SELEthernetController

IEC 61131 Library for ACSELERATOR RTAC® Projects

SEL Automation Controllers

Table of Contents

SELEthernetController

Introduction

The SELEthernetController library provides the ability to stream data through TCP and UDP ports. It provides client and server functionality for both TCP and UDP connections.

Various methods are used to initiate new read or write operations.

NOTE: See the ACSELERATOR RTAC Library Extensions Instruction Manual (LibraryExtensionsIM) for more information about the concepts used by the object-oriented extensions to the IEC 61131-3 standard.

Special Considerations

- Copying classes from this library causes unwanted behavior. This means the following:
  1. The assignment operator ":=" must not be used on any class from this library; consider assigning pointers to the objects instead.

```
// This is bad and in most cases will provide a compiler error
    such as:
// "C0328: Assignment not allowed for type class_SocketObject"
mySocketObject := otherSocketObject;
```

```
// This is fine
someVariable := mySocketObject.value;
// As is this
pt_mySocketObject := ADR(mySocketObject);
```

  2. Classes from this library must never be VAR_INPUT or VAR_OUTPUT members in function blocks, functions, or methods. Place them in the VAR_-IN_OUT section or use pointers instead.
- Classes in this library have memory allocated inside them. As such, they should only be created in environments of permanent scope (e.g., Programs, Global Variable Lists, or VAR_STAT sections).
- Opening and closing large numbers of Ethernet connections can take significant time. Take great care when using this library on a system with hard real-time requirements.

➤ This library does not have the ability to open ports in the RTAC firewall to allow inbound communication. This means that for the library to receive UDP packets or act as a TCP server, those ports must be opened as Ethernet Incoming Access Points in the AcRTAC project.

Supported Firmware Versions

You can use this library on any device configured using ACSELERATOR RTAC® SEL-5033 Software with firmware version R143 or higher.

Versions 3.5.0.6 and older can be used on RTAC firmware version R133 and higher.

Enumerations

Enumerations make code more readable by allowing a specific number to have a readable textual equivalent.

enum_SocketState

Enumeration	Description
LISTENING	The server socket is ready to receive communication requests.
OPEN	The socket is ready to send and receive UDP data.
CONNECTED	The socket has an active client server TCP session.
CLOSED	The socket will no longer allow communication.
ERROR	The socket has encountered an error and needs reconfiguration.

Aliases

This section lists aliases that this library defines.

SESSION_HANDLE

Alias	IEC 61131 Type
SESSION_HANDLE	POINTER TO BYTE

Functions

This library provides the following functions.

fun_StringToInaddr

Convert basic strings to the INADDR type this library uses. Strings provided to this function must be in the format *xxx.xxx.xxx.xxx*, where *xxx* is a number between 0 and 255.

Inputs

Name	IEC 61131 Type	Description
ipAddrString	STRING(15)	The string to convert to an INADDR.

Outputs

Name	IEC 61131 Type	Description
ipAddr	INADDR	The converted INADDR.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the string could be converted.

Processing

The `fun_StringToInaddr()` function does the following:

- ➤ Parses the provided string into four bytes.
- ➤ Outputs those bytes ordered in an INADDR.
- ➤ Returns false and outputs an IP address of 0.0.0.0 if the string could not be converted.

fun_InaddrToString

Converts an INADDR to a string in the format *xxx.xxx.xxx.xxx*, where *xxx* is a number between 0 and 255.

Inputs

Name	IEC 61131 Type	Description
ipAddr	INADDR	The INADDR to convert to a string.

Return Value

IEC 61131 Type	Description
STRING(15)	The resulting IP address as a string.

Processing

The `fun_InaddrToString()` function does the following:

- ➤ Parses the provided INADDR by its four bytes.
- ➤ Returns a string in the format *xxx.xxx.xxx.xxx*, where *xxx* is a number between 0 and 255.

Classes

Classes are a particular implementation of a Function Block(FB). They provide methods and properties, which a normal FB does not provide.

class_UdpSocket (Class)

This class provides a socket for sending and receiving using through use of the UDP protocol. Once enabled, the class creates, binds, sends, and receives to the configured ports.

Initialization Inputs

Name	IEC 61131 Type	Description
maxPacketSize	DINT	The maximum packet size, in bytes, to obtain through use of this socket. A value of zero or less results in the class imposing no limit on inbound packet size.

Properties

Name	IEC 61131 Type	Access	Description
State	enum_SocketState	R	The state of this socket.
LocalPort	UINT	R	The port number with which this socket interfaces locally.
LocalIPAddr	INADDR	R	The IP address with which this socket interfaces locally.
pt_Error	POINTER TO STRING	R	An error message describing any present error condition.

Properties are internal values made visible through Get and Set accessors. Access is defined as R (read), W (write), or R/W (read/write).

bootstrap_SetLocalIP (Method)

Perform a one-time setting of the local IP address and port this socket will use.

Inputs

Name	IEC 61131 Type	Description
localPort	UINT	The port number through which this socket receives and sends data through.
localIPAddr	INADDR	The IP address on the local box this socket uses. Use 0.0.0.0 to allow all local IP addresses.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the IP address and port number are set.

