Data*** from the Scanner menu.
3. The following window will be displayed.



4. Enter the output data to be set and click **Write**.

Note The output data will not be sent yet.

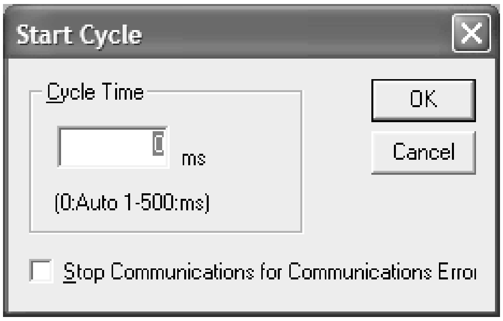
5. Perform the above operation for all Slaves that have output data.

Starting I/O Communications

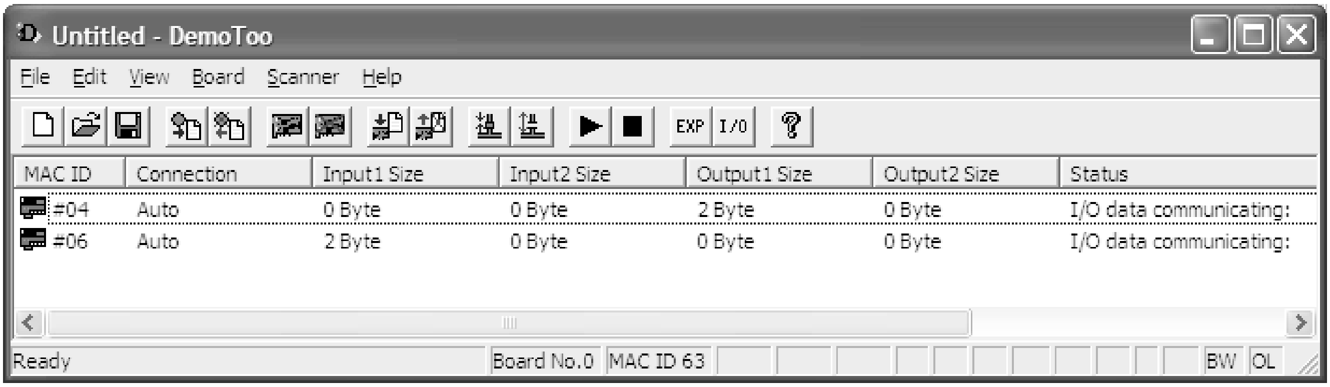
Use the following procedure to start I/O communications with Slaves.

1,2,3...

1. Select ***Start Cycle*** from the Scanner menu.
2. The following Start Cycle Window will be displayed.



3. Set the communications cycle time (Cycle Time) and click **OK**. Specify 0 to automatically set the communications cycle time.
4. I/O communications will start. The status will be displayed in the Status column for each Slave, as shown in the following diagram, if I/O communications have been started correctly.



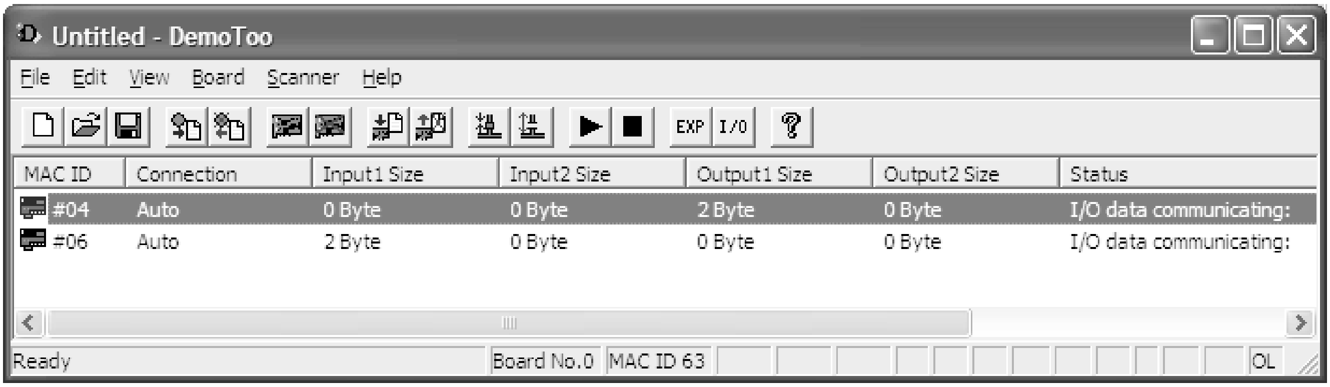
MAC ID	Connection	Input1 Size	Input2 Size	Output1 Size	Output2 Size	Status
#04	Auto	0 Byte	0 Byte	2 Byte	0 Byte	I/O data communicating:
#06	Auto	2 Byte	0 Byte	0 Byte	0 Byte	I/O data communicating:

Note It takes several seconds for a connection to be established (and until the status is displayed).

Refreshing and Monitoring Output and Input Data

Use the following procedure to refresh output data and monitor input data for each Slave during I/O communications.

