Pro-face
Human Machine Interface

Pro-face

Data-Sharing API

User Manual

Preface

Thank you for purchasing the Pro-Designer graphical editing software from Pro-face. Please note that the PS Series Type P target machines used in this manual's examples can be interchanged with the GP-2000 Series target machines.

Please read this manual thoroughly to understand the correct and safe use of this product and its features.

< Note >

1. It is forbidden to copy the contents of this manual, in whole or in part, except for the user's personal use, without the express permission of Digital Electornics Corporation of Japan.
2. The information provided in this manual is subject to change without notice.
3. This manual has been written with care and attention to detail. However, should you find any errors or ommissions, please contact Digital Electronics Corporation and inform us of your findings.
4. Please be aware that Digital Electronics Corporation shall not be held liable by the user for any damages, losses, or third-party claims arising from the uses of this product.

Table of Contents

Glossary

Client Target Machine (Client):	A target machine that can read from and write to the server's data-shared variables.
Data-Sharing API Client:	A machine that runs a Data-Sharing API application created using Data-Sharing APIs.
User-Application Client:	A target machine that runs a user application created in Pro-Designer. A user-application client can also act as a server.
Data-Sharing API:	The Data-Sharing Application Programming Interface (API) provides functions that allow programmers to create applications that can access or use variables provided by a Server Target Machine.
Data-Sharing API Application:	An application created with Data-Sharing APIs and that interacts with Pro-Designer Runtime. A Data-Sharing API Application, unlike a user application, cannot act as a server.
Data-Sharing Protocol:	The language used by target machines—clients and servers—to communicate with each other.
Pro-Designer Editor:	Pro-Designer—the HMI editor software used to create user applications that run in Pro-Designer Runtime.
Pro-Designer Runtime:	The program that executes the user application, and runs on a target machine.
Server Target Machine (Server):	A target machine that shares its variables with client target machines.
Target Machines:	A platform that Pro-Designer Runtime uses to execute the application file.
Client Target Machine:	A target machine that can read and write to the server's data-shared variables.
Server Target Machine:	A target machine that shares its variables with client target machines.
User Application:	An application file created in Pro-Designer, which runs on Pro-Designer Runtime.
Variables:	Data placeholders.

Chapter 1 Overview

This chapter provides a general explanation of the role of the Data-Sharing API.

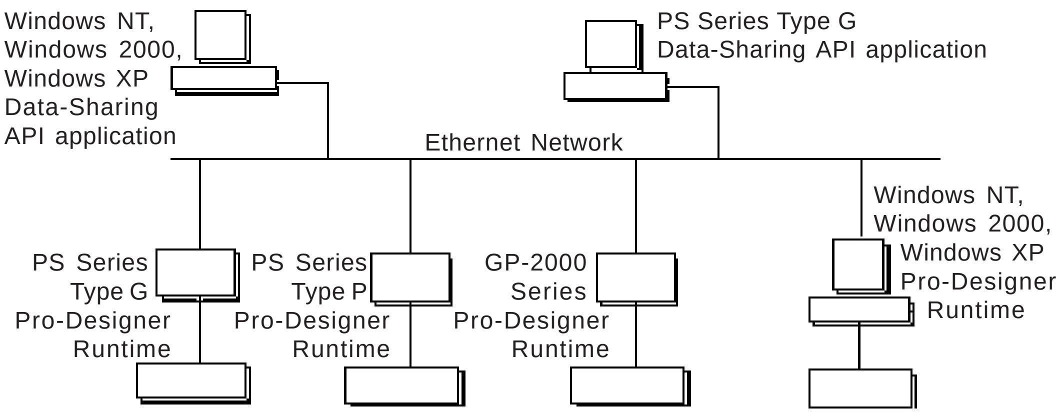
The Data-Sharing API is the instruction set available to the user so they can create custom applications that can communicate with Pro-Designer user applications. When the user application is sharing its variables, the Data-Sharing API provides a gateway for the custom application to read and write to these variables.

Please read this *Data-Sharing API User Manual* before using the Data-Sharing API programming set.

1.1 Operation

Use the Data-Sharing API to create an application that can access shared variables on target machines.

Variables must be shared in the user application for other machines to access the data. This process, known as data-sharing, enables target machines to share variables with other target machines, and to share variables with Data-Sharing API applications (see diagram, below).


Windows NT, Windows 2000, Windows XP Data-Sharing API application
PS Series Type G Data-Sharing API application
Ethernet Network
PS Series Type G Pro-Designer Runtime
PS Series Type P Pro-Designer Runtime
GP-2000 Series Pro-Designer Runtime
Windows NT, Windows 2000, Windows XP Pro-Designer Runtime

When you use data-sharing:

- Target machines can share variables with other target machines.
- Target machines can share variables with Data-Sharing API applications.

1.2 Features

- Data exchange can occur between a Pro-Designer Runtime target machine and a Data-Sharing API application.
- No additional setup or configuration is required in Pro-Designer to support data-sharing, which means that:
  - No extra hardware, such as a server, is required or affected.
  - No setup of PLCs is required.
  - Variable data can be transferred at higher speeds and with lower overhead costs.
- Because the software does not affect other hardware, the setup takes place in only one location.

Chapter 2 System Design

2.1 Operating Environment

Pro-Designer Runtime uses the Data-Sharing protocol to communicate with user applications on other target machines. The protocol is designed to run within the Pro-Designer Runtime communication system as an event-driven protocol. For data-sharing to occur, the protocol enables other target machines to use the variables shared by a user application.

In addition to working within the Pro-Designer Runtime system, you can use the Data-Sharing API to create an application that uses the protocol to access the shared variables on another target machine.

The following table lists the target machines that support Data-Sharing or the Data-Sharing API.

Target Machine	Data Sharing	Data Sharing API
PC/AT (Windows NT, 2000, XP)	√	√
PS Series Type G (Windows CE)	√	√
PS Series Type P	√	X
GP-2000 Series	√	X

2.2 How to Use Data-Sharing Functions

You can use data-sharing in the following ways:

- Connect to Pro-Designer Runtime from a Data-Sharing API application.

  An application created using the Data-Sharing API can access variables shared by a Pro-Designer Runtime user application.

- Connect to Pro-Designer Runtime from Pro-Designer Runtime.

  A Pro-Designer Runtime user application can access variables shared by another Pro-Designer Runtime user application.

Reference *Chapter 8 – "Operation"*

Notes

1. Data-Shared Variables
2. Server / Client Connections
3. Communication Process

Chapter 3 Feature Definitions

3.1 Data-Sharing Variables

- A server is any target machine that shares variables with other target machines.

  On the server, whenever the value of a shared variable changes, the updated value is sent to all the clients that use that variable.
- A client can both read from and write to the a server's shared variables, and include:
  - Target machines that run Pro-Designer Runtime
  - Machines (such as PCs) that run Data-Sharing API applications
- On the target machine, Pro-Designer user applications can act as server, client, or both.
- Data-Sharing API applications can act only as clients.

3.2 Server / Client Connections

A connection can be defined as an access point from a target machine to either a client or a server. In the client/server communication process, each target machine has a limited maximum number of connections.

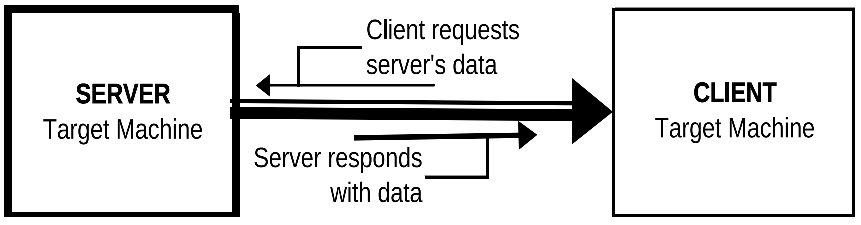
• PC/AT (PL Series):	32
• PS Series Type G:	16
• PS Series Type P:	8
• GP-2000 Series:	8
• Factory Gateway:	4

3.3 Communication Process

The diagram below illustrates the client/server communication process, whereby:

1. The client requests variables from the server.
2. The server responds by sharing the requested variables with the client.