Processing

The `bootstrap_SetLocalIP()` method does the following:

- ➤ Immediately returns FALSE if the port and IP address are already set.
- ➤ Sets the IP address and port on the local machine for use by this socket.

Open (Method)

Configure the class_UdpSocket to allow communication.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the port is ready to send data.

Processing

The `Open()` method does the following:

- ➤ Opens the port for sending and receiving communication.
- ➤ Returns FALSE if the socket was unable to bind.

Close (Method)

Unconfigure the class_UdpSocket to disable communication.

Processing

The `Close()` method does the following:

- ➤ Discards any unread messages.
- ➤ Makes the socket unable to send or receive data.
- ➤ Forces reopening of the socket before processing additional communication.

SendData (Method)

Send a block of data to the socket.

Inputs

Name	IEC 61131 Type	Description
pt_data	POINTER TO BYTE	The data to send.
numBytes	DINT	The quantity of data in bytes.
destIPAddr	INADDR	The IP address to which data are sent.
destPort	UINT	The destination port to which data are sent.

Return Value

IEC 61131 Type	Description
DINT	The number of bytes sent.

Processing

The `SendData()` method does the following:

- ➤ Sends data if `Open()` has been successfully called.
- ➤ Validates access to the pointer provided.
- ➤ Limits *numBytes* to a positive number.
- ➤ Sends *numBytes* of data, starting at *pt_data*, to the socket.
- ➤ Returns the number of bytes successfully sent.

SendQueuedData(Method)

Send a block of data to the socket from the front of a queue.

Inputs

Name	IEC 61131 Type	Description
destIPAddr	INADDR	The IP address to which data are sent.
destPort	UINT	The destination port to which data are sent.

Inputs/Outputs

Name	IEC 61131 Type	Description
queue	class_ByteDeque	The queued data to send.

Return Value

IEC 61131 Type	Description
DINT	The number of bytes sent.

Processing

The `SendQueuedData()` method does the following:

- ➤ Sends data if `Open()` has been successfully called.
- ➤ Sends all data, starting at the front of *queue*, to the socket.
- ➤ Returns the number of bytes successfully sent.
- ➤ Removes bytes sent from the front of *queue*.

ReceiveFrom (Method)

Overwrites *data* with the first packet available to the socket.

Inputs/Outputs

Name	IEC 61131 Type	Description
data	class_ByteVector	The container to which the data are written.

Outputs

Name	IEC 61131 Type	Description
fromIpAddr	INADDR	The IP address from which the data came.
fromPort	UINT	The port from which the data came.

Return Value

IEC 61131 Type	Description
DINT	The number of bytes loaded.

Processing

The `ReceiveFrom()` method does the following:

- ➤ Does nothing if the socket is not open or a new packet is not available.
- ➤ Deletes any data found in *data*.
- ➤ Places the first packet—as many as *maxPacketSize* bytes—from this socket in *data*.
- ➤ Returns the number of bytes loaded into *data*.

ReceiveToQueueFrom (Method)

Overwrites *data* with the first packet available to the socket.

Inputs/Outputs

Name	IEC 61131 Type	Description
data	class_ByteDeque	The container to which the data are written.

Outputs

Name	IEC 61131 Type	Description
fromIpAddr	INADDR	The IP address from which the data came.
fromPort	UINT	The port from which the data came.

Return Value

IEC 61131 Type	Description
DINT	The number of bytes loaded.

Processing

The `ReceiveToQueueFrom()` method does the following:

➤ Does nothing if the socket is not open or a new packet is not available.

➤ Deletes any data found in *data*.

➤ Places the first packet—as many as *maxPacketSize* bytes—from this socket in *data*.

➤ Returns the number of bytes loaded into *data*.

class_TcpClient (Class)

This class provides a client socket for the TCP protocol. Once enabled, the class creates, binds, and sends to the configured port. Because TCP is session-based communication, the client should close the session upon completion of the communication to conserve server resources.

Properties

Name	IEC 61131 Type	Access	Description
State	enum_SocketState	R	The state of this socket.
LocalPort	UINT	R	The port number provided by the user for use locally. A value of zero means an OS-chosen port value will be used.
LocalIPAddr	INADDR	R	The IP address with which this socket interfaces locally.

Properties

Name	IEC 61131 Type	Access	Description
DestIPAddr	INADDR	R	The IP address to which any triggered message is sent.
DestPort	UINT	R	The port number to which any triggered message is sent.
pt_Error	POINTER TO STRING	R	An error message describing any present error condition.

Properties are internal values made visible through Get and Set accessors. Access is defined as R (read), W (write), or R/W (read/write).

bootstrap_SetLocalIP (Method)

Perform a one-time setting of the local IP address and port this socket will use.

Inputs

Name	IEC 61131 Type	Description
localPort	UINT	The port number through which this socket receives and sends data. If this value is zero then the OS will choose a random outgoing port.
localIPAddr	INADDR	The IP address on the local box this socket uses. Use 0.0.0.0 to allow all local IP addresses.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the IP address and port number are set.

Processing

The `bootstrap_SetLocalIP()` method does the following:

- ➤ Immediately returns FALSE if the port and IP address are already set.
- ➤ Sets the IP address and port on the local machine for use by this socket.

