FinsGateway

SerialUnit ProtoDLL SDK Manual

(Adding to the FinsGateway Serial Communication Protocol)

First Edition

May 1998

OMRON Corporation

Contents

1 SerialUnit ProtoDLL SDK

The FinsGateway SerialUnit ProtoDLL SDK (hereafter, ProtoDLL SDK) explains how to develop a protocol conversion DLL (hereafter, ProtoDLL) for the SerialUnit, which is a FinsGateway serial communication unit.

The ProtoDLL establishes communication between a serial line device and a FinsGateway application by interpreting the dedicated protocol of the device. The FinsGateway SerialUnit is already provided with three ProtoDLLs (SYSWAY, SYSWAY-CV, and CompoWay/F).

Using the ProtoDLL SDK, a developer can provide ProtoDLLs for other protocols as well. The ProtoDLL SDK provides data and a sample for ProtoDLL development.

2 Copyright

Everything described or expressed in this FinsGateway Network Service Provider SDK (including documentation, API, electronic file, etc.) is copyrighted and owned by OMRON Corporation.

This FinsGateway Network Service Provider SDK is provided by OMRON Corporation to the user under special conditions. Therefore, the user is not allowed to divulge any of this information in any way, shape, or form, beyond the extent of this agreement, for any reason.

Any of the information concerning the SDK may be changed by OMRON Corporation without notice.

Any software developed using this SDK is not permitted to be marketed in competition with any product of OMRON Corporation.

3 FinsGateway SerialUnit Design Concept

The FinsGateway SerialUnit (hereafter, SerialUnit) is designed to communicate with a device using computer serial communication.

Serial communication is a commonly used communication process for control devices, which follows the standards of RS-232C. This widely used serial communication varies in communication procedure on the communication line (called protocol, here), depending on the device. The variations of this protocol cause a great workload for the design and development of communication applications.

FinsGateway was specifically designed to hide the differences in networks, so it treats serial communication the same as SYSMAC LINK, etc. The following challenges were encountered in accomplishing this:

- Varied protocols
- No network model

The SerialUnit was designed to allow for these differences in serial communication protocols by providing a DLL (Dynamic Link Library) for each protocol. The DLL simplifies the necessary adjustments, so the communication application does not need to consider the differences in protocol.

The SerialUnit characteristics are as follows:

- A structure that absorbs the differences in ProtoDLL protocols
- Supports many ProtoDLL protocols as add-ons
- Provides a network model
- Multiple protocols supported on the same line, by allowing independent protocol definition for each node

4 SerialUnit ProtoDLL Internal Structure

To understand the internal structure, it is necessary to understand how the SerialUnit handles serial communication as a FINS network. The information required to understand this is presented in the FinsGateway SerialUnit manual.

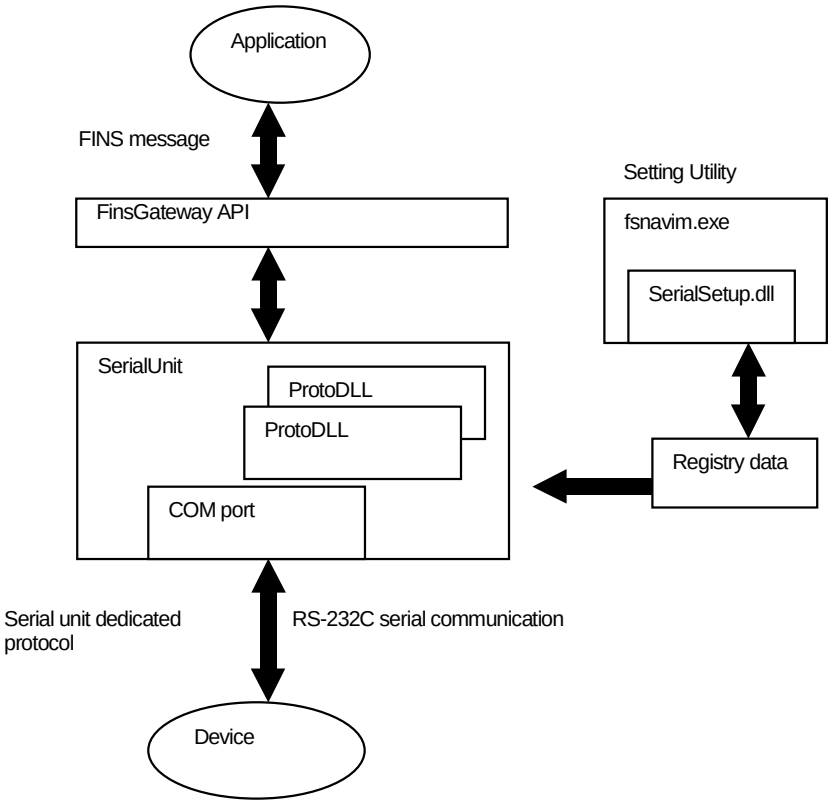
4.1 Overall Structure

The SerialUnit is configured of the following elements:

- SerialUnit (SeriUnit.exe), which operates as a FinsGateway communication unit
- Multiple ProtoDLLs to support each protocol
- Registry to perform the SerialUnit and ProtoDLL operation settings
- SerialSetup.dll for fsnavim.exe, which is the Setting Utility to perform the registry settings
- Srlexplore.exe, which is a utility to search for communication conditions

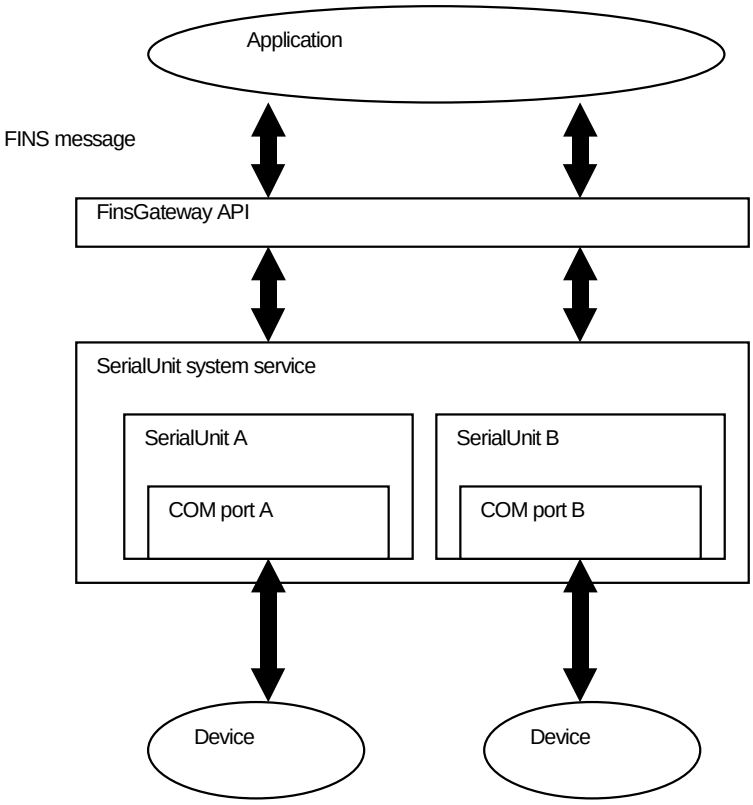
For an application to communicate with a serial target device, the application uses the FinsGateway FINS message API to send messages. With FinsGateway, the message is sent to the SerialUnit. The SerialUnit converts the message to the actual communication protocol, and communicates with the target device.

The SerialUnit protocol processing is performed with an addable DLL known as ProtoDLL. Therefore, adding a specific protocol to the ProtoDLL adds to the protocols supported by the SerialUnit. This also adds to the protocols supported by the application.



While the SerialUnit uses the ProtoDLL to perform the protocol conversion, the communication unit is fundamentally one application message. When the SerialUnit receives an application message, it selects the corresponding ProtoDLL and transfers the processing to that ProtoDLL. When the ProtoDLL completes the processing of that message, it sends the FINS response to the application, and returns execution to the SerialUnit.