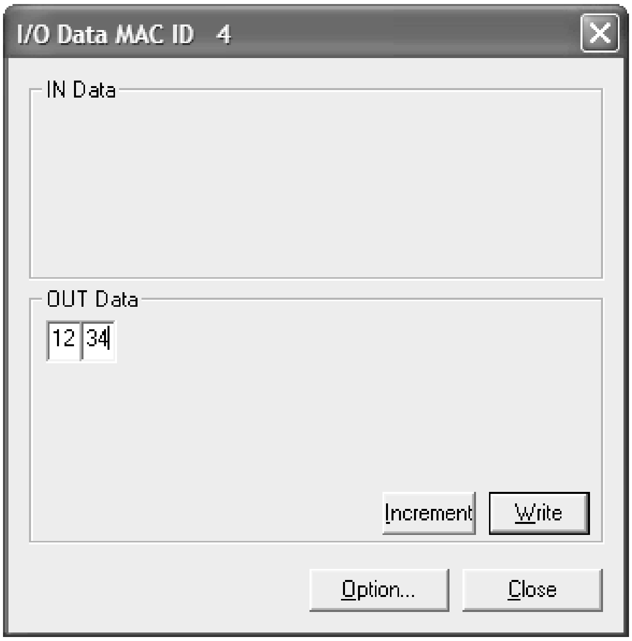
1,2,3... 1. Select the desired Slave.



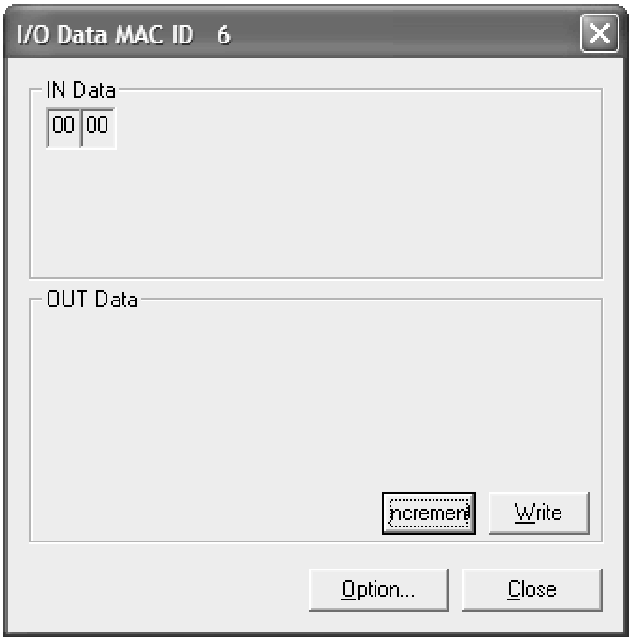
MAC ID	Connection	Input1 Size	Input2 Size	Output1 Size	Output2 Size	Status
#04	Auto	0 Byte	0 Byte	2 Byte	0 Byte	I/O data communicating:
#06	Auto	2 Byte	0 Byte	0 Byte	0 Byte	I/O data communicating:

2. Select ***I/O Data*** from the Scanner menu.
3. The I/O Data Window will be displayed, as shown below.

- Output Data



- Input Data



The input data will be displayed.

4. Multiple I/O data setting and monitoring windows can be displayed at the same time.

Automatically Refreshing Output Data

Use the following procedure to automatically refresh output data to be sent to Slaves.

1,2,3...

1. Display the I/O Data Window for the corresponding Slave.
2. Click **Increment**.
3. The current output data will be automatically incremented and sent.

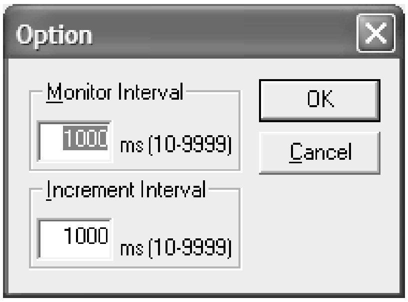
4. Click **Stop** to stop automatic refresh.

Changing Input Data Monitor Interval and Output Data Automatic Refresh Interval

Use the following procedure to change the monitor interval for input data and the automatic refresh interval for output data.

1,2,3...

1. Display the I/O Data Window for the corresponding Slave.
2. Click **Option**.
3. The following Option Window will be displayed.



Option

Monitor Interval

1000 ms (10-9999)

Increment Interval

1000 ms (10-9999)

4. Set the Monitor Interval or Increment Interval timer value, and click **OK**.
5. The input data monitoring and output data automatic refresh will be performed at the set intervals.

Stopping I/O Communications

Use the following procedure to stop I/O communications with Slaves.

1,2,3...

1. Select ***Stop Cycle*** from the Scanner menu.
2. I/O communications with Slaves will be stopped.

Disconnecting from the Network

Use the following procedure to disconnect from the DeviceNet network (i.e., go offline).

1,2,3...

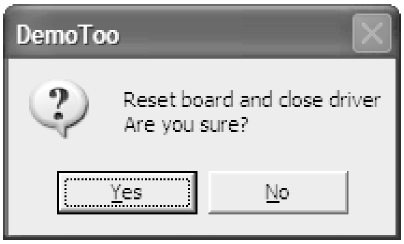
1. Select ***Offline*** from the Scanner menu.
2. The PCI Board will go offline, i.e., be disconnected from the DeviceNet network.
   The status bar will no longer show “OL” at the far right and the node address will no longer be displayed next to the Board number.