SetIP (Method)

Set the IP address and port to be used on the next `Open()` request.

Inputs

Name	IEC 61131 Type	Description
destIPAddr	INADDR	The IP address to which data is sent.
destPort	UINT	The destination port to which data is sent.

Processing

The `SetIP()` method does the following:

- ➤ Sets the IP address and port that will become the new destination the next time `Open()` is called.

Open (Method)

Configure the class_TcpClient to allow communication.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the port is ready to send data.

Processing

The `Open()` method does the following:

- ➤ Opens the port for sending and receiving communication, if `SetIP()` has been successfully run.
- ➤ Returns false if the socket was unable to connect.

Close (Method)

Unconfigure the class_TcpClient, to disable communication.

Inputs

Name	IEC 61131 Type	Description
forceClose	BOOL	Close the session without waiting for confirmation.

Processing

The `Close()` method does the following:

- ➤ Discards any unread data.
- ➤ Allows server to retain session until all data are read, if *forceClose* is FALSE.
- ➤ Makes the socket unable to send data or receive replies.
- ➤ Forces reopening of the socket before processing additional communication.

SendData (Method)

Send a block of data to the socket.

Inputs

Name	IEC 61131 Type	Description
pt_data	POINTER TO BYTE	The data to send.
numBytes	DINT	The quantity of data in bytes.

Return Value

IEC 61131 Type	Description
DINT	The number of bytes sent.

Processing

The `SendData()` method does the following:

- Does nothing if the socket is not open.
- Validates access to the pointer provided.
- Limits *numBytes* to a positive number.
- Sends all provided data to the socket.
- Returns the number of bytes successfully sent.

SendQueuedData (Method)

Send a block of data to the socket from the front of a queue.

Inputs/Outputs

Name	IEC 61131 Type	Description
queue	class_ByteDeque	The data to send.

Return Value

IEC 61131 Type	Description
DINT	The number of bytes sent.

Processing

The `SendQueuedData()` method does the following:

- Does nothing if the socket is not open.
- Sends all provided data, starting at the front of *queue*, to the socket.

- ➤ Returns the number of bytes successfully sent.
- ➤ Removes bytes sent from the front of *queue*.

ReceiveData (Method)

Appends a block of data from the socket to *data*.

Inputs

Name	IEC 61131 Type	Description
numBytes	DINT	The number of bytes requested.

Inputs/Outputs

Name	IEC 61131 Type	Description
data	class_ByteVector	The container to which the data are written.

Return Value

IEC 61131 Type	Description
DINT	The number of bytes loaded.

Processing

The `ReceiveData()` method does the following:

- ➤ Does nothing if the socket is not open.
- ➤ Requests data from the socket until there are either no more or it reaches *numBytes*, whichever happens first.
- ➤ Appends all data retrieved to *data*.
- ➤ Returns the number of bytes appended.

ReceiveToQueue (Method)

Pushes a block of data from the socket to the back of *queue*.

Inputs

Name	IEC 61131 Type	Description
numBytes	DINT	The number of bytes requested.

Inputs/Outputs

Name	IEC 61131 Type	Description
queue	class_ByteDeque	The container to which the data are written.

Return Value

IEC 61131 Type	Description
DINT	The number of bytes loaded.

Processing

The `ReceiveToQueue()` method does the following:

- ➤ Does nothing if the socket is not open.
- ➤ Requests data from the socket until there are either no more or it reaches *numBytes*, whichever happens first.
- ➤ Pushes all data retrieved to the back of *queue*.
- ➤ Returns the number of bytes pushed.

class_TcpServer (Class)

This class provides a listening socket for the TCP protocol. Once enabled the class creates, binds, and receives on the configured port.

Initialization Inputs

Name	IEC 61131 Type	Description
maxSessions	USINT	The maximum number of concurrent sessions to allow on this socket. This will be forced to at least one.

Properties

Name	IEC 61131 Type	Access	Description
State	enum_SocketState	R	The state of this socket.
LocalPort	UINT	R	The port number with which this socket interfaces locally.
LocalIPAddr	INADDR	R	The IP address with which this socket interfaces locally.
NumSessions	USINT	R	The number of client sessions connected to this server block.
SessionState	enum_SocketState	R	The state of the selected session. CLOSED if nothing is connected or CONNECTED if data can still be read from the socket.
DestIPAddr	INADDR	R	The IP address to which any triggered message is sent.

Properties

Name	IEC 61131 Type	Access	Description
DestPort	UINT	R	The port to which any triggered message is sent.
pt_Error	POINTER TO STRING	R	An error message describing any present error condition.

Properties are internal values made visible through Get and Set accessors. Access is defined as R (read), W (write), or R/W (read/write).

bootstrap_SetLocalIP (Method)

Perform a one-time setting of the local IP address and port this socket will use.

Inputs

Name	IEC 61131 Type	Description
localPort	UINT	The port number through which this socket receives and sends data.
localIPAddr	INADDR	The IP address on the local box this socket uses. Use 0.0.0.0 to allow all local IP addresses.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the IP address and port number are set.