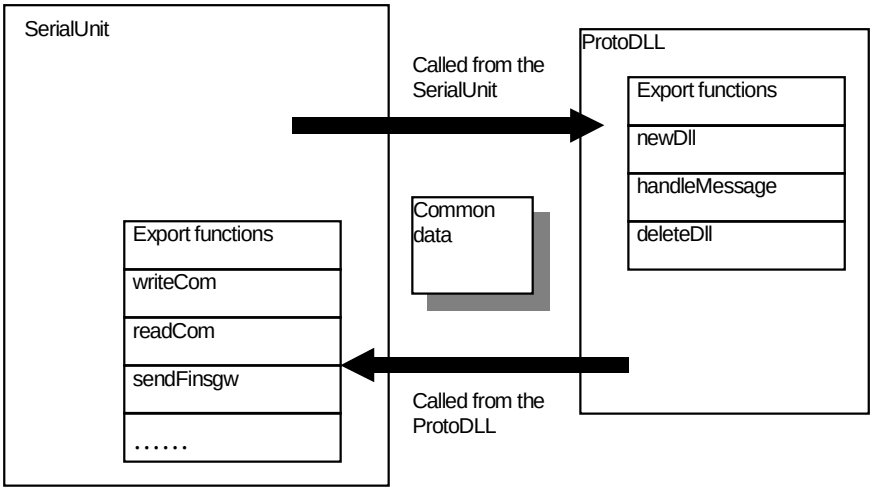
The SerialUnit is created by the SerialUnit system service for each line being used. Therefore, the SerialUnit for the line using COM1, and the SerialUnit for the line using COM2 operate as independent communication units. Between these multiple SerialUnits, there is only a thread boundary. In other words, when developing a ProtoDLL, if the ProtoDLL needs to use any global resources exclusively, it is necessary to protect those resources for the threads throughout the ProtoDLL.



4.2 Export Functions (Public Functions) and Common Data

The SerialUnit and ProtoDLL operate in cooperation with each other. The SerialUnit calls the ProtoDLL when necessary, and the ProtoDLL uses the SerialUnit processing whenever appropriate. This cooperation is based on the SerialUnit exporting functions to the ProtoDLL, and the ProtoDLL exporting functions to the SerialUnit. Some of the exported functions also use common data between the SerialUnit and the ProtoDLL.

To develop a new ProtoDLL, it is necessary to know the functions that the SerialUnit exports. It is also necessary to know the roles of the export functions to include in the ProtoDLL, and the proper stages for those functions to be executed.



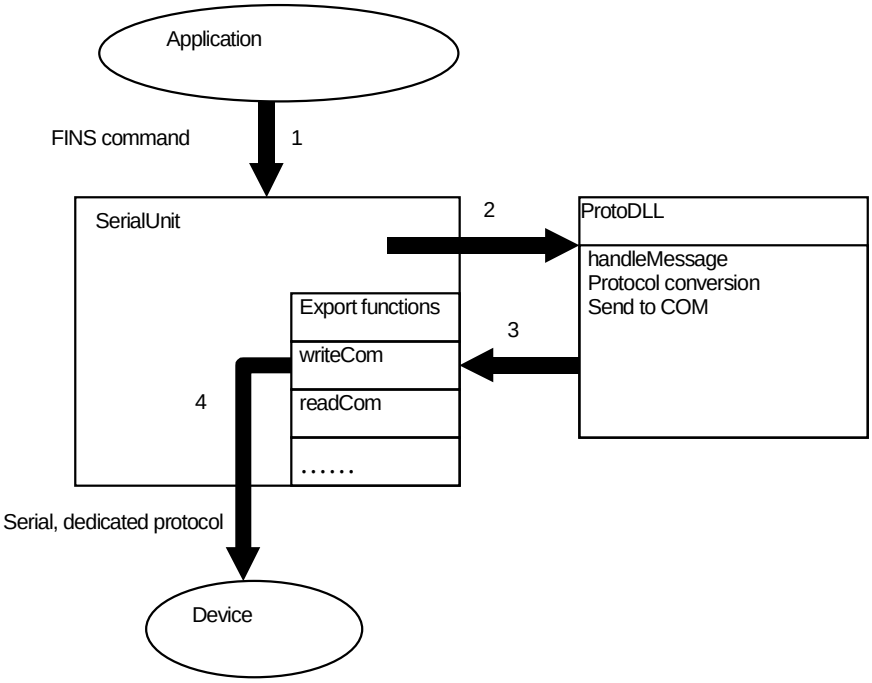
4.3 Communication Example

1. When the application sends one FINS message, the message is forwarded to the SerialUnit for that target address.

2. The SerialUnit determines the ProtoDLL, based on the target node address and the node protocol registered in the registry. The SerialUnit then executes the ProtoDLL export function, handleMessage.

3. From here, all the processing is handled inside the ProtoDLL handleMessage function. Inside handleMessage, the data is converted to the actual protocol required, and sent through the COM port to the target device. To send the data to the COM port, The SerialUnit export function, writeCom is used.

4. The SerialUnit called by writeCom sends the data it received from the ProtoDLL to the COM port.



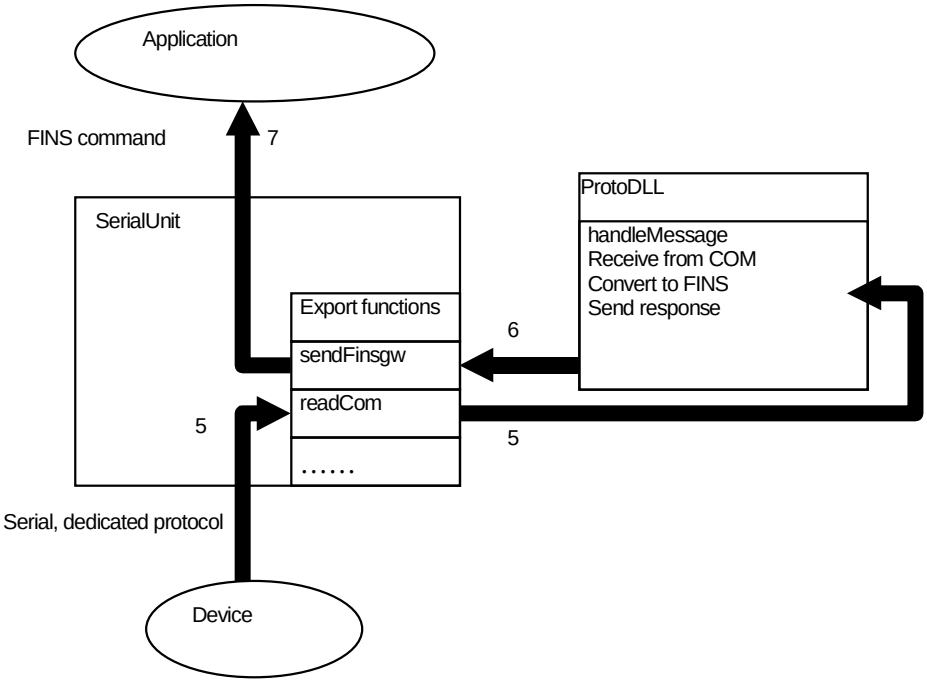
5. Next, the ProtoDLL handleMessage performs the response processing. The handleMessage executes the SerialUnit export function, readCom. The readCom function reads the data received from the COM port.

Based on the protocol, handleMessage continues to receive the device data through the COM port until it has a complete packet. When the data is complete, it is converted to a FINS response. This FINS response is the response to the FINS command sent by the application at the beginning of this process.

6. After the response is converted, handleMessage sends the FINS response to the application. Sending the response is performed by the SerialUnit export function, sendFinsgw.

7. The SerialUnit function sendFinsgw sends the FINS response to the application based on the message specified by the ProtoDLL.

When the processing for one message is complete, the ProtoDLL handleMessage returns to the SerialUnit. The ProtoDLL processing is then finished. This completes the processing for a FINS message from the application:



4.4 Registry and Other System Settings

The SerialUnit operation settings use the registry. These registry values are normally set with the Setting Utility, fsnavim.exe SerialSetup.dll.

The registry key that specifies the SerialUnit operation is stored in the following location:

HKEY_LOCAL_MACHINE¥SOFTWARE¥OMRON¥FinsGateway¥NetworkProvider¥Serial

These keys have the following sub-keys:

Key Name	Meaning
Unit ¥SerialLineUnit	Sets the operation of the SerialUnit, itself
Proto	Sets the protocol operation
Proto¥*protocol name*	Sets the specified protocol operation
Lines	Sets the line operation
Lines¥*line name* (i.e., COM1)	Sets the specified line operation
Lines¥*line name*¥Parameters	Sets the specified line parameters (communication conditions, etc.)
Lines¥*line name*¥Nodes	Sets the nodes of the specified line
Lines¥*line name*¥Nodes¥*node number*	Sets the specified node operation

The following values are set to the registry entries of these keys. Only the main entries are shown here:

Unit ¥SerialLineUnit Entry Name	Meaning
Lines	Enumerates the SerialUnit line names. The lines shown here are the registry keys specified in Lines¥linename.