Closing Boards

Use the following procedure to close the Board.

1,2,3...

1. Select ***Close*** from the Board menu.
2. The following confirmation message will be displayed.



DemoToo

Reset board and close driver
Are you sure?

3. Click **Yes** to close the Board.

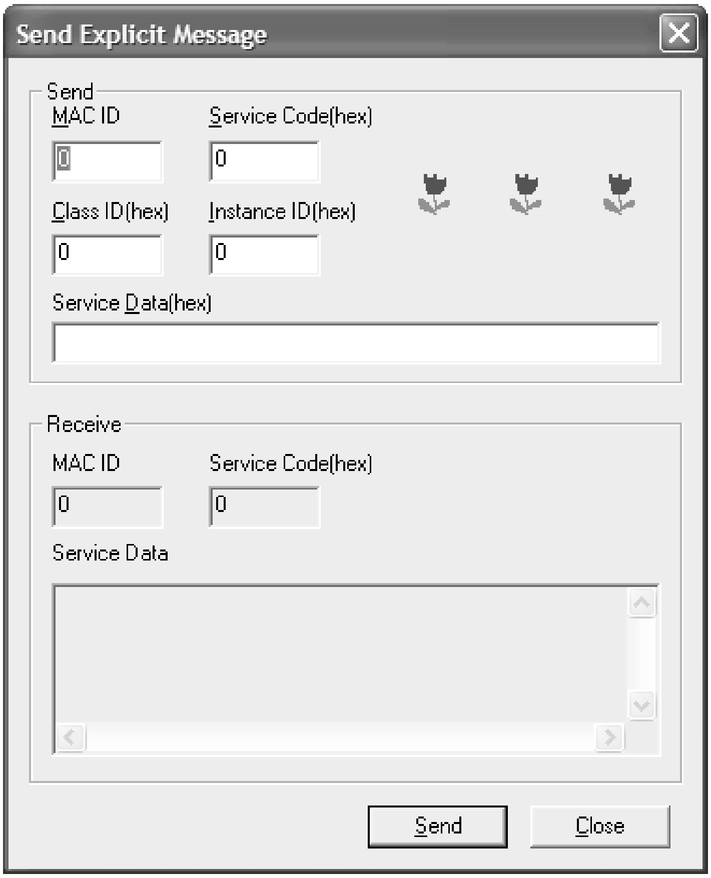
5-2-3 Usage Example for Explicit Message Communications

The DeviceNet Scanner Demo program has explicit message client functions. Explicit request messages can be sent to remote nodes and responses received.

Use the following procedure to send explicit request messages and receive explicit response messages.

1,2,3...

1. Open the Board and connect online to the network.
   Explicit request messages can be sent regardless if I/O communications are being performed or are stopped.
2. Select ***Send Explicit Message*** from the Scanner menu.
3. The following Send Explicit Message Window will be displayed.



Send Explicit Message

Send

MAC ID: 0

Service Code(hex): 0

Class ID(hex): 0

Instance ID(hex): 0

Service Data(hex):

Receive

MAC ID: 0

Service Code(hex): 0

Service Data

4. Set the destination node address (MAC ID) and explicit request message parameters, and click **Send**.
5. The explicit request message will be sent and the response data displayed.
6. Click **Close** to finish sending explicit request messages.

SECTION 6
Communications Timing

This section describes communications timing in remote I/O communications and message communications.

6-1 Remote I/O Communications Characteristics

This section describes the characteristics of remote I/O communications when a DeviceNet PCI Board is used with OMRON Slaves. Use this section for reference when planning operations that require precise I/O timing.

The equations provided here are valid under the following conditions:

1,2,3...

1. All of the required Slaves are participating in communications.
2. No errors have occurred in any of the nodes in the network.
3. There are no I/O refreshing requests or command requests from the computer to the board.

Note The values provided by these equations may not be accurate if the conditions described above are not satisfied.

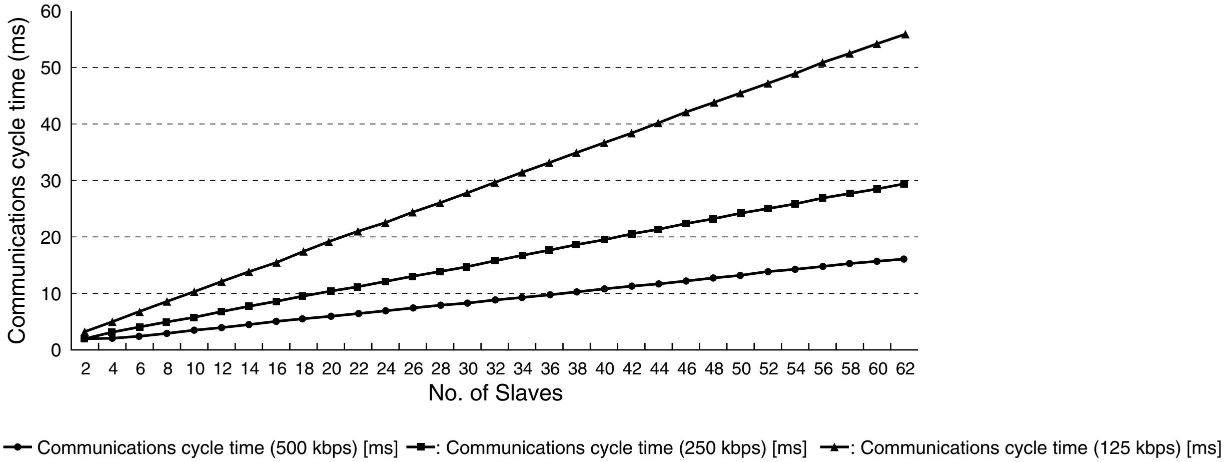
6-1-1 Communications Cycle Time

The communications cycle time is the time from the completion of a Slave's remote I/O communications processing until remote I/O communications with the same Slave are processed again. The communications cycle time is used to calculate the maximum I/O response time.