Client/Server Communication



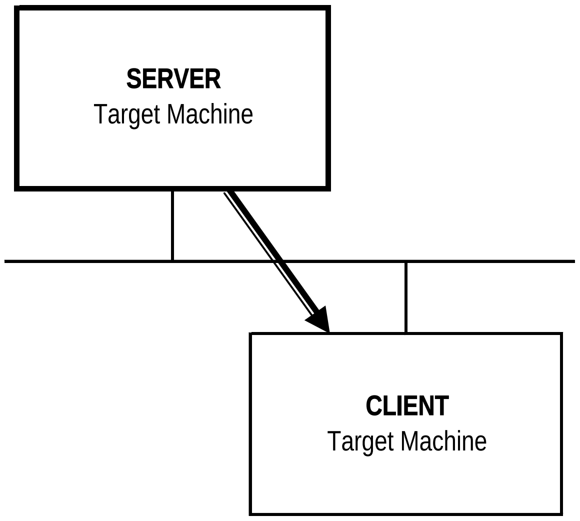
In the following examples, the direction of each arrow represents the flow of shared data from server to client during the communication process.

■ Client/Server 1-Way Communication

During one-way communication:

1. The client requests data from the server (thin line).
2. The server responds by sharing the requested data (thick line) with the client.

Client/Server One-Way Communication



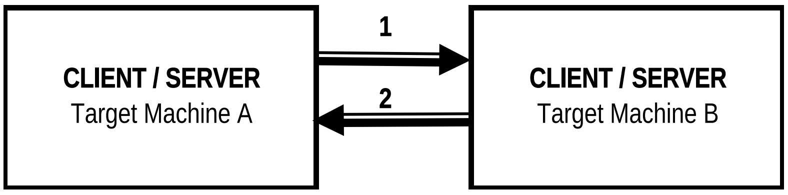
Note:

- When a client initially connects to the server, the server provides the client with the requested variables.
- Whenever changes occur to a server's data-shared variables, the values are updated on all the clients that use those variables.

■ Client/Server 2-Way Communication

The client/server two-way communication process is a peer-to-peer relationship between two target machines.

Client/Server Two-Way Communication



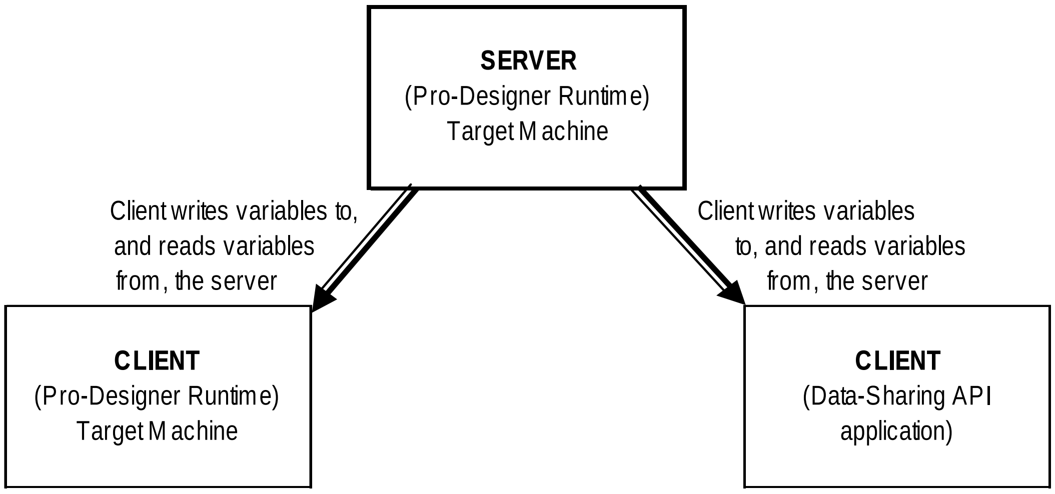
1. Target Machine A requests data (thin line) from Target Machine B. Target Machine B responds (thick line) by sharing the requested variable data with Target Machine A. In this case, Target Machine A is the client and Target Machine B is the server.
2. In the second communication, which can occur simultaneously with the first communication, the roles are reversed. Target Machine B is now the client and Target Machine A is the server. Target Machine B requests data (thin line) from Target Machine A, which responds by sharing the requested variable data with Target Machine B.

■ Multi-Client 1-Way Communication

Two types of clients can communicate with the server:

- Target machines running a user application in Pro-Designer Runtime
- Machines running a Data-Sharing API application

Multi-Client One-Way Communication


SERVER
(Pro-Designer Runtime)
Target Machine
Client writes variables to, and reads variables from, the server
Client writes variables to, and reads variables from, the server
CLIENT
(Pro-Designer Runtime)
Target Machine
CLIENT
(Data-Sharing API application)

The only difference between the multi-client and client/server one-way communication processes is that, during multi-client one-way communication, the server target machine shares its variables with more than one client.

■ Multi-Target 2-Way Communication Process

Multi-target communication is a simultaneous, two-way communication process, where the target machines can act as both server and client.

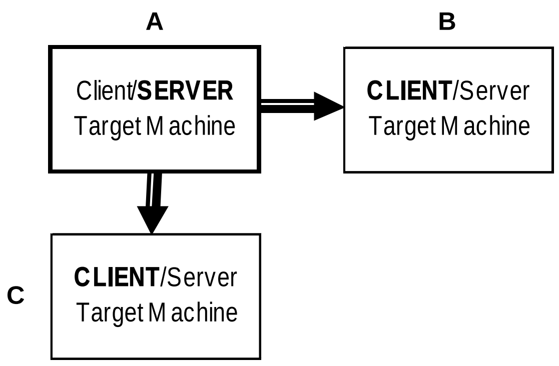
Each target machine can play the role of client and request data from the other target machines, which play the role of servers. In addition to playing the role of client, each target machine can also fill the role of server and share its variables with other target machines.

In this communication process, the target machines operate in a peer-to-peer relationship.

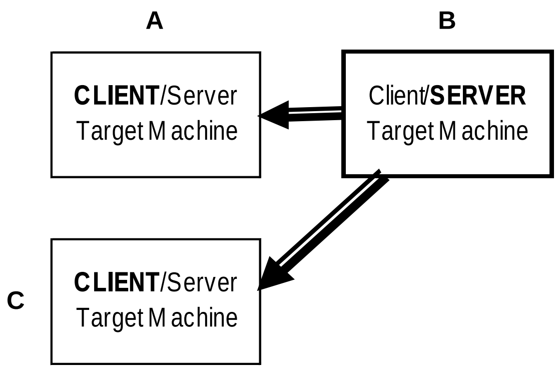
In the following diagram, the first three scenarios show the data-sharing process between three target machines, as a one-way communication process between server and client. Then, in the the fourth scenario, during the multi-target two-way communication process, all target machines communicate simultaneously—*not only as servers, but also as clients*—with the other client/server target machines.

Multi-Target Two-Way Communication

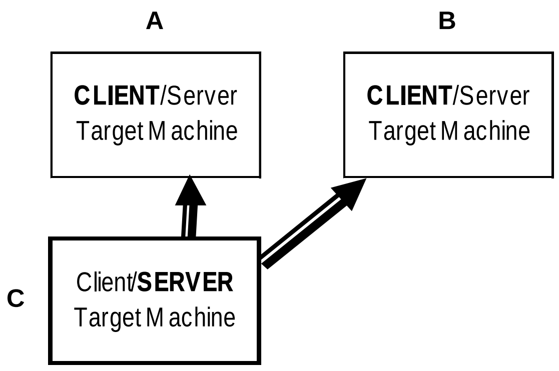
1. One-way communication between Target Machine A (acting as server) and Target Machines B and C (both acting as clients).