Proto¥ProtocolName Entry Name	Meaning
ProtoDLL	Specifies the full path of the ProtoDLL file. Environment variables can be used.

Lines¥LineName¥Nodes¥NodeNumber Entry Name	Meaning
Proto	Specifies the protocol used by this node. This value is the Proto¥protocolname in the registry.
ProtoSpecData	Stores protocol-specific data. Normally, it shows the actual protocol target address.
ProtoSpecSize	Indicates the size of the ProtoSpecData data, as a number of bytes.

4.5 Setting Utilities

The following two SerialUnit Setting Utilities are provided:

- SerialSetup.dll: this is the DLL called by fsnavim.exe, and is used to perform all the SerialUnit settings. This DLL cannot be changed from the ProtoDLL SDK.
- Serial Communication Explorer: this is the utility to check the communication conditions for communicating with the serial communication devices. This command can be changed from the ProtoDLL SDK.

4.6 Protocol Support Other than FINS

The SerialUnit and ProtoDLL have the capacity to support protocols other than the application FINS messages. For example, the SYSWAY protocol can be used as is by the application. However, the procedure for supporting protocols from the application other than FINS is not noted in the ProtoDLL SDK. It is not within the scope of the ProtoDLL SDK.

5 Development Environment

The following are required for ProtoDLL development:

- Development environment for DLL development
- FinsGateway SerialUnit
- FinsGateway SDK

The ProtoDLLs already provided to the SerialUnit (SYSWAY, SYSWAY-CV, CompoWay/F) were developed using Microsoft Visual C++. The programming language used is C-language, the library is the standard C library and Win32. The reason for not using C++, etc., or MFC, etc. is to maintain as much stability as possible.

The ProtoDLL is executed in the SerialUnit system service context. Consider the following points of system service design when designing the ProtoDLL:

- There is no user interface. Therefore, The ProtoDLL does not provide the user with error data or other information in the form of a dialog box.
- The means of providing error data are limited. The capacity for providing the user with data through a window is limited, so error data and other information is normally provided through the event log or UDM (Universal Data Monitor). Windows NT provides an event log. FinsGateway provides the UDM and the Windows95 event log.
- Common services are provided for multiple processes. In other words, it is not a good idea to design a service for a specific application. It is necessary to design the service thinking that varying applications will make asynchronous requests.
- A reliable design is the highest priority. The user is not normally aware of the operation of a service. A service built into the system generally keeps running after it is started, until the system shuts down.

For OLE/COM programming, the following point has proven to be important:

- COM instances are created for each user. A COM object design that assumes there is only ever one instance in the system can therefore encounter unexpected errors. This occurs when the system service and the logged on user are different.

6 ProtoDLL Development Precautions

There are several points that must be considered in the development of the ProtoDLL. There are multiple ProtoDLLs for one SerialUnit operation, and they use common resources to operate. It is therefore necessary to design each ProtoDLL so that it does not adversely affect any other ProtoDLL operation.

6.1 Common Resources

In the SerialUnit operation, there are common resources between the SerialUnitand the ProtoDLL. For example, one of the most important resources is the COM port. If one ProtoDLL changes the COM port settings, or closes the COM port, the previous settings must be restored before the ProtoDLL handleMessage is closed. The SerialUnitdoes not perform the COM port settings with each ProtoDLL handleMessage execution, so the changes made by one ProtoDLL to the COM port affect all the ProtoDLLs.

There is also data that is used in common between the SerialUnit and the ProtoDLLs that must not be changed by the ProtoDLLs. Refer to the Reference for details.

6.2 Other Protocols

One ProtoDLL handleMessage completely controls the SerialUnit operation. Until the handleMessage is closed, the SerialUnit performs no operations on its own, but simply provides the functions to be called by the handleMessage. Therefore, until the handleMessage processing is complete, other ProtoDLLs do not operate. As a result, from the perspective of the application, the SerialUnit only supports one protocol.

The SerialUnit protocol addition model executes one handleMessage for each application message, so SerialUnit operation supporting only one protocol differs from past models. It is best to consider whether the system really requires a ProtoDLL that controls the SerialUnit in this manner.

6.3 Troubleshooting and Precautions

The ProtoDLL developer must take sufficient precaution for handling potential errors. SerialUnit communication involves quite a large number of elements. When an error occurs, it can therefore be quite challenging to troubleshoot the system. For this reason, it is best to design means of troubleshooting into the ProtoDLL from the beginning.

The event log and UDM are two of the best tools for troubleshooting, and debugging. Using these effectively should be included in the original design of the ProtoDLL. The following operations must be considered carefully:

- Messages sent from the application
- Messages sent to/from the COM port
- Calls for the SerialUnit export function
- Calls from the SerialUnit for the ProtoDLL export function
- Protocol conversion (send/receive)
- Target device communication procedure

7 Event Log and UDM

An event log corresponding to Win32 is used to report errors detected by the NSP. The NSP status is monitored by the UDM (Universal Data Monitor). The event log has a system-wide, common record area which stores the logs reported from the various sources. This is generally used for recording the OS-level error states. The UDM has record area divided into selectable categories. Each record area stores the logs reported by the source specified for that area. This is generally used for tracing the operation of the specified source. As an example of using the event log and the UDM, the SerialUnit notifies the event log of operation failures. The data being sent through the SerialUnit is being traced by the UDM.

Using the event log and UDM to report is not a necessity of the NSP. Implementation of this function is optional. However, for system service debugging, or troubleshooting, implementation of this log function is recommended.

7.1 Event Log

Errors that occur during NSP operation need to be notified to the user. However, since the NSP operates as a system service, notifying the user of errors with a window is not always appropriate. For the NSP to notify the user, the Win32 event log is used.

For an event log model and operation details, refer to the Win32 Event Logging OverView. The event log characteristics in brief are as follows:

- Dividing the events broadly, they can be classified as System, Security, and Application.
- The event types are Errors, Warnings, and Information.
- The exact event message expressions are included in the files including the resources compiled by the Message Compiler (MC).

7.2 UDM

The UDM was designed as a general data monitor, and is exclusive to FinsGateway. Using the UDM is helpful for recording changes in the NSP status, debugging, and tracing execution during system operation. The UDM characteristics in brief are as follows:

- There is a log file for each category.
- When logging, the LDH (Log Data Handler) filter DLL can be specified.
- The detailed log display can be done with the specified LDH conversion DLL.
- Log start/stop can be specified from an application (normally a viewer application).

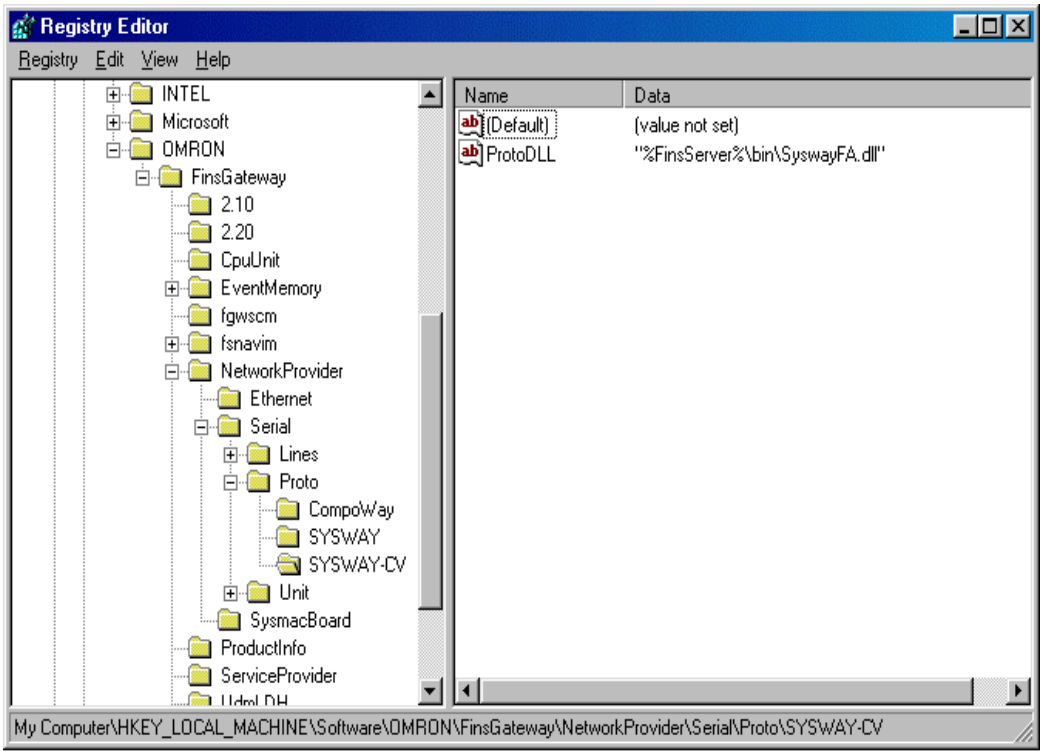
8 Installation

The ProtoDLL developed can be used from any folder. The SerialUnit searches for the DLL file in the registry fixed entry value, and loads it. The registry location to specify the DLL file is as follows:

HKEY_LOCAL_MACHINE¥SOFTWARE¥OMRON¥FinsGateway¥NetworkProvider¥Serial¥Proto¥ProtocolName

The accurate DLL file name is specified in ProtoDLL, as the character string value under this registry entry (REG_STRING or REG_EXPAND_STRING value). An environment variable can also be used.