The communications cycle time depends on a variety of factors such as the number of Masters in the Network and on whether or not message communications are being performed. The following explanation is for a network with one Master. For networks with two or more Masters, refer to *6-1-2 More than One Master in Network*.

Communications Cycle Time Graph

The following graph plots the communications cycle time against the number of Slaves, for a mixture of 16-output point Slaves and 16-input point Slaves. Bit-strobe input and poll connection output are used.



Equations for Calculating Communications Cycle Time

Use the equations shown below to calculate the communications cycle time (T_{RM}) for a network with one Master. (The minimum communications cycle time is actually 2 ms even if the result of this calculation is less than 2 ms.)

$$
\begin{aligned}
T_{RM} = {} & \Sigma \text{ (Communications time per Slave)} \\
& + \text{High-density Unit processing time} \\
& + \text{Explicit message processing time} \\
& + \text{COS/Cyclic connection communications time [ms]} \\
& + 0.01 \times N + 1.0 \text{ [ms]}
\end{aligned}
$$

Communications Time Per Slave:

This is the communications time required for a single Slave (see below). "Σ (Communications time per Slave)" represents the total of the "Communications time per Slave" for all the Slaves in the network.

High-density Unit Processing Time:

3.5 [ms]

This is added if there are any Slaves in the network that use at least 8 bytes for input, output, or both.

Explicit Message Processing Time:

$(0.11 \times T_B) \times n$ [ms]

Only added when explicit communications (transmission or reception) are performed.

n: Number of explicit messages that occur at the same time during one CPU Unit cycle (including transmission and reception).

T_B = Baud rate factor

($T_B = 2$ for 500 kbps, $T_B = 4$ for 250 kbps, and $T_B = 8$ for 125 kbps)

COS/Cyclic Connection Communications Time [ms]:

$\{(0.05 + 0.008 \times S) \times T_B\} \times n$ [ms]

S: Total input and output size (in bytes) for COS/Cyclic connection.

$T_B = 2$ for 500 kbps, $T_B = 4$ for 250 kbps, and $T_B = 8$ for 125 kbps

n: Number of nodes for COS/Cyclic connections that are used at the same time in one communications cycle.

N: Number of Slaves

Communications Time/ Slave

The communications time per Slave is the time required for communications with a single Slave. Use the following equations to calculate the communications time/Slave (T_{RT}) for each kind of Slave.

Output Slaves with less than 8 Bytes of Output

$T_{RT} = 0.016 \times T_B \times S_{OUT1} + 0.11 \times T_B + 0.07$ [ms]

S_{OUT1}: The number of Output Slave output words

T_B: Baud rate factor

($T_B = 2$ for 500 kbps, $T_B = 4$ for 250 kbps, and $T_B = 8$ for 125 kbps)

Input Slaves with less than 8 Bytes of Input

$T_{RT} = 0.016 \times T_B \times S_{IN1} + 0.06 \times T_B + 0.05$ [ms]

S_{IN1}: The number of Input Slave input words

T_B: Baud rate factor

($T_B = 2$ for 500 kbps, $T_B = 4$ for 250 kbps, and $T_B = 8$ for 125 kbps)

Mixed I/O Slaves with less than 8 Bytes each of Input and Output

$T_{RT} = 0.016 \times T_B \times (S_{OUT2} + S_{IN2}) + 0.11 \times T_B + 0.07$ [ms]

S_{OUT2}: The number of Mixed I/O Slave output words

S_{IN2}: The number of Mixed I/O Slave input words

T_B: Baud rate factor

($T_B = 2$ for 500 kbps, $T_B = 4$ for 250 kbps, and $T_B = 8$ for 125 kbps)

Slaves with more than 8 Bytes of Input or Output or both

$T_{RT} = T_{OH} + T_{BYTE\text{-}IN} \times B_{IN} + T_{BYTE\text{-}OUT} \times B_{OUT}$ [ms]

T_{OH} : Protocol overhead
$T_{BYTE\text{-}IN}$: The input byte transmission time
B_{IN} : The number of input bytes
$T_{BYTE\text{-}OUT}$: The output byte transmission time
B_{OUT} : The number of output bytes