A
B
C
Client/SERVER Target Machine
CLIENT/Server Target Machine
CLIENT/Server Target Machine

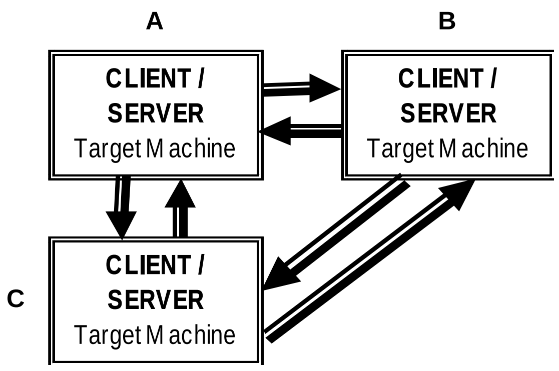
2. One-way communication between Target Machine B (acting as server) and Target Machines A and C (both acting as clients).


A
B
C
CLIENT/Server Target Machine
Client/SERVER Target Machine
CLIENT/Server Target Machine

3. One-way communication between Target Machine C (acting as server) and Target Machines B and C (both acting as clients).


A
B
C
CLIENT/Server Target Machine
CLIENT/Server Target Machine
Client/SERVER Target Machine

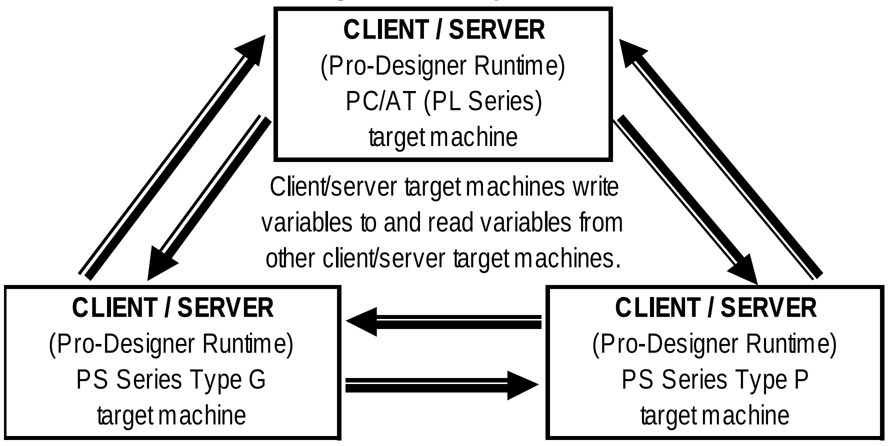
4. Two-way communication between Target Machines A, B, and C (each acting as both client and server).


A
B
C
CLIENT / SERVER Target Machine
CLIENT / SERVER Target Machine
CLIENT / SERVER Target Machine

■ Multi-Platform, Multi-Target 2-Way Communication

The multi-platform, multi-target two-way communication process refers to two-way communication between multiple types of target machines.

Multi-Platform, Multi-Target Two-Way Communication



In the diagram above, three target machines are simultanously sharing and accessing variables. The significant aspect of this communication process is that the platforms of each target machine is different. There is a PC/AT (PL Series) target machine, a PS Series Type G target machine, and a PS Series Type P target machine. Communication occurs seamlessly between the different platforms.

Notes

Chapter 4 Performance Specifications

A target machine can act as a Data-Sharing server, a client, or both. A machine running a Data-Sharing API application can act only as a client. Multiple clients can access the same variables.

Like any communication system, for data-sharing to run efficiently, certain criteria must be met. One of the criteria is limiting the number of shared variables.

The suggested maximum number of variables depends on:

- The number of variables that a client accesses from all other servers (i.e., clients can access variables from more than one server)
- The number of variables shared by the server

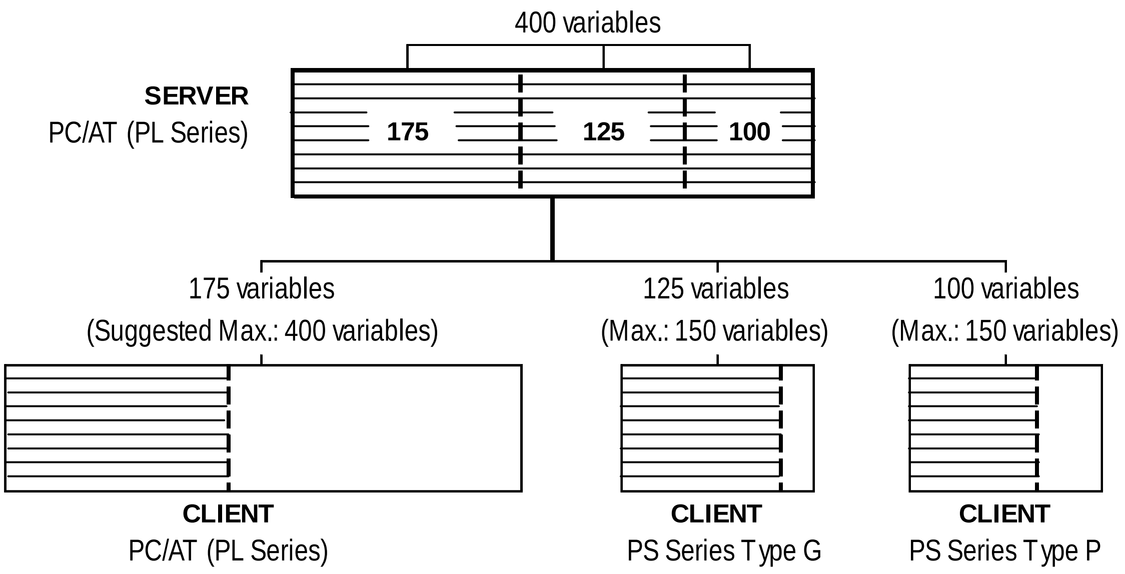
The suggested maximum number of variables shared by each type of target machine is:

• PC/AT (PL Series):	400
• PS Series Type G:	150
• PS Series Type P:	150
• GP-2000 Series:	150
• Factory Gateway:	75

■ Suggested Maximum Number of Variables Shared by the Server

In the following example, the variables accessed by each client are not accessed by any of the other clients. Of its 400 variables, the server shares:

- 175 with the PC/AT client
- 125 with the PS Series Type G client
- 100 with the PS Series Type P client



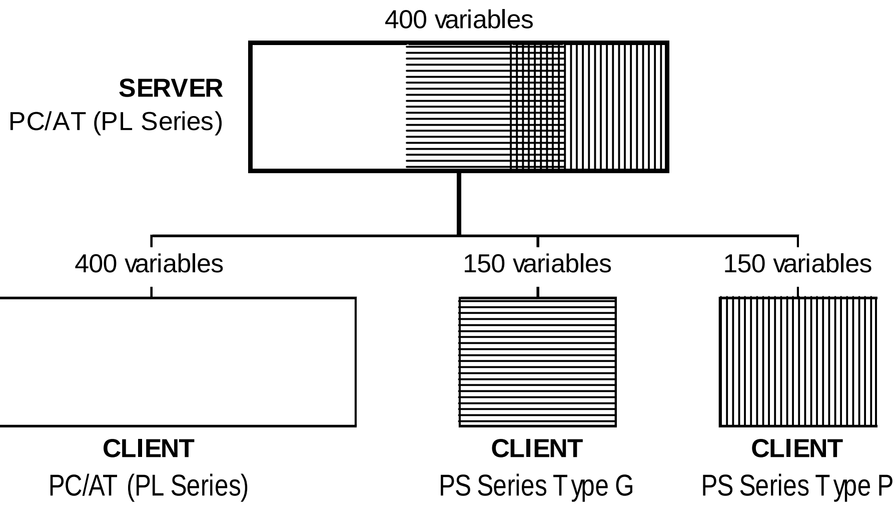
■ Suggested Maximum Number of Variables Accessible by Each Client

The following diagram is an example of the suggested maximum number of variables that each client can access from the server.