For example, if a SYSWAY ProtoDLL file is “%FinsServer%¥bin¥SyswayFA.dll”, it would be specified as follows in the registry configuration:



9 Debugging

Debugging the ProtoDLL uses the SerialUnit as a host application. The SerialUnit program file is “%FinsServer%¥bin¥SeriUnit.exe”.

In normal systems, SeriUnit.exe operates as a system service. However, for debugging, it is best to run SeriUnit.exe alone. If SeriUnit.exe is implemented not to operate as a service, but to be executed by double-clicking with the mouse, or specifying the command name, it can be operated as a console application and debugging is much simpler.

Therefore, for debugging the ProtoDLL, set SeriUnit.exe as a debug execution program in the ProtoDLL development environment (Microsoft Visual Studio, etc.).

When SeriUnit.exe is started, an MS-DOS prompt or command prompt will be displayed. When it is started in this manner, operations as a service become invalid, and starting/stopping from the service manager is impossible.

Press Ctrl+C on the keyboard to stop SeriUnit.exe.

The following precautions apply to debugging:

- When force-stopping SeriUnit.exe in Windows95 (other than by using Ctrl+C), the process to unload the DLL used by SeriUnit.exe is not properly called. As a result, the FinsGateway unit remains in the used state, and SeriUnit.exe cannot be started the next time. In this case, close all the applications using FinsGateway.
- The startup of SeriUnit.exe implemented as a service for Windows NT takes about 5 seconds. The ProtoDLL is loaded after this time.
- When starting SeriUnit.exe as a console application, do not use the service manager to start or stop the SeriUnit.exe.

10 Utility Software

The FinsGateway SerialUnit includes utility software. The points described in this chapter must be considered for this program.

10.1 Serial Communication Explorer Addition

The Serial Communication Explorer is a utility to check and display the communication conditions for the serial communication devices. Serial communication requires that the cable connections, communication conditions (transfer speed, character size, etc.), etc. match between the two devices. It is therefore necessary to confirm the communication conditions in the system. The Serial Communication Explorer stores the combination of the command and the response for each device when it checks the communication conditions.

The devices supported by the Serial Communication Explorer cannot be added dynamically, but the code can be changed, and a command and response can be added to check for a specific device.

The following are suggestions for adding new devices:

- New devices are implemented inheriting the virtual class, CvirtualDevice. The CvirtualDevice virtual class has true virtual functions already defined, so the inherited actual device class needs to over-ride these virtual functions.
- After implementing a class to represent the new device, create the one and only instance in the program. Create the instance by including the following in the file, Devices.cpp.

```
#include “StdAfx.h”
#include “devices.h”

CDeviceSysmac sysmac;
CDeviceThermoE5CK e5ck;
CDeviceCompoWay compoway;
//Add the new device class instance here.
CdeviceUnknow unknownDevice;
```

- In order to make the new device instance accessible by the GetDeviceArray function of the CvirtualDevice virtual class, place the instance address in a global array in the Devices.cpp file. This is done as follows:

```
static CVirtualDevice* gDevices[] =
{
        &sysmac,
        &e5ck,
        &compoway,
        //Add the new device class instance here.
        &unknownDevice,
        NULL
};
```

This completes the addition of a new device. Build the Serial Communication Explorer to confirm that the added device is included in the list of selectable devices.

11 Reference

The following is a list of the necessary data, functions for developing a ProtoDLL. For details on the event log data, refer to the online fsport-related reference.

11.1 Data Structure Reference

Data Structure	Function
tProtoDllInfo	Shows the data structure that stores the ProtoDLL data
tProtoDllProperty	Indicates the ProtoDLL properties
tComEnvPublic	Communicates the SerialUnit communication environment to the ProtoDLL
SerialSharedData	Stores the send/receive data
NODEINFO	Stores the node data of the target device

11.1.1 Data Structure of tProtoDllInfo

The data structure, tProtoDllInfo is a data structure that shows ProtoDLL data.

```
typedef struct _tagProtoDllInfo {
        HANDLE      handle;                                 //DLL handle
        CHAR        definedName[MAX_PATH];                  //DLL defined Name    (example: “HighLink”)
        CHAR        fileName[MAX_PATH];                     //DLL Filename (example: “c:¥HighLink.dll”)
        pProtoDllProperty protoProperty;
        BOOL        (*comNew)    () ;
        BOOL        (*comDelete)    () ;
        BOOL        (*comWriteFail)    () ;
        BOOL        (*comReadFail)    () ;
        BOOL        (*comFINS command)    () ;
        BOOL        (*comFINS response)    () ;
        BOOL        (*comMakeFrame)    () ;
        BOOL        (*comCheckFrame)    () ;
        BOOL        (*comGetWriteMessage)    () ;
        pProtoDllProperty           (*comQueryProperty)    () ;
        BOOL        (*comQueryDirection)    () ;
        BOOL        (*comCompleteDirection)    () ;
        tDllExportProcs         dllProcs;
        tExeExportProcs         exeProcs;
} tProtoDllInfo, *pProtoDllInfo;
```

Members

handle

ProtoDLL module handle. This value cannot be changed in the ProtoDLL.

definedName

ProtoDLL definition name. This value cannot be changed in the ProtoDLL.

fileName

ProtoDLL file name. This value cannot be changed in the ProtoDLL.

protoProperty

This is the data structure that shows the ProtoDLL properties. The ProtoDLL must have the values set in this data structure for newDll processing.

comNew

This is a function of an older version of ProtoDLL. Do not use it.

comDelete

This is a function of an older version of ProtoDLL. Do not use it.

comWriteFail

This is a function of an older version of ProtoDLL. Do not use it.

comReadFail

This is a function of an older version of ProtoDLL. Do not use it.

comFINScommand

This is a function of an older version of ProtoDLL. Do not use it.

comFINSresponse

This is a function of an older version of ProtoDLL. Do not use it.

comMakeFrame

This is a function of an older version of ProtoDLL. Do not use it.

comCheckFrame

This is a function of an older version of ProtoDLL. Do not use it.

comGetWriteMessage

This is a function of an older version of ProtoDLL. Do not use it.

comQueryProperty

This is a function of an older version of ProtoDLL. Do not use it.

comQueryDirection

This is a function of an older version of ProtoDLL. Do not use it.

comCompleteDirection

This is a function of an older version of the ProtoDLL. Do not use it.

dllProcs

This is the data structure that shows the functions that the ProtoDLL exports to the SerialUnit. The ProtoDLL must have the values set in this data structure for newDll.

exeProcs

This is the data structure that shows the functions that the SerialUnit exports to the ProtoDLL. The ProtoDLL needs to call the functions shown in these data structure values when executing the SerialUnit processing.

Remarks

tProtoDllInfo is a data structure that shows ProtoDLL data.

Within this member, the protoProperty and dllProcs require that the ProtoDLL set the values for the ProtoDLL function, newDll.

For the ProtoDLL the member, exeProcs is especially important. The function addresses that the SerialUnit exports to the ProtoDLL are all set in this member. Therefore, the ProtoDLL uses the function address values from this member to call the SerialUnit functions.