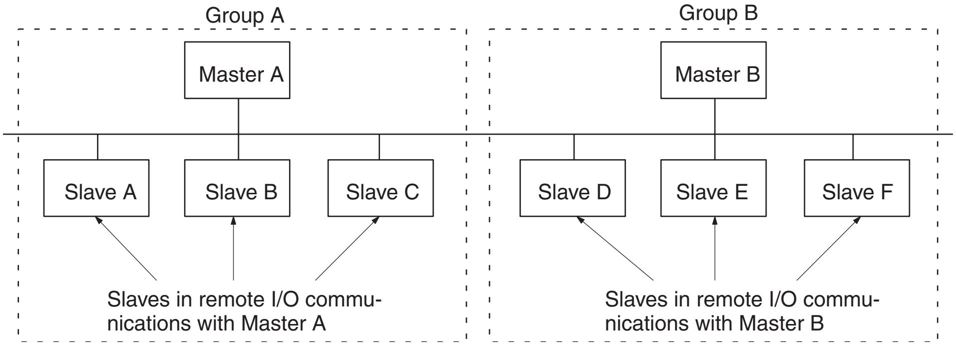
Baud rate	T_{OH}	$T_{BYTE\text{-}IN}$	$T_{BYTE\text{-}OUT}$
500 kbps	0.306 ms	0.040 ms	0.036 ms
250 kbps	0.542 ms	0.073 ms	0.069 ms
125 kbps	1.014 ms	0.139 ms	0.135 ms

For Input Slaves use $B_{OUT} = 0$, and for Output Slaves use $B_{IN} = 0$.

6-1-2 More than One Master in Network

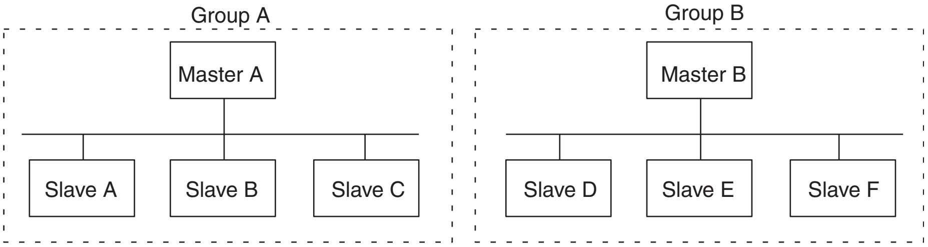
This section explains how to calculate the remote I/O communications cycle time (T_{RM}) when there is more than one Master in the Network. In this example there are two Master Units in the Network.

First, the Network is divided into two groups: Master A and the Slaves in remote I/O communications with it and Master B and the Slaves in remote I/O communications with it.


Group A
Group B
Master A
Master B
Slave A
Slave B
Slave C
Slave D
Slave E
Slave F
Slaves in remote I/O communications with Master A
Slaves in remote I/O communications with Master B

Note The Slaves are physically separated into two groups for clarity in this diagram, but the actual physical positions in the Network are irrelevant.

Next, we refer to the equations introduced in *6-1-1 Communications Cycle Time* and calculate the communications cycle time for each group as if they were separate Networks.



Group A communications cycle time: $T_{RM\text{-}A}$

Group B communications cycle time: $T_{RM\text{-}B}$

In a Network with two Masters, the communications cycle time for the entire Network is the sum of the communications cycle times for the two groups.

$$T_{RM} = T_{RM\text{-}A} + T_{RM\text{-}B}$$

Although this example shows only two Masters in the Network, the total communications cycle time for any Network can be calculated by dividing it into groups and adding the communications cycle times of all groups.

SECTION 7
Error Processing

This section describes troubleshooting and error processing procedures needed to identify and correct errors that can occur during DeviceNet PCI Board operation.

7-1 LED Indicators and Error Processing

The DeviceNet PCI Board have an MS (Module Status) indicator that indicates the status of the Board itself and an NS (Network Status) indicator that indicates the status of the Network. These indicators show when an error has occurred and what type of error it is.

This section explains the meaning of the LED indicators and the processing required when an error has occurred. This explanation assumes that the Board has been installed and set up properly.

Indicator Status during Normal Operation

The following table shows the status of the MS and NS indicators during normal operation.

Indicator status		Network/Board status	Comments
MS	NS		
OFF	OFF	Waiting for initialization	Boot program initialization processing is being executed.
Flashing (green)	OFF	Waiting for start of scanner program	Scanner firmware initialization processing is being executed.
ON (green)	OFF	MAC ID (node address) duplication check in progress	Waiting for online request or the request has been received and a node address duplication check is in progress between the Board and other nodes in the network.
ON (green)	Flashing (green)	Remote I/O communications stopped and message communications not established.	Communications have not been established with other nodes even though the Board is online. With message communications, this indicates that the local node has not sent a message to another node and that a message has not been received from another node.
ON (green)	ON (green)	Remote I/O or message communications in progress.	Indicates normal communications when remote I/O and/or message communications are active.

Indicator Status when an Error has Occurred

The following table shows the status of the MS and NS indicators when an error has occurred and lists probable causes for the errors.

Indicator status		Error	Probable cause and remedy
MS	NS		
Flashing (red)	No change	Hardware error (EEPROM)	An EEPROM error occurred. Replace the Board.
Flashing (red)	OFF	PC watchdog timer error	The Board's PC watchdog timer management function detected a PC watchdog timer error. (The computer application has stopped.) Restart the computer application or the computer itself.
ON (red)	OFF	Hardware error (Watchdog timer or RAM)	One of the following hardware errors occurred: • Board watchdog timer error • RAM error Replace the Board
No change	ON (red)	Node address duplication	The Board's node address has been set on another node. Change the node address settings to eliminate the duplication. Restart the computer or reset the Board.
		Bus off detected	Communications were stopped because a large number of data errors occurred. • Check the communications baud rate settings in all of the nodes. • Check that the cable lengths (trunk and drop lines) are within specifications. • Check for loose or broken cables. • Check that there are Terminating Resistors at each end of the trunk line and nowhere else in the network. • Check for excessive noise.