Processing

The `bootstrap_SetLocalIP()` method does the following:

- ➤ Immediately returns FALSE if the port and IP address are already set.
- ➤ Sets the IP address and port on the local machine used by this socket.

Open (Method)

Configure the class_TcpServer to allow for communication.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the port is ready to receive connections.

Processing

The `Open()` method does the following:

- Configures the server to receive client connections.
- Returns FALSE if the socket was unable to bind.

Close (Method)

Unconfigure the class_TcpServer to disable communication.

Inputs

Name	IEC 61131 Type	Description
forceClose	BOOL	Close the session without waiting for confirmation.

Processing

The `Close()` method does the following:

- Discards any unread data.
- Closes all open client sessions: gracefully if forceClose is FALSE.
- Makes the socket unable to send data or receive replies.
- Forces reconfiguration of the socket before attempting additional communication.

AcceptNextSession (Method)

Accept the next new inbound data session. This does not change the active session.

Return Value

IEC 61131 Type	Description
SESSION_HANDLE	The descriptor to allow further communication through this session.

Processing

The `AcceptNextSession()` method does the following:

- Does nothing if the socket is not open for listening.
- Accepts the next outstanding client session to as many as *maxSessions*.
- Updates `NumSessions` based on the sessions accepted.
- Returns the value SysSocket.RTS_INVALID_HANDLE if there are no outstanding sessions.
- Returns the handle to the accepted session.

SetSession (Method)

Select the session from which the class reads data from and replies to based on a previously received identifier.

Inputs

Name	IEC 61131 Type	Description
sessionID	SESSION_HANDLE	The identifier of the desired session as returned by `AcceptNextSession()`.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the session exists.

Processing

The `SetSession()` method does the following:

- ➤ Selects the read and write session tied to *sessionID*.
- ➤ Sets `DestIPAddr` and `DestPort` to the values for the selected session.
- ➤ Returns false and sets `DestIPAddr` to 0.0.0.0, `DestPort` to 0, and `SessionState` to closed if *sessionID* does not represent an active session.

GetSessionInfo (Method)

Get the IP address and port number related to a session handle.

Inputs

Name	IEC 61131 Type	Description
sessionID	SESSION_HANDLE	The identifier of a session as returned by `AcceptNextSession()`.

Outputs

Name	IEC 61131 Type	Description
sessionIPAddr	INADDR	The IP address attached to *sessionID*.
sessionPort	UINT	The port number attached to *sessionID*.

Processing

The `GetSessionInfo()` method does the following:

- ➤ Outputs the IP address and port number tied to *sessionID*.

- ➤ Sets *sessionIPAddr* to 0.0.0.0 and *sessionPort* to 0, if *sessionID* does not represent an active session.

CloseSession (Method)

Close the selected session and clear any pending data.

Inputs

Name	IEC 61131 Type	Description
forceClose	BOOL	Close the session without waiting for confirmation.

Processing

The `CloseSession()` method does the following:

- ➤ Closes the active session.
- ➤ Sets `DestIPAddr` and `DestPort` to zero.
- ➤ Discards any unread data.
- ➤ Allows client to retain session until all data are read, if *forceClose* is FALSE.
- ➤ Decrements `NumSessions`.

SendData (Method)

Send a block of data to the socket.

Inputs

Name	IEC 61131 Type	Description
pt_data	POINTER TO BYTE	The data to send.
numBytes	DINT	The quantity of data in bytes.

Return Value

IEC 61131 Type	Description
DINT	The number of bytes sent.

Processing

The `SendData()` method does the following:

- ➤ Does nothing if no valid session has been selected.
- ➤ Validates access to the pointer provided.
- ➤ Limits *numBytes* to a positive number.
- ➤ Sends all provided data to the socket.

- Returns the number of bytes successfully sent.

SendQueuedData (Method)

Send a block of data to the socket from the front of a queue.

Inputs/Outputs

Name	IEC 61131 Type	Description
queue	class_ByteDeque	The data to send.

Return Value

IEC 61131 Type	Description
DINT	The number of bytes sent.

Processing

The `SendQueuedData()` method does the following:

- Does nothing if the socket is not open.
- Sends all provided data, starting at the front of the queue, to the socket.
- Returns the number of bytes successfully sent.
- Removes bytes sent from the front of *queue*.

ReceiveData (Method)

Receive a block of data from the active session.

Inputs

Name	IEC 61131 Type	Description
numBytes	DINT	The number of bytes requested.

Inputs/Outputs

Name	IEC 61131 Type	Description
data	class_ByteVector	The container to which the data are written.

Return Value

IEC 61131 Type	Description
DINT	The number of bytes loaded into *data*.

Processing

The `ReceiveData()` method does the following:

- ➤ Does nothing if no valid session has been selected.
- ➤ Checks for available data in the selected session.
- ➤ Requests data from the socket until there are no more or it reaches *numBytes*, whichever happens first.
- ➤ Appends all data retrieved to *data*.
- ➤ Returns the number of bytes appended.

ReceiveToQueue (Method)

Pushes a block of data from the socket to the back of *queue*.

Inputs

Name	IEC 61131 Type	Description
numBytes	DINT	The number of bytes requested.