See Also

tProtoDllProperty, newDll

11.1.2 Data Structure of tProtoDllProperty

The data structure, tProtoDllProperty is the ProtoDLL property data structure.

```
typedef     struct      {
            LARGE_INTEGER       protoTypeMask;                  //supported protocol type
            const       DWORD   comSendLenMax;                  // send command maximum length
            const       DWORD   comReceiveLenMax;               // receive response maximum length
            const       BYTE    ProtoName[PROTONAMELEN];        // protocol name
            const       BYTE    suportControllerNames[MAXCNTL][PROTONAMELEN];
                                                                // support controller name
} tProtoDllProperty, *pProtoDllProperty;
```

Members

protoTypeMask

Shows the protocol types supported by the ProtoDLL. The protocol support shown here is the protocol used by the application, not the actual communication protocol. Normally, the ProtoDLL supports only the FINS protocol. To support a protocol for the application other than the FINS protocol, this member in the ProtoDLL takes a fixed numeric value for that protocol. The fixed protocol number is determined by the SerialUnit developer (Omron), the ProtoDLL developer cannot determine the value.

comSendLenMax

Shows the protocol maximum command length. The present SerialUnit does not use this value.

comReceiveLenMax

Shows the protocol maximum response length. The present SerialUnit does not use this value.

ProtoName

Shows the protocol name. This must be the same as the registry Serial¥Proto key name.

suportControllerNames

Shows the controller names supported by the protocol. The present SerialUnit does not use this value.

Remarks

tProtoDllProperty is the ProtoDLL property data structure.

The ProtoDLL must have the tProtoDllProperty value of the tProtoDllInfo data structure set for newDll processing.

The member protoTypeMask value setting is performed with the following code:

```
#include    <Largeint.h>

static      tProtoDllProperty       gProtoDllProperty;
//setup ProtoType in supporting
{
LARGE_INTEGER largeInt = ConvertUlongToLargeInteger (1) ;
gProtoDllProperty.protoTypeMask = LargeIntegerShiftLeft (largeInt, C_PROTOCOL_FINS) ;
}
```

See Also

tProtoDllInfo, newDll

11.1.3 Data Structure of tComEnvPublic

The data structure, tComEnvPublic is the construct that stores the SerialUnit communication environment data.

```
typedef struct TagComEnvPublic {
        char        LogicName[NAMELEN];   // COM port logical name
        char        ComName[NAMELEN];     // COM device name
        HANDLE      SerialHandle;                 // serial driver handle value
        HNET        FinsGWHandle;                 // FinsGateway network handle value
        FINSENV     FinsEnv;                      // FINS communication environment management construct
} tComEnvPublic, *pComEnvPublic;
```

Members

LogicName

Stores the character string that shows the SerialUnit logical line name. The character string is terminated with NULL.

ComName

Stores the character string that shows the physical device name of the COM port used by the SerialUnit (i.e., COM1, etc.). The character string is terminated with NULL.

SerialHandle

This is the COM port access file handle. It is normally the handle obtained by Win32 CreateFile. Howeever, when operating on a telephony device, it becomes a useful handle for TAPI. In this case, the processing used for a handle obtained using CreateFile may be invalid.

FinsGWHandle

This is the FinsGateway FINS message communication handle. The SerialUnit and ProtoDLL can use this handle to communicate with the application.

FinsEnv

Data structure to indicate the SerialUnit FinsGateway communication unit settings. This data structure is as follows:

```
typedef struct          FinsEnvironment {
        DWORD   NodeAddr;               // unit FINSnode address
        DWORD   UnitAddr;               // unit FINSunit address
        DWORD   FinsDataLength;         // communication data length for FinsGateway
} FINSENV, *FINSENVP;
```

Remarks

This data structure instance is created for each SerialUnit communication unit. Therefore, if the SerialUnits are using two COM ports, the SerialUnits will create two tComEnvPublic instances. These multiple tComEnvPublic instances do not interfere with each other.

However, a single tComEnvPublic instance is not thread-safe. In other words, do not create multiple threads throughout the ProtoDLL, and simultaneously operate a tComEnvPublic instance. Instead, use exclusive control in the ProtoDLL for that situation.

Be careful also of the fact that the tComEnvPublic instances are a shared resource between multiple ProtoDLLs. For example, if one ProtoDLL closes the COM port, and ends its processing, other ProtoDLLs can no longer communicate through the COM port, either.

See Also

openComDefault, closeCom

11.1.4 Data Structure of SerialSharedData

The SerialSharedData data structure is the area that stores the data shared by the SerialUnit and ProtoDLL.

```
typedef struct          _tagSerialSharedData {
        FINSHEAD                finsHead;               // FINS header management construct

        BYTE    *FINS command;                  // FINS command buffer start address
        DWORD   FINS commandLen;                // FINS command length (bytes)
        DWORD   FINS commandLenMax;             // FINS commandmaximum length (bytes)

        BYTE    *FINS response;                 // FINS response buffer start address
        DWORD   FINS responseLen;               // FINS response length (bytes)
        DWORD   FINS responseLenMax;            // FINS response maximum length (bytes)

        BYTE    *comSendData;                   // COM send data buffer start address
        DWORD   comSendDataPos;                 // COM send data stored position
        DWORD   comSendLenMax;                  // COM send data maximum length

        BYTE    *comReceiveData;                // COM receive data buffer start address
        DWORD   comReceiveDataPos;              // COM receive data stored position
        DWORD   comReceiveLenMax;               // COM receive data maximum length

        BYTE    *comWorkData;                   // COM WORK data buffer start address
        DWORD   comWorkDataPos;                 // COM WORK data stored position
        DWORD   comWorkLenMax;                  // COM WORK data maximum length

        BYTE    *CurrentControllerName;  // target model name
        DWORD   CurrentRetryTimes;              // current number of retires
        DWORD   CurrentTimeout;                 // current packet timeout time (msec)

} SerialSharedData, *pSerialSharedData;
```

Members

finsHead

Stores the FINSHEAD data used for communication with FinsGateway. This data initially contains the FINSHEAD from when the SerialUnit receives data from the application. The application data (application FINS address, etc.) that sent a command can be obtained from this data.

FINScommand

Stores the FINS command sent by the application. This is the data from the FINS command MRC. The command length is shown in the FINScommandLen member.

FINScommandLen

Stored the length of the FINS command sent by the application. This shows the data length of the FINScommand member.

FINScommandLenMax

Stores the maximum number of bytes of the FINScommand member. The ProtoDLL must not change this value.

FINSresponse

Stores the FINS response to send to the application. This is the data from the FINS response MRC. The command length is shown in the FINSresponseLen member.

FINS responseLen

Shows the length (number of bytes) of the FINS response to send to the application.

FINS responseLenMax

Shows the number of bytes of the area storing the FINSresponse member. The ProtoDLL must not change this value.

comSendData

Stores the data to send to the COM port. The data length is shown in the comSendDataPos member.

comSendDataPos

Shows the number of bytes of data to send to the COM port.

comSendLenMax

Shows the number of bytes of the comSendData member. The ProtoDLL must not change this value.

comReceiveData

Stores the data received from the COM port. The number of data is shown in the comReceiveDataPos member.

comReceiveDataPos

Shows the number of bytes of data received from the COM port.

comReceiveLenMax

Shows the maximum size (number of bytes) of the comReceiveData member. The ProtoDLL must not change this value.

comWorkData

Temporarily stores the data received from the COM port. The number of data is shown in the comWorkDataPos member.

comWorkDataPos

Shows the number of bytes of data stored in the comWorkData member.

comWorkLenMax

Shows the maximum size (number of bytes) of the comWorkData member. The ProtoDLL must not change this value.

CurrentControllerName

Stores the target node controller name of the message being processed as a character string.

CurrentRetryTimes

Stores the number of retries for sending.

CurrentTimeout

Stores the timeout time of the message being processed.

Remarks

SerialSharedData follows to data flow ddescribed below, and is used to share each type of data between the SerialUnit and the ProtoDLL:

1. The application sends the SerialUnit a FINS command.
2. The SerialUnit stores the command it receives in the finsHead member and the FINScommand member, and passes the processing on to the appropriate ProtoDLL.
3. The ProtoDLL uses the FINS command to create the data to send to the actual COM port and stores it in the comSendData member.
4. The ProtoDLL receives the data from the target device. The receive data is stored temporarily in the comWorkData member, and after the necessary packet processing it is stored in the comReceiveData member.
5. The ProtoDLL uses the response data it received to create the FINS response, stores it in the FINSresponse member, and sends it to the application as a response.