Reference *3.2 – “Communication Process”*

In this example, each client accesses the suggested maximum number of variables from the same server. Some or all of the variables accessed by each client are also being accessed by the other two clients.

- The PC/AT (PL Series) client accesses 400 variables, 250 of which are accessed by the other two clients.
- The PS Series Type G client accesses 150 variables, all of which are accessed by the PC/AT (PL Series) client, and 50 of which are accessed by the PS Series Type P client.
- The PS Series Type P client accesses 150 variables, all of which are accessed by the PC/AT (PL Series) client, and 50 of which are accessed by the PS Series Type G client.



▢ – variables accessed by the PC/AT only (150)
▤ – variables accessed by the PC/AT and the PS-G (100)
▥ – variables accessed by the PC/AT and the PS-P (100)
▦ – variables accessed by all three clients (50)

400 – total number of variables shared by the server

In this example, 400 is the total number of variables that the server shares with the clients. However, because each client accesses each variable separately, the server’s processing load is the same as if 700 variables are shared.

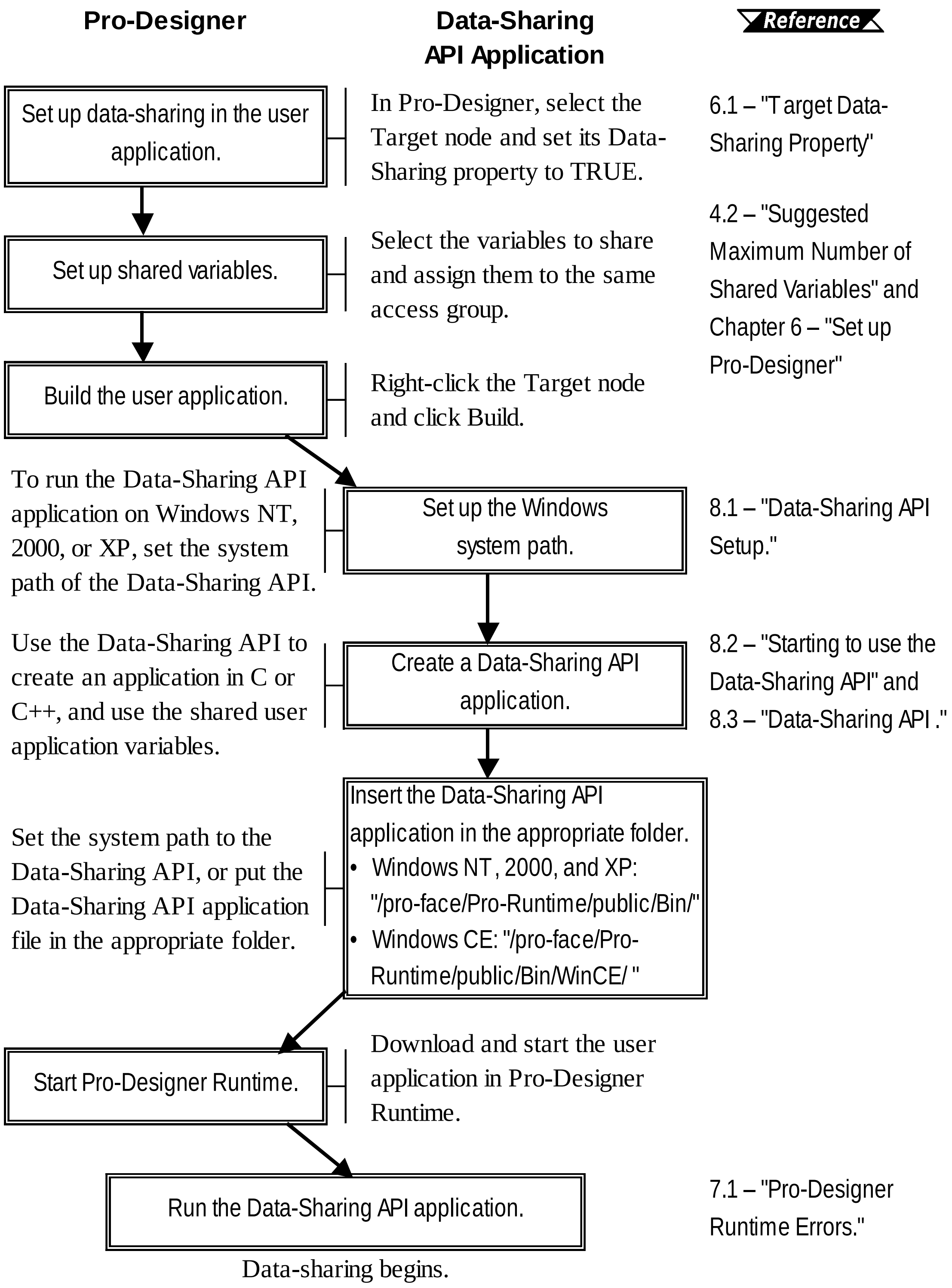
The number of variables that any one type of target machine can share is limited not by design, but by the speed required to process variables.

The suggested maximum number of variables shared between two target machines depends on:

- The type of target machines
- The number of variables shared by the server
- The number of variables that a client is accessing from other servers
- Processing load on clients and servers

Chapter 5 Steps: Using Data-Sharing API

Follow the steps in the flowchart to use the Data-Sharing API and access the variables shared by a Pro-Designer Runtime user application.



Pro-Designer
Data-Sharing API Application
Reference
Set up data-sharing in the user application.
Set up shared variables.
Build the user application.
Set up the Windows system path.
Create a Data-Sharing API application.
Start Pro-Designer Runtime.
Run the Data-Sharing API application.

Set up data-sharing in the user application. In Pro-Designer, select the Target node and set its Data-Sharing property to TRUE. 6.1 – "Target Data-Sharing Property"

Set up shared variables. Select the variables to share and assign them to the same access group. 4.2 – "Suggested Maximum Number of Shared Variables" and Chapter 6 – "Set up Pro-Designer"

Build the user application. Right-click the Target node and click Build.

Set up the Windows system path. To run the Data-Sharing API application on Windows NT, 2000, or XP, set the system path of the Data-Sharing API. 8.1 – "Data-Sharing API Setup."

Create a Data-Sharing API application. Use the Data-Sharing API to create an application in C or C++, and use the shared user application variables. 8.2 – "Starting to use the Data-Sharing API" and 8.3 – "Data-Sharing API ."

Insert the Data-Sharing API application in the appropriate folder.
- Windows NT , 2000, and XP: "/pro-face/Pro-Runtime/public/Bin/"
- Windows CE: "/pro-face/Pro-Runtime/public/Bin/WinCE/ "

Set the system path to the Data-Sharing API, or put the Data-Sharing API application file in the appropriate folder.

Start Pro-Designer Runtime. Download and start the user application in Pro-Designer Runtime.

Run the Data-Sharing API application. 7.1 – "Pro-Designer Runtime Errors."

Data-sharing begins.

Notes

Chapter 6 Set up Pro-Designer

6.1 Target DataSharing Property

To enable data-sharing on a target machine, in the Pro-Designer editor, select the Target node, and in the Inspector set the DataSharing property to TRUE.

The DataSharing property changes operations as follows.

	DataSharing = TRUE	**DataSharing = FALSE**
Functionality	Variables in the user application are not shared.	Variables in the user application can be shared with other Targets in the Pro-Designer project and with Data-Sharing API applications.
Accessing variables in other user applications	Can read to / write from variables in other target machines.	Can read to / write from variables in other target machines.

6.2 Target IPAddress and Port Properties

The IPAddress property is available when you click the Target node. Enter the IP address of the target machine where this user application will be running. The defined IP address is used by data-sharing to enable other target machines to access this target's variables.