Inputs/Outputs

Name	IEC 61131 Type	Description
queue	class_ByteDeque	The container to which the data are written.

Return Value

IEC 61131 Type	Description
DINT	The number of bytes loaded.

Processing

The `ReceiveToQueue()` method does the following:

- ➤ Does nothing if the socket is not open.
- ➤ Requests data from the socket until there are no more or it reaches *numBytes*, whichever happens first.
- ➤ Pushes all data retrieved to the back of *queue*.
- ➤ Returns the number of bytes pushed.

Benchmarks

Benchmark Platforms

The benchmarking tests recorded for this library are performed on the following platforms.

- SEL-3555
  - Dual-core Intel i7-3555LE processor
  - 4 GB ECC RAM
  - R134-V0 firmware
- SEL-3530
  - R134-V0 firmware
- SEL-3505
  - R134-V0 firmware

Benchmark Test Descriptions

fun_StringToInaddr

The posted time is the average execution time of 100 consecutive calls for the string "192.168.100.100".

fun_InaddrToString

The posted time is the average execution time of 100 consecutive calls for the IP address 192.168.100.100.

class_UdpSocket.Open

The posted time is the average execution time of 100 successful method calls to open a socket.

class_UdpSocket.Close

The posted time is the average execution time of 100 successful method calls to close a socket.

class_UdpSocket.SendData

The posted time is the average execution time of 100 consecutive calls when sending 504 bytes of data, resulting in a 512-byte total packet size.

class_UdpSocket.SendQueuedData

The posted time is the average execution time of 100 consecutive calls when sending 504 bytes of data, resulting in a 512-byte total packet size.

class_UdpSocket.ReceiveFrom

The posted time is the average execution time of 100 consecutive calls when receiving 504 bytes of data, resulting in a 512-byte total packet size.

class_UdpSocket.ReceiveToQueueFrom

The posted time is the average execution time of 100 consecutive calls when receiving 504 bytes of data, resulting in a 512-byte total packet size.

class_TcpClient.SetIP

The posted time is the average execution time of 100 consecutive calls.

class_TcpClient.Open

The posted time is the average execution time of 100 successful method calls to open a socket.

class_TcpClient.Close

The posted time is the average execution time of 100 successful method calls to close a socket.

class_TcpClient.SendData

The posted time is the average execution time of 100 consecutive calls when sending 1400 bytes of data.

class_TcpClient.SendQueuedData

The posted time is the average execution time of 100 consecutive calls when sending 1400 bytes of data.

class_TcpClient.ReceiveData

The posted time is the average execution time of 100 consecutive calls when receiving 1400 bytes of data.

class_TcpClient.ReceiveToQueue

The posted time is the average execution time of 100 consecutive calls when receiving 1400 bytes of data.

class_TcpServer.Open

The posted time is the average execution time of 100 successful method calls to open a socket.

class_TcpServer.Close

The posted time is the average execution time of 100 successful method calls to close a socket.

class_TcpServer.AcceptNextSession

The posted time is the average execution time of 100 successful method calls when there is another session to accept.

class_TcpServer.SetSession

The posted time is the average execution time of 100 consecutive calls.

class_TcpServer.GetSessionInfo

The posted time is the average execution time of 100 consecutive calls.

class_TcpServer.CloseSession

The posted time is the average execution time of 100 successful method calls to close a session.

class_TcpServer.SendData

The posted time is the average execution time of 100 consecutive calls when sending 1400 bytes of data.

class_TcpServer.SendQueuedData

The posted time is the average execution time of 100 consecutive calls when sending 1400 bytes of data.

class_TcpServer.ReceiveData

The posted time is the average execution time of 100 consecutive calls when receiving 1400 bytes of data.

class_TcpServer.ReceiveToQueue

The posted time is the average execution time of 100 consecutive calls when receiving 1400 bytes of data.

Benchmark Results

Operation Tested	Platform (time in μs)		
	SEL-3555	SEL-3530	SEL-3505
fun_StringToInaddr	2	12	18
fun_InaddrToString	6	40	63
class_UdpSocket.Open	25	150	330
class_UdpSocket.Close	13	65	117
class_UdpSocket.SendData	26	150	380
class_UdpSocket.SendQueuedData	25	160	380
class_UdpSocket.ReceiveFrom	12	90	200
class_UdpSocket.ReceiveToQueueFrom	12	100	220
class_TcpClient.SetIP	1	1	1
class_TcpClient.Open	67	740	1200
class_TcpClient.Close	41	310	660
class_TcpClient.SendData	1	3	6
class_TcpClient.SendQueuedData	1	3	5
class_TcpClient.ReceiveData	8	80	110
class_TcpClient.ReceiveToQueue	7	80	110
class_TcpServer.Open	36	200	400
class_TcpServer.Close	20	100	140
class_TcpServer.AcceptNextSession	20	97	130
class_TcpServer.SetSession	2	4	6
class_TcpServer.GetSessionInfo	2	5	14
class_TcpServer.CloseSession	33	320	690
class_TcpServer.SendData	1	3	6
class_TcpServer.SendQueuedData	1	3	5
class_TcpServer.ReceiveData	9	80	110
class_TcpServer.ReceiveToQueue	8	75	110

Examples

These examples demonstrate the capabilities of this library. Do not mistake them as suggestions or recommendations from SEL.