The data stored in the SerialSharedData member is valid for one cycle of the processing described above. In other words, after the ProtoDLL completes the processing of the application message and passes excution to the SerialUnit, the SerialSharedData value is irrelevant.

This data structure instance is created for each SerialUnit communication unit. Therefore, if there are two SerialUnits using two COM ports, the SerialUnits will create two SerialSharedData instances. These multiple SerialSharedData instances do not interfere with each other.

However, a single SerialSharedData instance is not thread-safe. In other words, do not create multiple threads throughout the ProtoDLL, and simultaneously operate a SerialSharedData instance. Instead, use exclusive control in the ProtoDLL for that situation.

See Also

openComDefault, closeCom

11.1.5 Data Structure of NODEINFO

The NODEINFO data structure stores the node-related data that shows the communication target.

```
typedef    struct     NodeInformation {
           DWORD      FinsNode;                  // FINS node address
           char       Proto[NAMELEN];            // protocol type
           char       Controller[NAMELEN];                  // model name
           DWORD      SpecSize;                  // protocol specification data size
           BYTE       SpecData[SPECMAX];                    // protocol specification data
           pProtoDllInfo dllInfo;                // ProtoDLL Information
           VOID*      SpecPtr1;                  //ProtoDLL Specific Pointer
} NODEINFO, *NODEINFOP;
```

Members

FinsNode

This is the FINS node address. It cannot be changed by the ProtoDLL.

Proto

This is a character string that shows the node protocol type. It cannot be changed by the ProtoDLL.

Controller

This is a character string that shows the node controller model name. It cannot be changed by the ProtoDLL.

SpecSize

This shows the size of the data (number of bytes) in the SpecData member. It cannot be changed by the ProtoDLL.

SpecData

This shows the protocol-depenedent data allocated to each node. For most protocols, this data is the protocol-based unit number. It cannot be changed by the ProtoDLL.

dllInfo

This shows the ProtoDLL data corresponding to the node. It cannot be changed by the ProtoDLL.

SpecPtr1

This is a pointer that can attach data to a node. Its purpose is left open. However, the ProtoDLL does not normally nbeed to use this member.

Remarks

NODEINFO stores data about the communication target node. This data stores the data written in the registry by the Setting Utility SerialSetup.dll.

In the SerialUnit communication model, a serial line device is allocated to a FINS network node. A protocol is set for each node. The SpecData member obtains and stores the setting data from the registry required by the ProtoDLL for each node.

The FINS network node address and the serial line address do not necessarily always match. For example, in SYSMAC WAY, the SYSMAC WAY unit number and the FINS node address do not always match. In this case, it is valid to store the serial line address for actual communication in the SpecData member. By doing this, a serial line address can be attached to a node, and the ProtoDLL can change the communication target device on the line for each node.

See Also

openComDefault, closeCom

11.2 ProtoDLL Export Function Reference

Function Name	Function
newDll	ProtoDLL initial processing
handleMessage	Application message processing
deleteDll	ProtoDLL close processing

11.2.1 newDll

newDll performs the ProtoDLL initial processing.

```
BOOL WINAPI    newDll (
               pProtoDllInfo           pdi,
               HKEY                    subKey
               )
```

Parameters

pdi

This shows the data structure that stores the ProtoDLL data.

subKey

This is the already open handle of the registry key for the ProtoDLL. The registry key is Serial¥Proto¥protocolname. The ProtoDLL can use this handle to operate the registry entry. Do not close this handle with the ProtoDLL.

Return Value

Must return TRUE when the processing is successful. If FALSE is returned, the SerialUnit will not use this ProtoDLL.

Remarks

NewDll is executed immediately after the SerialUnit loads the ProtoDLL. The ProtoDLL performs the following initial processing, and then must provide the SerialUnit the necessary data.

- Sets the addresses of the ProtoDLL export functions to pdi->dllProcs. The SerialUnit executes the functions set to dllProcs, after newDll.
- Sets the ProtoDLL properties to pdi->protoProperty. The SerialUnit uses the protoProperty data to select the appropriate ProtoDLL when there is more than one.

The symbol name that newDll exports must be newDll. The symbol newDll must also be publicized in the module definition file (.DEF).

See Also

tProtoDllInfo

11.2.2 handleMessage

handleMessage processess the messages from the application.

```
BOOL WINAPI
handleMessage (
               pProtoDllInfo           pdi,
               HCOMENV                 hComEnv,
               pSerialSharedData       pShd,
               NODEINFOP               nodep
               )
```

Parameters

pdi

Indicates the data structure that stores the ProtoDLL data.

hComEnv

SerialUnit communication environment data handle

pShd

Specifies the SerialUnit and ProtoDLL communication data structure instance/pointer.

nodep

Provides the communication target FINSnode data.

Return Value

Returns TRUE when successful.

Remarks

handleMessage is the function that the ProtoDLL performs to process messages from the application. This is one of the most important processes for the ProtoDLL. The SerialUnit and the ProtoDLL process the messages from the application as follows:

1) The application sends a message. If the message is to a serial line device, it is received by the SerialUnit.
2) When the SerialUnit receives a message from the application, it obtains the protocol registered in the registry for that node address, and looks for the appropriate ProtoDLL.
3) After locating the appropriate ProtoDLL, the SerialUnit calls the handleMessage function of the ProtoDLL, and transfers the message processing to the ProtoDLL.
4) The ProtoDLL handleMessage interprets the message, and converts it to the protocol needed to be sent on the serial line.
5) The ProtoDLL uses the SerialUnit export function comWrite, and sends the converted data to the COM port.
6) The ProtoDLL uses the SerialUnit export function comRead to the read the target device response data from the COM port.
7) The ProtoDLL interprets the response data, and converts it to a FINS response to send to the application.
8) The ProtoDLL uses the SerialUnit export function sendFinsgw to send the message to the application.
9) When one full cycle of application message processing is complete, the ProtoDLL returns from the handleMessage function, and returns processing to the SerialUnit.

The SerialUnit does not play any main role until the ProtoDLL handleMessage is returned. Therefore, the ProtoDLL functions are able to perform a lot of protocol processing. However, it is necessary to rememebr that during a long handleMessage process, other protocols cannot use the SerialUnit, either.

See Also

11.2.3 deleteDll

deleteDll performs the ProtoDLL exit processing.

```
VOID WINAPI deleteDll (
pProtoDllInfo               pdi
            )
```

Parameters

pdi

Indicates the data structure that stores the ProtoDLL data.

Return Value

No return value.

Remarks

deleteDll performs the ProtoDLL exit processing.

This function is called by the SerialUnit just before the ProtoDll is unloaded.

See Also

tProtoDllInfo

11.3 SerialUnit Export Function Reference

The following is the list of SerialUnit I/F for the ProtoDLL:

Function Name	Function
getComEnv	Obtains the SerialUnit I/F data structure ComEnv
openCom	Opens the COM port (not implemented)
openComDefault	Opens the COM port with the default values
closeCom	Closes the COM port
writeCom	Sends data to the COM port
readCom	Receives data from the COM port
getComDcb	Obtains the COM port DCB
setComDcb	Sets the COM port DCB
checkSerialEvent	Waits for an event from the COM port
startComXfr	Performs the COM port communication initial processing
sendFinsgw	Sends to FinsGateway
receiveFinsgw	Receives from FinsGateway
sendFinsErrorResponse	Sends an error response to FinsGateway
makeErrFinsFrame	Generates a FINS error response
putFINS logFromWin32	Reports the FINS communication unit error log

11.3.1 getComEnv

getComEnv obtains the data from the SerialUnit communication environment handle.

```
pComEnvPublic WINAPI ExeProcs_getComEnv (
HCOMENV hComEnv                    //HANDLE of ComEnv (COM port Environment)
        ) ;
```

Parameters

hComEnv

SerialUnit communication environment data handle. This handle is passed on as a parameter for the ProtoDLL message handler function, handleMessage.

Return Value

This is the pointer to the tComEnvPublic data structure taken from the handle. The data structure shared by the ProtoDLL and the SerialUnit is stored in tComEnvPublic.

Remarks

The SerialUnit uses the tComEnvPublic data structure to tell the ProtoDLL the communication environment. This data structure is passed to the SerialUnit in the handle parameter of the ProtoDLL message handle function handleMessage. The ProtoDLL takes the pointer to the actual data structure from this handle using the getComEnv function, and accesses the data structure.

See Also

handleMessage, tComEnvPublic

11.3.2 openCom

At present, openCom only has a function entry, and performs no actual operation.

```
BOOL WINAPI ExeProcs_openCom (
        )
```

Parameters

None.