Indicator status		Error	Probable cause and remedy
MS	NS		
No change	OFF	Send error: Network power supply error	The communications power supply isn't being supplied properly. Check the network power supply and network cable wiring.
		Send error: Transmission timeout	A transmission couldn't be completed successfully for one of the following reasons: • There are no Slaves in the network. • There is another Master in the Network. • There is an error in the CAN controller. • The Master and Slave baud rate settings don't agree. Check the following: • Check the communications baud rate settings in all of the nodes. • Check that the cable lengths (trunk and drop lines) are within specifications. • Check for loose or broken cables. • Check that there are Terminating Resistors at each end of the trunk line and nowhere else in the network. • Check for excessive noise.
No change	Flashing (red)	Verification error: Slave doesn't exist	A Slave registered in the scan list does not exist in the network. Check the following: • Check the communications baud rate settings in all of the nodes. • Check that the cable lengths (trunk and drop lines) are within specifications. • Check for loose or broken cables. • Check that there are Terminating Resistors at each end of the trunk line and nowhere else in the network. • Check for excessive noise.
		Unsupported Slave	A Slave is connected that has an I/O size exceeding 200 bytes. Remove Slaves from the network if they have an I/O size greater than 200 bytes.
		Verification error: Slave I/O size mismatch	The I/O size of a Slave registered in the scan list doesn't match the actual Slave in the network. Check the Slave and create the scan list again.
		I/O Communications error	A timeout occurred in I/O communications. Check the following: • Check the communications baud rate settings in all of the nodes. • Check that the cable lengths (trunk and drop lines) are within specifications. • Check for loose or broken cables. • Check that there are Terminating Resistors at each end of the trunk line and nowhere else in the network. • Check for excessive noise.
OFF	OFF	System error	Replace the Board.

7-2 Identifying Errors Detected by Functions

This section lists the errors that can be identified using the Board's Master API functions. The return value returned by API functions indicates when an error has occurred during execution of the function. (Refer to *Checking for Errors with Function Return Values* on page 45)

Error Codes

All of the Board's Master API functions are bool-type functions, so FALSE is returned as the return value when an error has occurred during execution. When FALSE is returned, the GetLastError() function can be used to read more detailed error information (the error code.)

The following table lists the error codes and the and the probable causes for those errors.

Note Refer to *Checking for Errors with Function Return Values* on page 45 for information on how to check errors using error codes.

Error code	Value	Likely cause and remedy
SCAN_INVALID_HANDLE	0x20000001	The driver handle is invalid. Specify a valid driver handle and execute the function again.
SCAN_ALREADY_USED	0x20000002	The specified driver is already being used by another application. Execute the function again after the other application ends.
SCAN_ERROR_DRIVER_FUNCTION	0x20000003	An error occurred in the driver function. Check the parameters and execute the function again.
SCAN_NOT_EXIST_DEVICE	0x20000004	There isn't a Board with the specified Board ID. Specify the Board ID set on a Board in the computer.
SCAN_TIMEOUT	0x20000005	The command timed out. Reset the Board and execute the function again.
SCAN_NOT_MODULE_PATH	0x20000006	The communications module path cannot be found. Install again.
SCAN_NOT_EXIST_MODULE	0x20000007	The communications module cannot be found. Install again.
SCAN_NOT_ALLOCATE_MEMORY	0x20000008	Memory cannot be allocated. Close other applications and execute the function again.
SCAN_NOT_OPEN_DRIVER	0x20000009	The driver cannot be opened. Check that the driver is installed properly and execute the function again.
SCAN_NOT_CREATE_EVENT	0x2000000A	The interrupt event cannot be created.
SCAN_INVALID_PARAMETER	0x2000000B	A parameter is invalid. Specify the parameter correctly and execute the function again.
SCAN_NOT_EXECUTECOMMAND	0x2000000C	An error occurred during execution of the command. Check the parameters and execute the function again.
SCAN_NOT_DOWNLOAD	0x2000000D	An error occurred while downloading the communications module. Check that there isn't a problem with the hardware and execute the function again.
SCAN_NOT_ENOUGH_BUFFER	0x20001000	There isn't enough buffer space to store the data. Increase the size of the buffer.
SCAN_RES_NOT_RECEIVED	0x20001001	The client response has not been received.
SCAN_REQ_NOT_RECEIVED	0x20001002	The server request has not been received.
SCAN_INVALID_REGISTHANDLE	0x20001003	The window or thread handle being registered is invalid. Specify a valid handle.
SCAN_INVALID_REGISTMSG	0x20001004	The notification message being registered is invalid. Specify a valid message.
SCAN_NOT_EVENTDATA	0x20001005	There is no event data.
SCAN_NOT_EVENT	0x20001006	There isn't such an event.
SCAN_FILE_IO_ERROR	0x20001007	An error occurred during file access. Check that the file exists and isn't corrupted and then execute the function again.
SCAN_INVALID_CMD	0x20010010	The command code is invalid. Specify a valid code.
SCAN_SYSTEM_BUSY	0x20010011	The system is busy.
SCAN_OFF_LINE	0x20010020	The function can't be executed because the Board is offline. Switch online and execute the function again.
SCAN_PROCESSING_ON_LINE	0x20010021	The function can't be executed because the connection is initializing. Wait a moment and execute the function again.
SCAN_ON_LINE	0x20010022	The function can't be executed because the Board is online. Switch offline and execute the function again.
SCAN_SCANNING	0x20010023	The function can't be executed because scanning is in progress. Stop scanning and execute the function again.
SCAN_NOT_SCANNING	0x20010024	The function can't be executed because scanning is stopped. Start scanning and execute the function again.