When the Target node's DataSharing property is TRUE, the Port property becomes available. Port defines the communication port number used by the Data-Sharing protocol. If you don't define a port number, 6000 is used.

6.3 Shared Variable Reference Format

To reference a shared variable, enter the target and variable name:

[Target Name].[Variable Name]

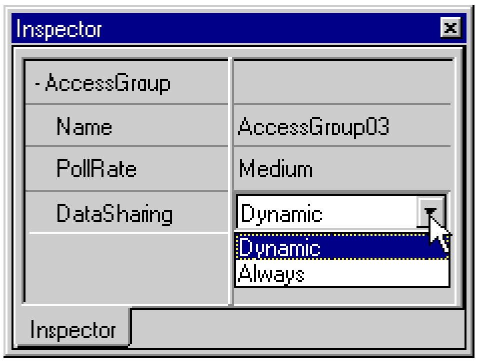
For example, **Assembly7.OverflowTank** refers to the variable **OverflowTank** in the target **Assembly7**.

Script Example

```
int itemp;
itemp = 123;
Assembly7.OverflowTank.write(itemp);
```

6.4 Access Group DataSharing Property

From the Navigator window's Project tab, expand Target and I/O Manager, then click the AccessGroup node. In the Inspector, set the DataSharing property to define whether the variables assigned to the access group are always monitored (Always), or are monitored only when being used on the target machine (Dynamic).



Name	AccessGroup03
PollRate	Medium
DataSharing	Dynamic

Dynamic An access group is set to Dynamic, by default, so that updates will occur:

- when a variable is used in the current panel of either the target machine or a remote machine
- when a variable's KeepHistory property is set to True
- when a variable is used in trend graphs
- when a variable alarm is enabled
- when a variable is used in scripts

Pro-face recommends that you use the default setting (Dynamic) in the DataSharing property.

Always Pro-face recommends that you set only the highest priority variables to Always. Otherwise, system performance will decline.

Note:

- To run both Pro-Designer Runtime and a Data-Sharing API application on the same machine, start Pro-Designer Runtime before starting the Data-Sharing API application.
- To start the Data-Sharing API application, insert a short-cut to Pro-Designer Runtime in the Windows Startup menu, then create a script in the user application (*createProcess*).

Chapter 7 Pro-Designer Runtime

This chapter describes Pro-Designer Runtime features related to shared variables.

7.1 Pro-Designer Runtime Errors

The following table lists error conditions.

	Condition	Problem	Error Confirmation
1	Server target machine does not start up	Pro-Designer Runtime does not start up on the server target machine.	An error message displays on the client target machine, and only the display area of Value animations is visible. When the data type for the Value animation is string, then the display area and the defined string are visible.
		The cable that connects the server target machine and the PLC was disconnected during the startup of Pro-Designer Runtime.	A PLC communication error message displays on the server target machine. On the client target machine, an error message displays initially, but then the user application runs with variable values of 0. Float and integer variables are 0, and discrete variables are displayed with the OFF label. When a value animation displays a string variable, only the display area is visible.
2	No communication with server target machine. i.e., unable to read from or write to data-shared variables.	The cable that connects the server target machine and the PLC has been disconnected.	A PLC communication error message displays on the server target machine. On the client target machine, an error message displays initially, but processing continues with previously polled values.
		Pro-Designer Runtime has shut down on the server target machine.	On the client target machine, an error message initially displays, but processing continues with previously polled values.
		The cable that connects the server target machine and the client target machine is disconnected.	On the client target machine, an error message initially displays, but processing continues with previously polled values.

7.2 Changing IP Address at Runtime

When using Data-Sharing and you change the network address on a server target machine, the client target machines cannot find the server anymore. This error occurs even when you change the IP address using Pro-Designer Runtime's configuration menu.

When you want to change the IP address of a server target machine, make sure you also change the user application's IP address in the Pro-Designer editor. Define the new IP address in the Target properties and rebuild all the targets that refer to the user application's variables. That means that user applications on all the client target machines must be built and downloaded again so they refer to the server's new IP address.

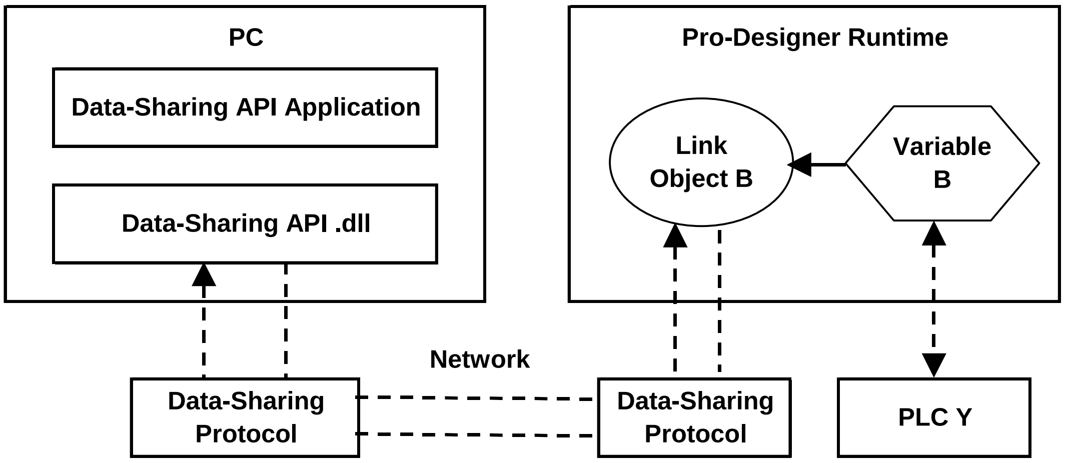
Note: Changing the IP address of one target machine may seem like an innocent change, but if the target machine is acting as a server, then the change has ramifications on multiple target machines and the user applications for all of them must be rebuilt and downloaded.

Chapter 8 Operation

8.1 Data-Sharing API Setup

8.1.1 Connecting Data-Sharing API and Pro-Designer Runtime

Using the the Data-Sharing API, you can create applications in C or C++ that access variable or PLC register values in Pro-Designer Runtime. The Data-Sharing API .dll files are required to run the application.



8.1.2 Data-Sharing API and Protocol Setup

The following settings are required to use the Data-Sharing API and Data-Sharing protocol from an application.

■ System Path

[Windows NT, Windows 2000, or Windows XP]

When using the Data-Sharing API, setting up the system path is required so access is available to all sub-directories in Pro-Designer Runtime. The system path is used to load the kernel, configuration, and error system.

To set up the system path (Windows NT, Windows 2000, or Windows XP):

1. Open the Windows Control Panel and click the **System** icon.
2. In the **System Properties** dialog box, click the **Environment** tab, scroll down the **System Variables** listbox and click the Path variable.
3. Add the path "*installed directory*/pro-face/Pro-Runtime/public/bin" to the **Value** box and click **Set**.

This path is also used as a parameter when initializing the Data-Sharing API (i.e., InitRuntimeAdapter, InitRuntimeAdapterEx).

[Windows CE]

Place the application file (.exe) created with the Data-Sharing API into "*installed directory*/public/bin/WinCE."

■ Project Configuration

The second parameter used by the InitRuntimeAdapterEx function defines the project configuration. The project configuration is a text file which, among other things, identifies the port number used by the Data-Sharing protocol.

To create the configuration file for the Data-Sharing API application, copy the Project.cfg (*installed directory*/Pro-face/Docs/Cfg/Project.cfg) into the directory of the Data-Sharing API application.

Define two parameters when you start InitRuntimeAdapterEx. In the first parameter, define the complete path of Pro-Designer Runtime (*installed directory*/pro-face/Pro-Runtime/public/bin). In the second parameter, define the complete path of the project configuration file.

8.2 Starting to use the Data-Sharing API

The Data-Sharing API is provided as a Windows .dll file.

The following is an introduction to use of the API.

Reference ***8.7 – "API Details"***

- ■ **RuntimeAdapter.h** is located in "*installed directory*/Pro-face/Docs/Include." This header file lists all the data types used in the Data-Sharing API.
- ■ To use the Data-Sharing API, the user created application must load the RuntimeAdapter.dll file and get a pointer to the process (LoadLibrary,

8.3 Data-Sharing API

Name	**Description**
InitRuntimeAdapter	Initializes the Data-Sharing APIs. Must be called before any other Data-Sharing API calls.
InitRuntimeAdapterEx	An extended Data-Sharing API initialization function that enables running a Data-Sharing API application on the same platform as Pro-Designer Runtime. To ensure the same Project.cfg file, and thus the same port number, is not used for both, the second parameter defines the location of the configuration file for the Data-Sharing API application.
ConnectToVars	Connects the Data-Sharing API application's variable list with the server target machine's variable list. A connection handle is returned for each variable that is successfully connected. Otherwise, an invalid handle (–1) is returned. The Data-Sharing API application is responsible for allocating and releasing the buffer for the list of handles.
DisconnectFromVars	Releases all the variable connections from the list of handles.
WriteDataToVar	Uses the defined connection to write a value to the shared variable. Returns TRUE when the operation is successful.
RegConnectErrorInformCallback	Registers the function called when a variable connection cannot be established.
RegUpdateDataCallback	Registers the function called when the Data-Sharing API application receives updated data.
ShutdownRuntimeAdapter	Ends the Data-Sharing API application.

Note:

- When defining the path of Project.cfg (for InitRuntimeAdapterEx), name the complete path, including the filename.
- When using Pro-Designer Runtime and the Data-Sharing API application on the same machine, the IP address passed to the ConnectToVars function should be “INET:127.0.0.1:xxxx,” where xxxx is the Port number of the user application on the server target machine. The 127.0.0.1 IP address enables access to Pro-Designer Runtime without going through the network, even though the target machine has a different IP address.

8.4 Steps to Access Shared Data

8.4.1 Initialize the Data-Sharing API

- Use InitRuntimeAdapterEx to initialize the Data-Sharing API and the project configuration file. Before making this call, set up the system path—required for loading the kernel, configuration, and error system. Once the Data-Sharing API is initialized, use the RegConnectErrorInformCallback and RegUpdateDataCallback functions in the Data-Sharing API application to register the functions that handles errors and variable updates.

8.4.2 Read Data

- Use RegisterUpdateDataCallback to register the function that handles data updates with the Data-Sharing API. RegisterUpdateDataCallback is called whenever the Data-Sharing API receives new data from the server target machine.
- ConnectToVars is asynchronous. This function immediately returns an empty variable handle, which is populated when the server target machine responds with the requested variable data. If an error has occurred, such as the variable is not available, the Data-Sharing API notifies the application by calling the function registered with RegConnectErrorInformCallback.

8.4.3 Write Data

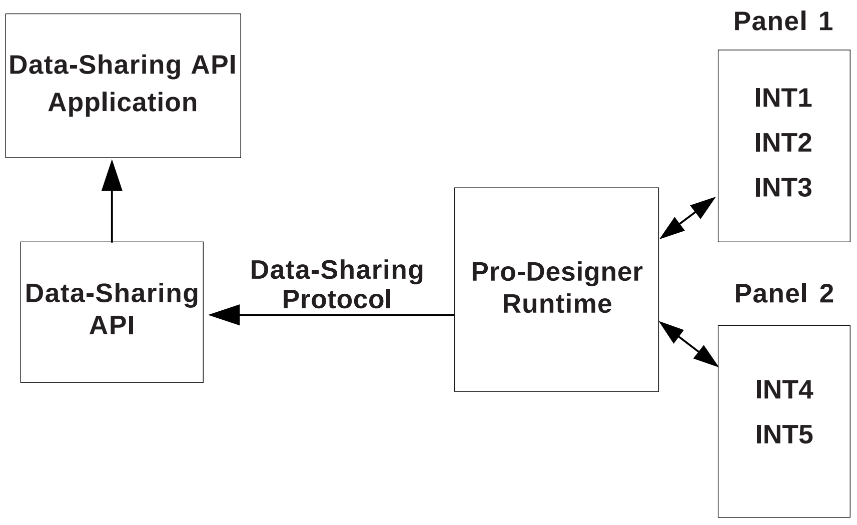
- The Data-Sharing API application can write data to a shared variable at any time using the WriteDataTo function.

  When data from a variable is no longer required, you can release the connection to the variable in the Data-Sharing API application by calling DisconnectFromVars. Or, to shutdown the Data-Sharing API altogether, call ShutdownRuntimeAdapter.

8.5 Examples: Data-Sharing API

The following examples show how to use the Data-Sharing API in specific situations.

Example 1 – Data-Sharing API Application Reads Data from Pro-Designer Runtime



■ InitRuntimeAdapterEx(runtime directory, configuration file)

- Called once per process, initializes the Data-Sharing API.
- Runtime directory is the root directory of Pro-Designer Runtime (e.g. *installation directory*\pro-face\Pro-Runtime\public).
- The configuration file is the “Project.cfg” file used to configure the Data-Sharing API.

■ RegUpdateDataCallback(UpdateDataCallback)

- In this example, UpdateDataCallback is the function defined by the user to handle data updates from the Data-Sharing API.
- When connecting, this function is called once for each variable so its value is initialized.

■ ConnectToVars()

- The Data-Sharing API application connects to all the variables in the Pro-Designer Runtime user application, and uses VAR_READ_ONLY_ATTRIB (RuntimeAdapter.h).
- When the connection is established, UpdateDataCallback is called once for each variable, to initialize the variable values in the connection list.
- ConnectToVars returns immediately, and when there is an error, such as the defined variable name does not exist, an error is passed to the user defined function ErrorInformCallback.

■ RegConnectErrorInformCallback(ErrorInformCallback)

- ErrorInformCallback, a function defined in the Data-Sharing API application, is called by the Data-Sharing API when it could not connect to a Pro-Designer variable.

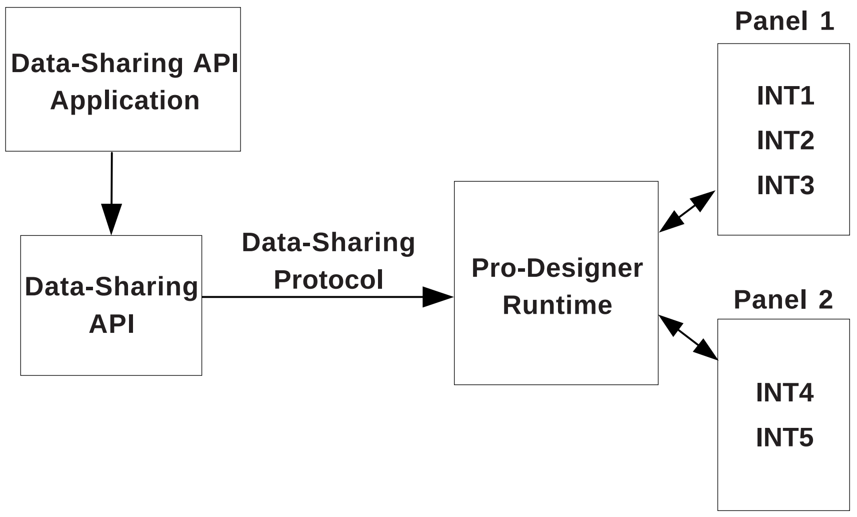
■ DisconnectFromVars()

- This function breaks connections to the specified variables.

■ ShutdownRuntimeAdapter()

- This function is called once per process, to end the Data-Sharing API when exiting the process.

Example 2 – Data-Sharing API Application Writes Data to Pro-Designer Runtime



■ InitRuntimeAdapterEx(runtime directory, configuration file)

- Called once per process, initializes the Data-Sharing API.
- Runtime directory is the root directory of Pro-Designer Runtime (e.g. *installation directory*\pro-face\Pro-Runtime\public).
- The configuration file is the “Project.cfg” file used to configure the Data-Sharing API.

■ ConnectToVars()

- The Data-Sharing API application connects to all the variables in the Pro-Designer Runtime user application, and uses VAR_READ_ONLY_ATTRIB (RuntimeAdapter.h).
- ConnectToVars returns immediately, and when there is an error, such as the defined variable name does not exist, an error is passed to the user defined function ErrorInformCallback.

■ RegConnectErrorInformCallback(ErrorInformCallback)

- ErrorInformCallback, a function defined in the Data-Sharing application, is called by the Data-Sharing API when it could not connect to a Pro-Designer variable.

■ WriteDataToVar()

- The Data-Sharing API application uses this function to write values to variables in Pro-Designer Runtime. Identify the variable by using the handle returned by the ConnectToVars function. You can write only to one variable at a time. Call this function multiple times to write to more than one variable.

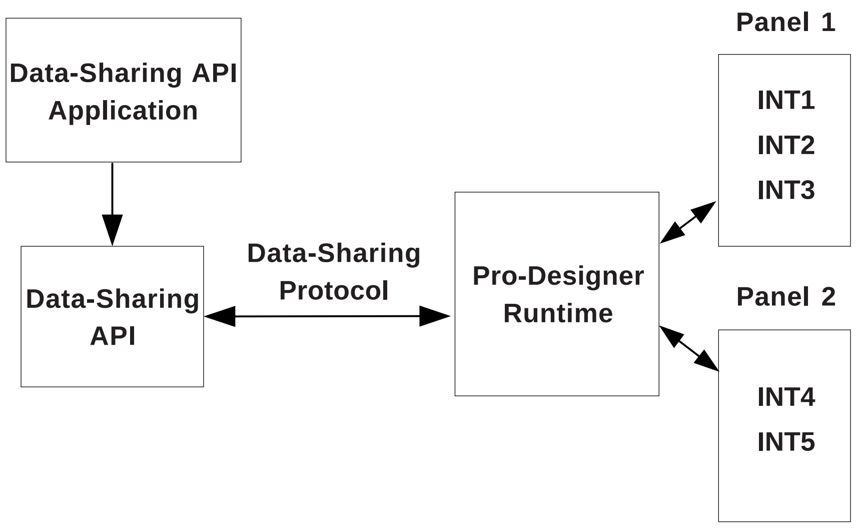
■ DisconnectFromVars()

- This function breaks connections to the specified variables.

■ ShutdownRuntimeAdapter()

- This function is called once per process, to end the Data-Sharing API when exiting the process.

Example 3 – Data-Sharing API Application Reads / Writes Data to Pro-Designer Runtime



■ InitRuntimeAdapterEx(runtime directory, configuration file)

- Called once per process, initializes the Data-Sharing API.
- Runtime directory is the root directory of Pro-Designer Runtime (e.g. *installation directory*\pro-face\Pro-Runtime\public).
- The configuration file is the “Project.cfg” file used to configure the Data-Sharing API.

■ RegUpdateDataCallback(UpdateDataCallback)

- In this example, UpdateDataCallback is the function defined by the user to handle data updates from the Data-Sharing API.
- When connecting the variables, this function is called once for each variable so their values are initialized.

■ ConnectToVars()

- The Data-Sharing API application connects to all the variables in the Pro-Designer Runtime user application, and uses VAR_READ_ONLY_ATTRIB (RuntimeAdapter.h).
- When the connection is established, UpdateDataCallback is called once for each variable, to initialize the variable values in the connection list.
- ConnectToVars returns immediately, and when there is an error, such as the defined variable name does not exist, an error is passed to the user defined function ErrorInformCallback.

■ RegConnectErrorInformCallback(ErrorInformCallback)

- ErrorInformCallback is the user created function called by the Data-Sharing API when a variable is not found on the Pro-Designer Runtime.

■ WriteDataToVar()

- The Data-Sharing API application calls this function to write values to a variable in the Pro-Designer Runtime user application. Identify the variable by using the handle returned by the ConnectToVars function. You can write only to one variable at a time. Call this function multiple times to write more than one variable.

■ DisconnectFromVars()

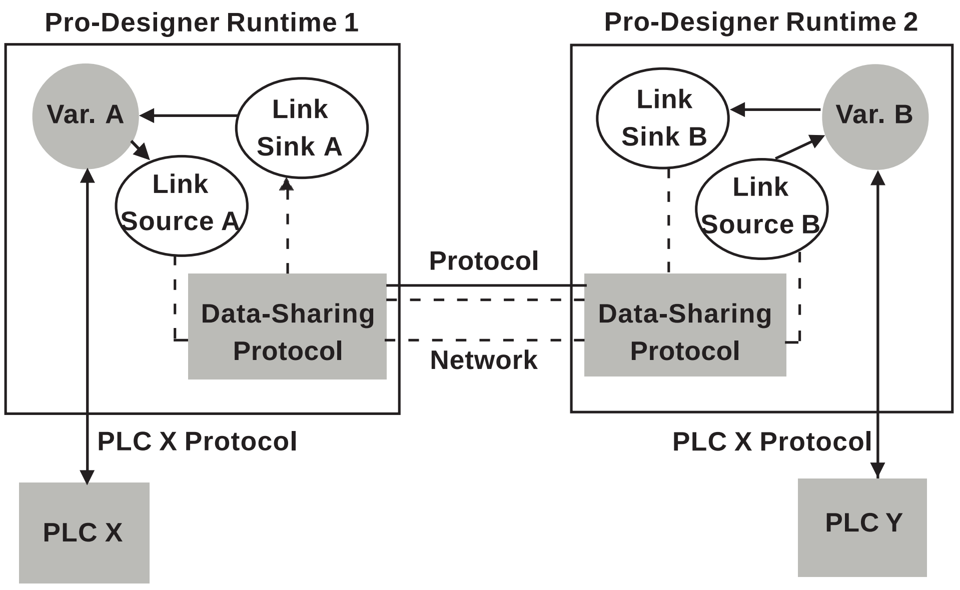
- This function breaks connections to the specified variables.

■ ShutdownRuntimeAdapter()

- This function is called once per process, to end the Data-Sharing API when exiting the process.

8.6 Connection from One Pro-Designer Runtime to Another

The following diagram shows how Pro-Designer Runtime connects to another instance of Pro-Designer Runtime.



All the necessary Java, configuration, and communication data files are generated by the Pro-Designer editor, so no extra setup is required by the user.

- In Pro-Designer Runtime 1, Variable A reads and writes data to and from PLC X.
- In Pro-Designer Runtime 2, Variable B reads and writes data to and from PLC Y.
- In addition, Variable A and Variable B share their data with the other runtime.

8.7 API Details

■ Details of RuntimeAdapter.h

- Some of the data types used by the functions listed in this section are defined in RuntimeAdapter.h (see table, below).

Type	Value	Constant
Variable Data Type		
unsigned char		UNIT8;
unsigned short		UNIT16;
unsigned int		UNIT32;
signed char		INT8;
signed short		INT16;
signed int		INT32;
Attribute		
Write Only	1	VAR_WRITE_ONLY_ATTRIB
Read Only	2	VAR_READ_ONLY_ATTRIB
Read Write	3	VAR_READ_WRITE_ATTRIB
Requested Data Type		
Integer	0	VAR_TYPE_INT
Float	1	VAR_TYPE_FLOAT
String	2	VAR_TYPE_STRING
Discrete	3	VAR_TYPE_DISCRETE

Name : InitRuntimeAdapter

```
bool      InitRuntimeAdapter
(
        UNICHAR*   SystemPath
)
```

Parameters : UNICHAR* SystemPath
Path of Pro-Designer Runtime system. NT or 2000 path is "*installation directory*/pro-face/pro-runtime/public/bin," Win CE path is "*installation directory*/public/bin/WinCE."