Return Value

Always TRUE.

Remarks

openCom is not the function name, and does not perform any processing at present. It always returns TRUE.

To open the COM port, use one of the following procedures:

1) Open the COM port using the openComDefault function. This opens the COM port with the SerialUnit default settings, and attaches the port to the SerialUnit.

2) Open the COM port using the Win32 function. This opens the COM port for the exclusive use of the ProtoDLL.

See Also

openComDefault, closeCom

11.3.3 openComDefault

openComDefault opens the COM port allocated to the SerialUnit in its default state, and attaches it to the SerialUnit.

```
BOOL WINAPI ExeProcs_openComDefault (
HCOMENV  hComEnv                    // COM port management construct pointer
         )
```

Parameters

hComEnv

Specifies the SerialUnit communication environment data handle.

Return Value

If the operation was successful, it returns TRUE; if it fails, it returns FALSE.

Remarks

openComDefault opens the COM port. When the SerialUnit first calls the ProtoDLL, the COM port is already open. Therefore, the ProtoDLL odes not normally need to open or close the COM port.

However, the ProtoDLL design may be such that it is advatageous to open or close the COM port from the ProtoDLL. If the ProtoDLL is opening/closing the COM port, the ProtoDLL needs to use openComDefault to open the COM port in the previous state it was being used by the SerialUnit after each message processing, and before starting the next message processing. The next message may not need to be processed by the same ProtoDLL, and one ProtoDLL is not to affect the operation of any other ProtoDLL.

If the ProtoDLL is not going to close the COM port or change any of the COM port settings, openComDefault is not needed.

Using openComDefault to open the COM port to the default state means using the communication conditions of the SerialSetup.dll.

See Also

closeCom

11.3.4 closeCom

closeCom closes the COM port allocated to the SerialUnit.

```
VOID WINAPI ExeProcs_closeCom (
HCOMENV  hComEnv                    // COM port management construct pointer
         ) ;
```

Parameters

hComEnv

Specifies the SerialUnit communication environment data handle.

Return Value

No return value.

Remarks

closeCom closes the COM port. When the SerialUnit first calls the ProtoDLL, the COM port is already open. Therefore, the ProtoDLL odes not normally need to open or close the COM port.

However, the ProtoDLL design may be such that it is advatageous to open or close the COM port from the ProtoDLL. If the ProtoDLL is opening/closing the COM port, the ProtoDLLneeds to use closeCom to close the COM port.

closeCom does not only close the COM port, but also maintains consistency with the other management data. Therefore, the ProtoDLL must use closeCom to close the COM port.

If the ProtoDLL is closes the COM port with closeCom, the same ProtoDLL needs to use openComDefault to open the COM port in the previous state it was being used by the SerialUnit after each message processing, and before starting the next message processing.

If the ProtoDLL is not going to close the COM port or change any of the COM port settings, closeCom is not needed.

See Also

openComDefault

11.3.5 writeCom

writeCom sends data to the COM port.

```
DWORD WINAPI ExeProcs_writeCom (                    //return: Win32 ErrorCode (see GetLastError)
        HCOMENV             hComEnv,    // COM port management construct pointer
        pSerialSharedData   pShd,       // packet management data
        LPDWORD  numOfBytesWritten
        ) ;
```

Parameters

hComEnv

Specifies the data structure handle showing the SerialUnit communication environment.

pShd

Specifies the SerialUnit and ProtoDLL communication data structure instance pointer.

numOfBytesWritten

The number of data bytes written to the COM port is stored and returned.

Return Value

A Win32 error code is returned. For details about this value, refer to Win32 GetLastErrorfunction.

Remarks

writeCom sends data to the COM port.

The data to send is pShd->comSendData. The size of the data to send is the value in pShd->comSendDataPos. Before the ProtoDLL calls writeCom, it must set these pShd member values.

The number of data bytes actually sent to the COM port is stored in the numOfBytesWritten parameter.

The data sent by this function is automatically logged in the UDM.

See Also

11.3.6 readCom

readCom receives data from the COM port.

```
BOOL WINAPI ExeProcs_readCom (
        HCOMENV             hComEnv,    // COM port management construct pointer
        pSerialSharedData   pShd        // packet management data
        ) ;
```

Parameters

hComEnv

Specifies the data structure handle to indicate the SerialUnit communication environment.

pShd

Specifies the SerialUnit and ProtoDLL communication data structure instance pointer.

Return Value

The Win32 ReadFile result is returned.

Remarks

readCom receives data from the COM port. Normally, it receives data after detecting an event from the COM port using checkSerialEvent.

readCom first clears pShd->comWorkData. After that, it uses Win32 ReadFile toreceive data from the COM port, stores that data in pShd->comWorkData, and stores the number of bytes received in pShd->comWorkDataPos.

The COM port will not necessarily always be able to receive the total expected number of packets at once. Most responses are received in multiple packets.

The ProtoDLL must execute readCom as many times as necessary, and build the packets for that protocol. In other words, the ProtoDLL response receive processing is to copy the data received in pShd->comWorkData to pShd->comReceiveData.

The data received by this function is automatically logged in the UDM.

See Also

checkSerialEvent

11.3.7 getComDcb

getComDcb obtains the data structure DCB, which indicates the COM port settings.

```
BOOL WINAPI ExeProcs_getComDcb (
HCOMENV hComEnv,
LPDCB   dcb
        ) ;
```

Parameters

hComEnv

Specifies the data structure handle showing the SerialUnit communication environment.

LPDCB

Stores the obtained DCB.

Return Value

The Win32 SetCommState result is returned.

Remarks

getComDcb obtains the data structure DCB stipulated by Win32, that shows the COM port settings of the COM port opened by the SerialUnit.

See Also

setComDcb

11.3.8 setComDcb

setComDcb sets the data structure DCB, which indicates the COM port settings.

```
BOOL WINAPI ExeProcs_setComDcb (
HCOMENV hComEnv,
LPDCB   dcb
) ;
```

Parameters

hComEnv

Specifies the data structure handle showing the SerialUnit communication environment.

LPDCB

Stores the DCB to set.

Return Value

The Win32 SetCommState result is returned.

Remarks

setComDcb sets the data structure DCB stipulated by Win32, that shows the COM port settings of the COM port opened by the SerialUnit.

This setting enables changing all the COM port settings from the ProtoDLL.

See Also

getComDcb

11.3.9 checkSerialEvent

checkSerialEvent waits for a COM port I/O event.

```
BOOL WINAPI ExeProcs_checkSerialEvent (
        HCOMENV             hComEnv,    // COM port management construct pointer
        pSerialSharedData   pShd,               // packet management data
        DWORD               fdwEvtMask,// mask to distinguish the event to make valid
                                                    (see SetCommMask (win32))
        DWORD               *pEvent             // serial communication event area
        ) ;
```

Parameters

hComEnv

Specifies the data structure handle showing the SerialUnit communication environment.

pShd

Specifies the SerialUnit and ProtoDLL communication data structure instance pointer.

fdwEvtMask

Specifies the type of event to detect. This is the same as the Win32 SetCommMask mask value.

pEvent

Specifies the area to store the generated event. The value to store is the same as the Win32 WaitCommEvent.

Return Value

If an event is generated, TRUE is returned. If no event is generated within the time set for the timeout, FALSE is returned.

Remarks

checkSerialEvent waits for an event from the COM port. While waiting for an event, this function blocks thread processing. If an event is generated, or the time specified in pShd->CurrentTimeouttime elapses without an event being generated, this function ends.

Receiving from the SerialUnit COM port uses asynchronously overlapped IO. The ProtoDLL COM port receive processing is as follows:

1. Uses checkSerialEvent to confirm whether there is data to be received from the COM port.

2. Uses readCom to receive the data when an event is generated to indicate that data is being received.

3. Repeats the first two steps until the target device response data is complete.

See Also

readCom, SetCommMask (Win32), WaitCommEvent (Win32)

11.3.10 startComXfr

startComXfr initializes COM port communication.

```
BOOL WINAPI ExeProcs_startComXfr (
        HCOMENV             hComEnv,    // COM port management construct pointer
        pSerialSharedData   pShd,       // packet management data
        DWORD fdwAction                 //PurgeComm execution (see PurgeComm (win32))
        ) ;
```

Parameters

hComEnv

Specifies the data structure handle showing the SerialUnit communication environment.

pShd

Specifies the SerialUnit and ProtoDLL communication data structure instance pointer.

fdwAction

Specifies the action to give to Win32 PurgeComm.