Error code	Value	Likely cause and remedy
SCAN_AUTO_SCANNING	0x20010025	The function can't be executed because the Board is operating in disabled mode. Switch to enabled mode and execute the function again.
SCAN_FIXED_SCANNING	0x20010026	The Board is already operating in disabled mode.
SCAN_BUS_OFF	0x20010029	The function can't be executed because a bus off error occurred. Clear the error and execute the function again.
SCAN_CONNECTED	0x2001002A	The connection is already established.
SCAN_DISCONNECTED	0x2001002B	The connection is already disconnected.
SCAN_NO_NET_POWER	0x2001002C	The network power supply is not being supplied. Connect the network power supply and execute the function again.
SCAN_TX_TIME_OUT	0x2001002D	A timeout occurred in the transmission. Check that the destination node is connected and the network isn't overloaded with high-priority messages. Wait a moment and execute the function again.
SCAN_DUP_MAC_ERROR	0x2001002E	The specified node address is duplicated on another device. Specify another node address.
SCAN_INVALID_BAUD_RATE	0x20010040	The communications baud rate is invalid. Specify a valid baud rate.
SCAN_INVALID_MAC_ID	0x20010041	The node address is invalid. Specify a valid node address.
SCAN_INVALID_SCAN_TYPE	0x20010042	The scan type is invalid. Specify a valid scan type and execute the function again.
SCAN_INVALID_IO_SIZE	0x20010043	The data size is invalid. Specify a valid data size and execute the function again.
SCAN_INVALID_CLASS_ID	0x20010044	The specified class ID is not registered.
SCAN_INVALID_INSTANCE_ID	0x20010045	The instance ID is invalid. Specify a valid instance ID.
SCAN_INVALID_ATTRIBUTE_ID	0x20010046	The attribute ID is invalid. Specify a valid attribute ID.
SCAN_INVALID_SELF_SLAVE	0x20010047	An effective Slave scan list is not registered. Register in the Slave scan list.
SCAN_TOO_MANY_CLASS	0x20010048	Too many class IDs have been registered.
SCAN_INVALID_SCAN_LIST	0x20010049	There are no devices registered in the scan list. Register the devices.
SCAN_NO_COS_CNXN	0x20010060	A COS connection has not been established. Establish the connection and execute the function again.
SCAN_NO_EMC_CNXN	0x20010061	An explicit message connection has not been established. Establish the connection and execute the function again.
SCAN_EMC_CNXN_TIME_OUT	0x20010062	A timeout occurred when establishing the explicit message connection. Check the target of the connection and execute the function again.
SCAN_SND_MSG_TOO_LONG	0x20010063	The transmission data size is too long.
SCAN_RSP_MSG_TOO_LONG	0x20010064	The reception data size is too long.
SCAN_DEST_DEVICE_OVERFLOW	0x20010065	The transmission buffer overflowed. Allow a little more time between transmissions and execute the function again.
SCAN_MEMORY_ACCESS_CONFLICT	0x20010080	The I/O allocations overlap. Set the I/O allocations again to eliminate the overlap.
SCAN_MEMORY_ERROR	0x20010081	An EEPROM write error occurred.
SCAN_SUM_CHECK_ERROR	0x20010082	An checksum error occurred.

7-3 Error Log Function

The DeviceNet PCI Board is equipped with an error log function that can store up to 64 error records on errors that occur. The records show the error code and time of occurrence.

The error log data can be read from the Board with the SCAN_GetErrorLog() function. (See page 101 for details.) The error log can be cleared with the SCAN_ClearErrorLog() function. (See page 102 for details.)

Note

1. Important error data in the error log is saved to non-volatile memory. Error data not saved to non-volatile memory is held only while the Board is open. All error data not saved to non-volatile memory is cleared when the Board is closed, the power turned OFF, or the Board reset.
2. Error data saved to non-volatile memory can be read once the Board is opened or started again.
3. Error logs saved to non-volatile memory can also be read with the Scanner Error History Viewer utility that is included with the Board's software.

Error Log Data

Each error log record has the following structure. The Board's error log can store up to 64 error records. If an error occurs when the error log is full, the oldest record is erased to make room for the new record.

Variable name	Data type	Contents (BCD)
ErrCode	WORD	Error code
DetailCode	WORD	Detail code
Second	BYTE	Second when error occurred
Minute	BYTE	Minute when error occurred
Hour	BYTE	Hour when error occurred
Day	BYTE	Day when error occurred
Month	BYTE	Month when error occurred
Year	BYTE	Year when error occurred

Note The error log data is defined in the ERROR_INFO structure. (See page 101 for details.)