Implement the best practices of your organization when using these libraries. As the user of this library, you are responsible for ensuring correct implementation and verifying that the project using these libraries performs as expected.

Sending Data Out a UDP Socket

Objective

A user has an array of bytes formatted to place on the network and needs to send it to a specific IP address and port via UDP.

Solution

The user can create the program shown in *Code Snippet 1* to send the byte array out each task cycle.

Code Snippet 1 prg_UdpOut

```
PROGRAM prg_UdpOut
VAR
    // Configuration Information - Uses any available interface.
    LocalIPAddress : STRING(15) := '0.0.0.0';
    LocalPortNumber : UINT := 5000;
    DestinationIPAddress : STRING(15) := '10.10.10.10';
    DestinationPortNumber : UINT := 5000;

    // Data to send each task cycle.
    Data : ARRAY [1 .. 1000] OF BYTE;

    // Socket to send data on.
    UdpSocket : class_UdpSocket(maxPacketSize := 1024);

    // Initialization variables.
    SocketInitialized : BOOL := FALSE;
    LocalIP : SELEthernetController.INADDR;
    DestIP : SELEthernetController.INADDR;
END_VAR
```

```
IF NOT SocketInitialized THEN
    fun_StringToInaddr(LocalIPAddress, ipAddr => LocalIP);
    fun_StringToInaddr(DestinationIPAddress, ipAddr => DestIP);
    UdpSocket.bootstrap_SetLocalIP(LocalPortNumber, LocalIP);
    UdpSocket.Open();
    SocketInitialized := TRUE;
ELSE
    UdpSocket.SendData(ADR(Data[1]), SIZEOF(Data),
                     DestIP, DestinationPortNumber);
END_IF
```

Creating a UDP Server

Objective

A user would like the RTAC to be able to receive data from multiple clients over the UDP protocol.

After some internal validation of a received packet, the server will reply with `OK` in ASCII if the packet was correctly formatted.

Assumptions

This solution assumes that the AcRTAC project containing the provided code includes an Access Point opening at the desired port and that the inbound packets are not larger than 1024 bytes.

Solution

The user can create the program shown in *Code Snippet 2* to receive one inbound data packet each task cycle.

Code Snippet 2 prg_UdpServer

```
PROGRAM prg_UdpServer
VAR
    // Configuration Information - Uses any available interface.
    LocalIPAddress : STRING(15) := '0.0.0.0';
    LocalPortNumber : UINT := 1515;

    // Storage for inbound and outbound messages.
    DataIn : SELEthernetController.class_ByteVector;
    DataOut : STRING(2) := 'OK';

    // The socket and its initialization data.
    UdpSocket : class_UdpSocket(maxPacketSize := 1024);
    SocketInitialized : BOOL := FALSE;
    LocalIP : SELEthernetController.INADDR;

    // Workbench variables for storing current client information.
    DestIP : SELEthernetController.INADDR;
    DestPort : UINT;
    PacketValid : BOOL;
END_VAR
```

```
IF NOT SocketInitialized THEN
    fun_StringToInaddr(LocalIPAddress, ipAddr => LocalIP);
    UdpSocket.bootstrap_SetLocalIP(LocalPortNumber, LocalIP);
    UdpSocket.Open();
    SocketInitialized := TRUE;
ELSE
    IF 0 <> UdpSocket.ReceiveFrom(DataIn, fromIpAddr => DestIP, fromPort =>
        DestPort) THEN
        ; // Set PacketValid based on the contents of DataIn.
        IF PacketValid THEN
            UdpSocket.SendData(ADR(DataOut), 2, DestIP, DestPort);
        END_IF
    END_IF
END_IF
```

Creating a TCP Client

Objective

A user would like the RTAC to be able to send data to a TCP server for modification.

Assumptions

For this use case, we assume that the server doubles any data that are sent to it.

Solution

The user can create the program defined in *Code Snippet 3* to send packets to the remote server and receive data in reply.

Code Snippet 3 prg_TcpClient

```
PROGRAM prg_TcpClient
VAR
    // Configuration Information - Uses any available interface.
    LocalIPAddress : STRING(15) := '0.0.0.0';
    LocalPortNumber : UINT := 2442;
    DestinationIPAddress : STRING(15) := '10.10.10.10';
    DestinationPortNumber : UINT := 2442;

    // Storage for data being sent and received.
    DataOut : ARRAY [1 .. 200] OF BYTE;
    DataIn : SELEthernetController.class_ByteVector;

    // The socket and its initialization state.
    TcpClient : class_TcpClient;
    SocketInitialized : BOOL := FALSE;

    // IP address variables.
    LocalIP : SELEthernetController.INADDR;
    DestIP : SELEthernetController.INADDR;

    // Flags used in manipulating the state of the function.
    IsSending : BOOL;
    DataSent : DINT;
    DataReceived : DINT;
END_VAR
```

```
IF NOT SocketInitialized THEN
    fun_StringToInaddr(LocalIPAddress, ipAddr => LocalIP);
    fun_StringToInaddr(DestinationIPAddress, ipAddr => DestIP);
    TcpClient.bootstrap_SetLocalIP(LocalPortNumber, LocalIP);
    TcpClient.SetIP(DestIP, DestinationPortNumber);
    TcpClient.Open();
    SocketInitialized := TRUE;
    IsSending := TRUE;
ELSE
    IF IsSending THEN
        DataSent := TcpClient.SendData(ADR(DataOut[1]), 200);
        DataIn.Recycle();
        IsSending := FALSE;
        DataReceived := 0;
    ELSE
        // Here we request the total data expected minus anything
        // already received and add it to anything already received.
        // This allows the reception of data to cross multiple cycles.
        DataReceived := DataReceived +
                    TcpClient.ReceiveData((DataSent * 2)
                                 - DataReceived, DataIn);
        IF DataReceived = (2 * DataSent) THEN
            ; // Do some work based the server's reply.
            IsSending := TRUE;
        END_IF
    END_IF
END_IF
```

Parsing Network Traffic With a Deque

This example demonstrates using a deque for network communication.

Objective

Parse the data stream of information sent from a TCP server to the TCP client on the RTAC. Increment a counter every time the characters "SEL" are seen in the stream.

Assumptions

This example assumes that there is a server to connect to and streams data through the connection once the connection is established. It also assumes that the library Queue has been inserted in the project.

Solution

The deque is used as storage for information received from a TCP socket. The received data are then searched for the string 'SEL' and a counter is incremented every time the string is found. As it searches for the string, data are discarded from the front of the deque. This implementation is shown in *Code Snippet 4*

Code Snippet 4 prg_SELSearch

```
PROGRAM prg_SELSearch
VAR
    // Configuration Information - Uses any available interface.
    LocalIPAddress : STRING(15) := '0.0.0.0';
    LocalPortNumber : UINT := 2442;
    DestinationIPAddress : STRING(15) := '10.10.10.10';
    DestinationPortNumber : UINT := 2442;

    // The socket and its initialization state.
    TcpClient : class_TcpClient;
    SocketInitialized : BOOL := FALSE;
    // IP address variables.
    LocalIP : SELEthernetController.INADDR;
    DestIP : SELEthernetController.INADDR;
    // Deque to hold received data.
    DataIn : Queue.class_ByteDeque(0);
    // The string to search for in the incoming data.
    TargetString : STRING := 'SEL';
    // A temporary string used to find the targetString.
    Peek : STRING(3);
    // A flag for breaking the search loop.
    Searching : BOOL;
    // Counter for the number of times the targetString is found.
    Counter : UDINT := 0;
END_VAR
```

```
IF NOT SocketInitialized THEN
    fun_StringToInaddr(LocalIPAddress, ipAddr => LocalIP);
    fun_StringToInaddr(DestinationIPAddress, ipAddr => DestIP);
    TcpClient.bootstrap_SetLocalIP(LocalPortNumber, LocalIP);
    TcpClient.SetIP(DestIP, DestinationPortNumber);
    TcpClient.Open();
    SocketInitialized := TRUE;
ELSE
    // Read new data from the socket.
    IF 0 <> TcpClient.ReceiveToQueue(10_000, DataIn) THEN
    searching := TRUE;
        WHILE searching DO
            // See if the string 'SEL' appears in the front of the deque.
            IF 3 = DataIn.Front(ADR(peek), 3) THEN
                IF peek = TargetString THEN
                    (* The string 'SEL' was found, increment the counter
                       and erase the string from the deque. *)
                    Counter := Counter + 1;
                    DataIn.EraseFront(3);
                ELSE
                    (* The string wasn't found, erase the first character
                       and continue looking. *)
                    DataIn.EraseFront(1);
                END_IF
            ELSE
                Searching := FALSE;
            END_IF
        END_WHILE
    END_IF
END_IF
```

Configuring a Simple TCP Server With Sessions

Objective

Parse all input data from several clients looking for the string “Hello.” When the string is found, that client receives a reply “World$N$R”.

The server handles accepting and tracking all sessions and iterates across them to allow each time to process.

Assumptions

This example assumes that Port 10024 is open through use of an Access Point configured as Ethernet Incoming and set to receive Raw TCP. It also assumes that the client can access that port through any intermediary firewalls. It also assumes that the library Queue has been inserted in the project.

All error checking has been omitted to facilitate the brevity of this implementation.

There must be a structure defined to hold session related data. *Code Snippet 5* shows an example for this implementation.

Code Snippet 5 Session Structure

```
TYPE struct_SessionInfo :
STRUCT
    // There is a connected session stored here.
    Active : BOOL := FALSE;
    // The Handle for this session.
    Handle : SESSION_HANDLE := SysSocket.RTS_INVALID_HANDLE;
    // Deque containing incoming data ready for consumption.
    ReceiveDeque : class_ByteDeque(0);
    // Deque containing data ready to be sent to the telnet client.
    SendDeque : class_ByteDeque(0);
END_STRUCT
END_TYPE
```

Solution

Instantiate a class_TcpServer and associated data control objects as seen in *Code Snippet 6*. Allow this program to run every task scan.