Return Value

Always returns TRUE in the current implementation.

Remarks

startComXfr initializes COM port communication. This function is automatically executed before passing the message processing from the SerialUnit to the ProtoDLL.

If the ProtoDLL need to perform the COM port communication initialization for some reason, this function can be called. startComXfr performs the following processing:

- Clears the I/O buffer of the COM port using Win32 PurgeComm.
- Clears the SerialSharedData data structure comSendData and comReceiveData.
- Sets the SerialSharedData data structure CurrentRetryTimes to 0.
- Sets the SerialSharedData data structure CurrentTimeout to the timeout value set in the registry.

See Also

PurgeComm (Win32)

11.3.11 sendFinsgw

sendFinsgw sends FinsGateway FINS messages.

```
int WINAPI ExeProcs_sendFinsgw (    //return: send data length (see.Fins_sendData)
        HCOMENV             hComEnv,    // COM port management construct pointer
        pSerialSharedData   pShd        // packet management data
        ) ;
```

Parameters

hComEnv

Specifies the data structure handle showing the SerialUnit communication environment.

pShd

Specifies the SerialUnit and ProtoDLL communication data structure instance pointer.

Return Value

Returns the number of data bytes. This is the same as the Fins_sendData return value.

Remarks

sendFinsgw sends FINS messages through FinsGateway.

This function sends the data stored in the pShd->FINSresponse to the target specified in pShd->finsHead.

Normally, pShd->finsHead contains the FINSHEAD data that the SerialUnit received from the application. However, this first data is FINS command data, and cannot be used to send the response to the application with sendFinsgw (because the target network is the SerialUnit). To return a FINS response to the application, the pShd->finsHead must be modified for a response. To convert it for a response, FinsHead_composeResponse is generally used.

Data sent by this function is logged in the UDM.

See Also

11.3.12 receiveFinsgw

receiveFinsgw receives FinsGateway FINS messages.

```
int WINAPI ExeProcs_receiveFinsgw ( //return: receive data length (see: Fins_receiveData)
        HCOMENV             hComEnv,    // COM port management construct pointer
        pSerialSharedData   pShd,       // packet management data
        DWORD               dwTimeLimit //timeout time
        ) ;
```

Parameters

hComEnv

Specifies the data structure handle showing the SerialUnit communication environment.

pShd

Specifies the SerialUnit and ProtoDLL communication data structure instance pointer.

dwTimeLimit

Specifies the FINS message receive timeout value in milliseconds.

Return Value

Returns the number of data bytes. This is the same as the Fins_receiveData return value.

Remarks

receiveFinsgw receives FINS messages through FinsGateway.

This function waits for a FINS message to be received from FinsGateway, and stores the data in the pShd->finsHead and pShd->FINS commands when it is received.

Normally, a ProtoDLL only receives one FINS message from an application, and that message is received automatically through the SerialUnit. The ProtoDLL does normally need to receive another FINS message from the application.

Data received by this function is logged in the UDM.

See Also

11.3.13 sendFinsErrorResponse

sendFinsErrorResponse sends FINS error responses to the application.

```
int WINAPI ExeProcs_sendFinsErrorResponse (     //return:
        HCOMENV             hComEnv,    // COM port management construct pointer
        pSerialSharedData   pShd,       // packet management data
        BYTE                bMres,              // main response code
        BYTE                bSres               // subresponse code
        ) ;
```

Parameters

hComEnv

Specifies the data structure handle showing the SerialUnit communication environment.

pShd

Specifies the SerialUnit and ProtoDLL communication data structure instance pointer.

bMres

Error response main response code.

bSres

Error response subresponse code.

Return Value

Returns the number of data bytes. This is the same as the Fins_receiveData return value.

Remarks

sendFinsErrorResponse is a utility function that generates the FINS error response message from the FINS error response code sends it to the application.

This function executes makeErrFinsFrame internally, generates a FINS error response, and sends the response to the application.

See Also

makeErrFinsFrame

11.3.14 makeErrFinsFrame

makeErrFinsFrame generates a FINS error response.

```
BOOL WINAPI
ExeProcs_makeErrFinsFrame (
        HCOMENV             hComEnv,    // COM port management construct pointer
        pSerialSharedData   pShd,       // packet management data
        BYTE                bMres,              // main response code
        BYTE                bSres               // subresponse code
        ) ;
```

Parameters

hComEnv

Specifies the data structure handle showing the SerialUnit communication environment.

pShd

Specifies the SerialUnit and ProtoDLL communication data structure instance pointer.

bMres

Error response main response code.

bSres

Error response subresponse code.

Return Value

If a FINS error response is created, TRUE is reutrned. If the command was specified as response not needed, or broadcast, no response is created, and FALSE is returned.

Remarks

MakeErrFinsFrame uses the FINS error response code to make a FINS error response message. The response is generated as follows:

1. Creates FINSHEAD from pShd->finsHead.
2. Creates FINS error response message for pShd->FINSresponse.

This function operates on the premise that there is a FINSHEAD from an application stored in pShd->finsHead. Therefore, if the ProtoDLL is to modify pShd->finsHead directly, this function may not operate properly.

See Also

sendFinsErrorResponse

11.3.15 putFINSlogFromWin32

putFINSlogFromWin32 records the communication unit error log.

```
BOOL WINAPI    ExeProcs_putFINS logFromWin32 (
               HCOMENV              hComEnv,    // COM port management construct pointer
               pSerialSharedData    pShd,                   // packet management data
               DWORD                win32Code
               ) ;
```

Parameters

hComEnv

Specifies the data structure handle showing the SerialUnit communication environment.

pShd

Specifies the SerialUnit and ProtoDLL communication data structure instance pointer.

win32Code

Specifies an error cause or FINS log code.

Return Value

Returns TRUE when successful.

Remarks

putFINSlogFromWin32 records the communication unit error log. This log is the FINS communication unit log, and is not the event log or UDM. The data logged here is read by reading the FINS command log.

This log is normally called when there is a COM port communication error, or an invalid packet discard. These functions are implemented as follows:

- If win32Code is 0, not logged
- If win32Code is a Win32 value (CE_FRAME, CE_RXPARITY, CE_OVERRUN, CE_IOE), logged in the FINS log defined in FINSlog.h
- If win32Code is a value other than shown above, the win32Code value is judged as an error code, and an Invalid Packet Discard error is generated.

Notes

Performing FINS communication on top of serial communication is a good concept, but the actual design includes many challenges. We had to consider the many and varied protocols that are used for serial cables. We had to be able to handle all the existing protocols, and even include provisions for those that may yet be developed without the necessity of modifying the SerialUnit. Even though this receivees little attention, it was a very difficult challenge to overcome. Before settling on the present SerialUnit addition method, it went through a major specification rewrite. The previous addition method is not included in this SDK, but has been retained for compatibility purposes. This SDK manual still carries the remains of that previous method, such as member variables that are not used, and may be an obstacle to the understanding of the reader.

Implementation progressed, and the first time that FINS message communication was successfully completed with a PLC on SYSMAC WAY was a moment of joy for the designer. An application that had been operated on SYSMAC LINK, etc. could now be used without modification for communication with serial devices. The SerialUnit is for application developers, so we proceeded with this complex design hoping it would be to their benefit.

We are very grateful to Mr. K.H., who resolved many of our requirements. He implemented most of the SerialUnit and ProtoDLLs, and also demonstrated much understanding of the priority of technological value over deadlines, and workloads.

Miss K.Y., who made the temperature controller ProtoDLL, gave us the opportunity to communicate with devices other than PLCs, using a protocol other than FINS. The simple monitoring software that she and Mr. M.O. developed also gave us many hints and suggestions. We learned many areas of improvement for FinsGateway, which requires many, complicated settings to establish communication, so that the user-application does not need to consider the actual communication protocols. Mr. T.F., who mainly was in charge of the SerialSetup.dll, developed a simple user interface to allow the user to edit the extremely complicated registry. Mr. T.F. and Mr. K.H. also implemented the first Serial Communication Explorer. We are very grateful to them for agreeing so pleasantly to let us include this useful utility in FinsGateway. We have receive a lot of feedback directly from the sales support people. There was great value in the feedback received regarding the pre-release product.

Mr. Y.N., the author of the SDK, was in the position of approving the SerialUnit design and implementation. If the present SerialUnit is lacking in any way whatsoever, that is fully my responsibility.