Return : bool

Comments : Initializes the Data-Sharing API with the Pro-Designer Runtime system. This call must be made prior to any other calls in the Data-Sharing API application. After successfully initializing the API with the runtime system, returns TRUE.

Name : InitRuntimeAdapterEx

```
bool       InitRuntimeAdapter
(
           UNICHAR*    SystemPath,
           UNICHAR*    ConfigFilename
)
```

Parameters : UNICHAR* SystemPath

Path of Pro-Designer Runtime system. NT or 2000 path is "*installation directory*/pro-face/pro-runtime/public/bin," Win CE path is "*installation directory*/public/bin/WinCE."

UNICHAR* ConfigFilename

Entire path of configuration file, including the filename

Return : bool

Comments : Expands on InitRuntimeAdapter. In addition to initializing the Data-Sharing API with the Pro-Designer Runtime system, this method also identifies the configuration file.

This call must be made prior to any other calls in the Data-Sharing API application. After successfully initializing the API with the runtime system and pointing to the configuration file, returns TRUE.

Name : ConnectToVars

```
bool       InitRuntimeAdapter
(
           UNICHAR*    ServerAddr,
           UINT16      NumOfVars,
           UNICHAR*    VarNameList[],
           BYTE*       DataTypeList,
           BYTE*       DirAtribList,
           UINT32*     AssignAppHandleList,
           CONNECTHANDLES*      RetAdapterHandleList
)
```

Parameters : UNICHAR* ServerAddr

Identifies the IP address and Port number of the server target machine the Data-Sharing API application is trying to access.

UNIT16 NumOfTags

Defines the number of variables to connect.

UNICHAR* RemoteVariableNameList[]

Lists the names of the variables to connect in the server target machine

BYTE* DataTypeList

Lists the data types of the variables requested (Integer, Discrete, Float, or String)

BYTE* DirAttribList

Lists the variable read/write control (R, W, R/W)

		UINT32* AssignAppHandleList Lists the handles for each variable provided by the Data-Sharing API application. When a variable value is updated, it uses the variable handle defined here.
		CONNECTHANDLE* RetAdapterHandleList Lists the handles for each variable returned by the Data-Sharing API. The corresponding variable handle is used when the Data-Sharing API application makes a read or write request to the shared variable.
Return	:	bool
Comments	:	Uses the Data-Sharing API to make connections to the defined list of variables in the server target machine. A connection handle is returned for each variable that successfully connects. Otherwise, an invalid handle (-1) is returned. The caller is responsible for allocating and releasing the buffer for the connection handle list.

Name : DisconnectFromVars

```
bool      DisconnectFromVars
(
 UINT16  NumOfHandles,
 CONNECTHANDLE*  AppHandleList
)
```

Parameters	:	UINT16 NumOfHandles Defines the number of connection handles to remove
		CONNECTHANDLE* AdapterHandleList The handle list returned when ConnectToVars is used to establish variable connections.
Return	:	bool
Comments	:	Removes all the variable connections in this list of handles.

Name : WriteDataToVar

```
bool      WriteDataToVar
(
          CONNECTHANDLE  AdapterHandle,
          UINT16         DataLen,
          void*          Data
)
```

Parameters : CONNECTHANDLE AdapterHandle

Defines the variable data for the write operation by passing the corresponding connection handle.

UINT16 DataLen

Defines the length of the data in bytes. Discretes are 1 byte, integers and floats are 4 bytes, and strings can be any number of bytes.

void* Data

The actual data written to the variable on the server target machine.

Return : bool

Comments : Uses the variable connection handle to write data to a shared variable on the server target machine. After successfully writing the data to the variable, returns TRUE; otherwise, returns FALSE.

Writing a value that is the same as the one currently set to a variable will not cause a data change operation in Pro-Designer Runtime.

Name : RegConnectErrorInformCallback

```
void      RegConnectErrorInformCallback
(
 bool (
          CONNECT_STATUS  ConnectionStatus,
          UINT16  NumOfHandles,
          UINT32* AppHandleList   )
)
```

Parameters : bool (* ErrorInformCallback)

User created method that's called when there is an error. Use the defined parameters and return type for ErrorInformCallback.

CONNECT_STATUS ConnectionStatus

Shows the operation or error status of the asynchronous connection.

UINT16 NumOfHandles

When the ConnectionStatus is an error, this parameter defines the number of handles in AppHandleList.

UINT32* AppHandleList

When the ConnectionStatus is an error, this parameter defines the list of application handles with errors.

Return : void

Comments : Use this function to create the function (ErrorInformCallback) that will handle error conditions when attempting to make variable connections. The function must use the defined parameters and return a bool:

```
CONNECT_STATUS ConnectionStatus
UINT16 NumOfHandles
UINT32* AppHandleList
```

The user defined function shows the status of connections:

- RTA_CONNECTING
- RTA_CONNECTED
- RTA_TAGNAME_ERROR
- RTA_TOO_MANY_TAGS_ERROR
- RTA_VERSION_ERROR

When establishing a connection or when a connection is cut (RTA_CONNECTING, RTA_CONNECTED), the Data-Sharing API populates the AppHandleList with pointers to the IP address and port number (UNICODE text strings) of the target machine that was connected or disconnected.

When you try to access more than the number of connections the server target machine can support, ConnectionStatus is RTA_TOO_MANY_TAGS_ERROR. This error can also result when multiple client target machines access variables on the server.

When you try to access a variable that isn't available on the server target machine, ConnectionStatus is RTA_TAGNAME_ERROR.

When you try to access a server target machine that is using a different version of the Data-Sharing protocol, then ConnectionStatus is RTA_VERSION_ERROR.

When there is a connection error, every heartbeat ConnectToVars attempts to connect to the server target machine, and RTA_CONNECTING is passed to this error function.

When the connection is successful with all the defined variables, ConnectionStatus is RTA_CONNECTED.

When the connection is successful with a portion of the variables, then ConnectionStatus is RTA_TAGNAME_ERROR and AppHandleList contains the variable connections that failed.

When there is a problem with the connection, ErrorInformCallback returns RTA_CONNECTING; when the connection is successful, returns RTA_CONNECTED.

Name : RegUpdateDataCallback

```
void      RegUpdateDataCallback
(
 bool (
          UINT32  AppHandle,
          UINT16  DataByteLen,
          void*           Data     )
)
```

Parameters : bool (* UpdateDataCallback)

User created method that's called by the Data-Sharing API when it receives updated variable data from the server target machine. Use the defined parameters and return type.

UINT32 AppHandle

A valid variable handle from the list of handles.

UINT16 DataByteLen

Total byte length of the data.

void* Data

Data that is updated by UpdateDataCallback.

Return : void

Comments : Use this function to create the function (UpdateDataCallback) that will handle variable updates from the Data-Sharing API. The function must use the defined parameters:

```
UINT32  AppHandle
UINT16  DataByteLen
void*           Data
```

The function must return a bool value.

Name : ShutdownRuntimeAdapter

```
bool      ShutdownRuntimeAdapter ( )
```

Parameters : void

Return : bool

Comments : Returns TRUE when the Data-Sharing API is shut down successfully.

Notes

Chapter 9 Sample Code

Code samples for a working Data-Sharing API application are located in:

installed directory/pro-face/docs/sample

Files in the folder are as follows.

• Dsapi/main.cpp	Sample code
• Dsapi/Dsapi.dsw	Project work space
• Dsapi/Dsapi.dsp	Project file
• Dsapi/RuntimeAdapter.h	Header file
• Dsapi/Debug	Stores the debug application
• Dsapi/Release	Stores the production-ready application

Notes