Verification

To verify that this solution is functional, start running the server on the RTAC and then attach with a raw TCP connection on port 10024. Any instance of the word “Hello” should be followed immediately by the echo “World.”

Characters other than new-lines should eventually echo “Usage : Hello.”

The code as written should accept as many as five concurrent sessions with more allowed after a given session disconnects.

Code Snippet 6 prg_SimpleServer

```
PROGRAM prg_SimpleServer
VAR CONSTANT
    // The maximum number of sessions allowed for this server.
    _c_NumSessions : USINT := 5;
    // The maximum bytes to read per scan.
    _c_NumBytes : DINT := 1024;
END_VAR
VAR
    // Configuration Information
    // Uses any available interface.
    LocalIPAddress : STRING(15) := '0.0.0.0';
    LocalPortNumber : UINT := 10024;
    ListenIPAddr : INADDR;

    SocketInitialized : BOOL := FALSE;
    TcpServer : class_TcpServer(_c_NumSessions);

    Sessions : ARRAY [1 .. _c_NumSessions] of struct_SessionInfo;

    Peek : STRING(5);
    Reply : STRING(7) := 'World$R$N';
    Reply2 : STRING(15) := 'Usage : Hello$R$N';

    tempSession : SESSION_HANDLE;
    sendByteCount : DINT;
    numSend : DINT;
    i : UDINT;
END_VAR
```

```
IF NOT SocketInitialized THEN
    IF SELEthernetController.fun_StringToInaddr(LocalIPAddress, ipAddr =>
         ListenIPAddr) THEN
        TcpServer.bootstrap_SetLocalIP(LocalPortNumber, ListenIPAddr);
        IF TcpServer.Open() THEN
            SocketInitialized := TRUE;
        END_IF
    END_IF
ELSE
    // Clean up any session that is in error.
    IF TcpServer.SessionState = ERROR THEN
        TcpServer.CloseSession(TRUE);
    END_IF

    // Accept any new sessions if possible.
    FOR i := 1 TO _c_NumSessions DO
        IF NOT Sessions[i].Active THEN
            // There is a session object available,
            // see if there is a new session pending.
            tempSession := TcpServer.AcceptNextSession();
            IF tempSession <> SysSocket.RTS_INVALID_HANDLE THEN
                Sessions[i].Handle := tempSession;
                Sessions[i].Active := TRUE;
                Sessions[i].SendDeque.Recycle();
                Sessions[i].ReceiveDeque.Recycle();
            ELSE
                //There are no outstanding sessions. Stop asking.
                EXIT;
            END_IF
        END_IF
    END_FOR

    // Cycle through all current sessions and process their data.
    FOR i := 1 TO _c_NumSessions DO
        IF Sessions[i].Active THEN
            IF TcpServer.SetSession(Sessions[i].Handle) THEN
                // Read any data from the socket.
                TcpServer.ReceiveToQueue(_c_NumBytes,
                    Sessions[i].ReceiveDeque);

                // Here is where meaningful work would be added.
                WHILE Sessions[i].ReceiveDeque.Size >= 5 DO
                    // See if the string 'SEL' appears in the front of the
                        deque.
                    IF 5 = Sessions[i].ReceiveDeque.Front(ADR(Peek), 5) THEN
                        IF peek = 'Hello' THEN
                            Sessions[i].SendDeque.PushBack(ADR(Reply), 7);
                            Sessions[i].ReceiveDeque.EraseFront(5);
                        ELSIF peek[0] <> 10 and peek[0] <> 13 THEN
                            // The string wasn't found, erase the first
                            // character and continue looking.
                            Sessions[i].SendDeque.PushBack(ADR(Reply2), 15);
                            Sessions[i].ReceiveDeque.EraseFront(1);
                        ELSE
                            Sessions[i].ReceiveDeque.EraseFront(1);
                        END_IF
                    ELSE
                        EXIT;
                    END_IF
                END_WHILE

                TcpServer.SendQueuedData(Sessions[i].SendDeque);

                // If the session closed while working with it, clean up.
                IF TcpServer.SessionState = CLOSED THEN
                    Sessions[i].Handle := RTS_INVALID_HANDLE;
                    Sessions[i].Active := FALSE;
                END_IF
            ELSE
                // If the session closed while doing other work, clean up.
                Sessions[i].Handle := RTS_INVALID_HANDLE;
                Sessions[i].Active := FALSE;
            END_IF
        END_IF
    END_FOR
END_IF
```

Release Notes

Version	Summary of Revisions	Date Code
3.5.1.0	➤ Allows new versions of ACSELERATOR RTAC to compile projects for previous firmware versions without SEL IEC types "Cannot convert" messages. ➤ Must be used with R143 firmware or later.	20180921
3.5.0.6	➤ Allow the class to recover instead of closing the socket when attempting to send large amounts of data all at once.	20160501
3.5.0.5	➤ Added queues as input and output mechanisms for socket data. ➤ Allow TCP client ports to connect to a server without binding to a specific local port.	20150511
3.5.0.3	➤ Made TCP sockets not throw error when outgoing data buffer is full.	20141107
3.5.0.2	➤ Initial release.	20141010