Error Code Table

The following table shows the error codes that can be recorded by the error log function and the corresponding errors. Refer to *7-1 LED Indicators and Error Processing* for details on error processing.

Error code (Hex)	Error	Detailed information		EEPROM storage
		1st byte (Offset = 0002h)	2nd byte (Offset = 0003h)	
0001	PC watchdog timer error A timeout occurred in the PC_WDT monitoring function.	00 Hex	00 Hex	Yes
0101	Can't send response message when offline.	80 Hex	Bits 0 to 5: Client device MAC ID Bit 6: OFF Bit 7: ON	No
0106	Can't send response message because MAC ID duplication error occurred.	80 Hex	Bits 0 to 5: Client device MAC ID Bit 6: OFF Bit 7: ON	No
0107	Response message transmission failed because connection is not established yet.	80 Hex	Bits 0 to 5: Client device MAC ID Bit 6: OFF Bit 7: ON	No
0109	Destination node's buffer full	80 Hex	Bits 0 to 5: Client device MAC ID Bit 6: OFF Bit 7: ON	No

Error code (Hex)	Error	Detailed information		EEPROM storage
		1st byte (Offset = 0002h)	2nd byte (Offset = 0003h)	
0111	Service data too long	80 Hex	Bits 0 to 5: Client device MAC ID Bit 6: OFF Bit 7: ON	No
0117	Internal buffer full	00 Hex	Bits 0 to 5: Server device MAC ID Bit 6: OFF Bit 7: ON	No
0112	Response message discarded because the message ID is invalid	80 Hex	Bits 0 to 5: Client device MAC ID Bit 6: OFF Bit 7: ON	No
0211	MAC ID duplication error	00 Hex	Local MAC ID	No
021A	Setting table logic error	00 Hex	0A Hex: Scan list 0B Hex: Slave scan list 0C Hex: Message monitoring timer list	Yes
0340	Bus off error	00 Hex	00 Hex	No
0341	Network power supply error	00 Hex	00 Hex	No
0342	Transmission timeout error	00 Hex	00 Hex	No
0343	I/O configuration error	03 Hex Unsupported Slave	Unsupported Slave’s MAC ID	No
0344	Verification error	01 Hex: Non-existent Slave (including use of the local MAC ID) 02 Hex: Invalid vendor ID 03 Hex: Invalid product type 04 Hex: Invalid product code 05 Hex: Unsupported connection 06 Hex: I/O size mismatch 07 Hex: Invalid connection path	Slave’s MAC ID	No
0345	I/O communications error	01 Hex: Master function 02 Hex: Slave function	With Master function: Slave’s MAC ID With Slave function: Master’s MAC ID	No
0346	Scanning stopped due to I/O communications error	01 Hex: I/O communications error 02 Hex: Network power supply error 03 Hex: Transmission timeout	I/O communications error: Faulty Slave’s MAC ID Network power supply error: Local MAC ID Transmission timeout: Local MAC ID	No

Error code (Hex)	Error	Detailed information		EEPROM storage
		1st byte (Offset = 0002h)	2nd byte (Offset = 0003h)	
0348	Response message discarded because a new request was received	80 Hex	Bits 0 to 5: Client device MAC ID Bit 6: OFF Bit 7: ON	No
0601	System error	Undefined		Yes
0602	Memory error	01 Hex: Read error 02 Hex: Write error	06 Hex: Error log 09 Hex: Identity information 0A Hex: Scan list 0B Hex: Slave scan list 0C Hex: Message monitoring timer list 0D Hex: Communications cycle time set value	Yes (see note)

Note The error record will not be recorded in EEPROM if the memory error occurred in the error log area of EEPROM.

Revision History

A manual revision code appears as a suffix to the catalog number on the front cover of the manual.

Cat. No. W381-E1-03

Revision code

The following table outlines the changes made to the manual during each revision. Page numbers refer to the previous version.

Revision code	Date	Revised content
1	October 2000	Original production
02	July 2005	Revisions were made throughout the manual to add information on the DeviceNet Scanner SDK and make accompanying changes.
03	September 2013	• Information was added for Windows 7 support. • Changes were made for consistency with information in other manuals.

OMRON Corporation **Industrial Automation Company**
Tokyo, JAPAN

Contact: www.ia.omron.com

Regional Headquarters

OMRON EUROPE B.V.
Wegalaan 67-69-2132 JD Hoofddorp
The Netherlands
Tel: (31)2356-81-300/Fax: (31)2356-81-388

OMRON ELECTRONICS LLC
One Commerce Drive Schaumburg,
IL 60173-5302 U.S.A.
Tel: (1) 847-843-7900/Fax: (1) 847-843-7787

OMRON ASIA PACIFIC PTE. LTD.
No. 438A Alexandra Road # 05-05/08 (Lobby 2),
Alexandra Technopark,
Singapore 119967
Tel: (65) 6835-3011/Fax: (65) 6835-2711

OMRON (CHINA) CO., LTD.
Room 2211, Bank of China Tower,
200 Yin Cheng Zhong Road,
PuDong New Area, Shanghai, 200120, China
Tel: (86) 21-5037-2222/Fax: (86) 21-5037-2200

Authorized Distributor: