Integrated Spectrum Management System

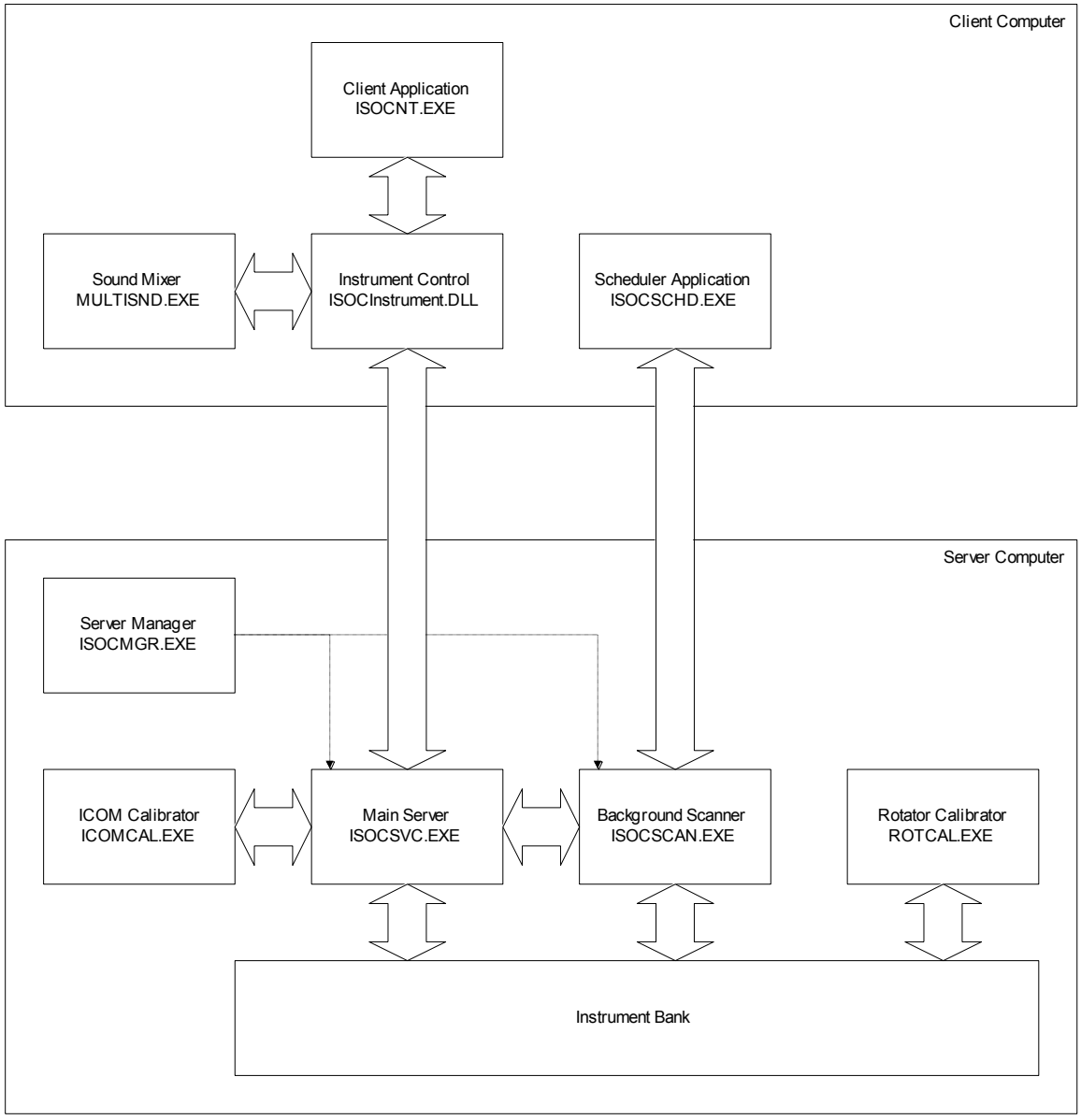
ISOC for Windows

Application Programming Interfaces

Last Updated: 2/19/2013 9:40:00 PM

The Integrated Spectrum Observation Center is a suite of applications providing flexible remote access to instrument suites. Several application components are controlled via interfaces that are sufficiently generic to be used from external (user-developed) applications.

The following diagram demonstrates the components of the ISOC for Windows system:



For a more detailed description of the role and operation of these components, please refer to the ISOC for Windows Internal Software Architecture manual.

Three of the components seen in this diagram offer an API that can be utilized by third-party applications.

The ISOCSVC.EXE server application represents the core of the ISOC system. It provides remote access to all ISOC instruments, and provides additional functionality as well, including support for the switch matrix, remote control power bar, background command execution, processing of graphical traces, digital audio, and more.

The ISOCInstrument.DLL is an ActiveX control for use within Windows applications and Web scripts. It provides a simple graphical interface to the ISOCSVC.EXE server, including the ability to display instrument traces, process background command results, and play back digital audio. An ISOC client application would normally use the ISOCInstrument.DLL instead of accessing the ISOCSVC.EXE server directly.

The ISOCSCAN.EXE is another server application that performs scheduled background scans. Presently it can perform background level scans utilizing the ESN or ICOM radio receivers, and background audio recording.

This document provides a (necessarily incomplete, since the ISOC suite is still a work-in-progress) quick reference for interfacing with these key ISOC components.

> ***Preliminary:*** All information contained within this document is preliminary. This document is released to provide advance information to those who wish to consider utilizing ISOC components within their applications. However, the programs and components described in this document are under development themselves, and thus they are subject to change.

The main ISOC server, ISOCSVC.EXE, offers a control interface via a TCP socket; normally, on port 25449. Applications can connect to this socket, perform simple authentication, and execute any of the control functions listed in this section, or perform interactive communication with instruments that the ISOC server can control.

> ***Note:*** Accessing the ISOCSVC service directly from your code is not recommended. When possible, use the ISOCInstrument ActiveX control (see section 3) for the purpose of communicating with the service.

The purpose of the ISOCSVC service is to provide a reliable means to access remote instrumentation, perform one-off and repetitive commands, and obtain results at high efficiency. Commanding the ISOCSVC service is performed through a TCP socket, using human-readable command strings. Depending on the nature of the command, responses may be either human-readable or binary.

The ISOCSVC service is protected against unauthorized access using a simple authentication scheme. The authentication algorithm is encoded in the `ISOCDev.dll` library (see Appendix A.)

After authentication bytes are successfully exchanged, the ISOCSVC service is ready to receive commands. These commands are used to query the service's status, obtain control of an instrument, set up secondary channels of communication, send data to, or receive data from the instrument, and terminate the connection.

1.1. Operations Overview

Typically, an application utilizes the ISOCSVC service as follows:

1. Connect to the ISOCSVC service

   The application creates a TCP socket and connects to the ISOC server on the designated port number (port 25449). If the connection is successful, it performs authentication.

2. Obtain a list of instruments

   The application issues the appropriate command and reads a list of instruments present on the server. (This is used, for instance, to present a list of selections to the user.)

3. Obtain control of an instrument

   The application requests control of the desired instrument. If the request is completed successfully, a two-way data path is established between the application and the requested instrument. It is also important to maintain activity on the communication channel by issuing special "keep-alive" commands; otherwise, the server will automatically terminate inactive connections.

4. Set up background channels of communication

   This step is optional. The client application may instruct the server to perform periodic commands and send the results to the designated UDP port on the client. A similar mechanism is used for obtaining traces that represent the content of the graphical display on certain instruments, and for obtaining digital audio.

5. Terminate the connection

   The application asks the server to terminate the connection or it simply closes the socket, causing the server to treat it as a "hang up" event.

There can be several TCP sockets open between a client and a server. One TCP socket can be used to control at most one instrument at a time. Thus, controlling multiple instruments requires the simultaneous use of multiple TCP sockets, which is best implemented using a multi-threaded approach.

1.2. Instrument Commanding

The ISOC server contains no instrument-specific functionality. (Exceptions include ICOM calibration information, antenna rotator calibration information, and support for the remote control power bar and switch matrix.) Instruments are commanded using instrument-specific commands that are described in each instrument's programming manual.

The ISOC server provides two mechanisms for communicating with an instrument: commands and queries. A convention borrowed from HP-IB is used to distinguish between the two. A string from the client is interpreted as a command if it terminates with a semicolon, and a query if it terminates with a question mark. In case of a query, the server attempts to read a response from the instrument.

This convention works well with instruments on the HP-IB bus. However, if fails in case of instruments on the serial port, for instance, that may send unsolicited data. Therefore, the server also provides a mechanism to communicate these unsolicited responses to the client.

In addition to directly executed commands, the server also provides a means to run commands in the background, at scheduled intervals.

1.3. Command and Data Packets

Although the ISOC server command set consists of human readable command words and parameters, information is often exchanged in binary form. For this reason, a simple length-prefixed protocol is used for sending and receiving data. All transmissions, be it commands from the client, or replies from the server, are prefixed with a four-byte word indicating the length of the following transmission. Because if this protocol, it is not possible to exercise the ISOC server from a simple TCP socket client such as a telnet client program.

1.4. Data Formats

1.4.1. Commands

All commands sent by the client to the server are length-prefixed ASCII strings terminated by the newline ('\n') character. The length is encoded as a four-byte (LSB) integer. For more information, see Appendix B.

Commands that are processed by the server itself are prefixed by the forward slash ('/') character. Any command line that does not begin with a forward slash is interpreted as a command for the instrument to which the socket in use has been connected to.

Commands that are sent to an instrument must terminate with the semicolon (';') or question mark ('?') character. Due to the nature of the GPIB bus (used for connecting to most instruments) an application program is required to know in advance when to read data from the instrument. A de facto standard for GPIB instruments is to use the question mark character to indicate a query (which produces output), as distinguished from ordinary commands (terminated by a semicolon) which do not produce output. Although many instruments do not actually require that the question mark or semicolon be appended, the ISOC service does require this in order to distinguish queries from commands that produce no output.

1.4.2. Server Responses

All responses sent by the server to the client are length-prefixed: a four-byte (LSB) integer indicates the number of bytes that follow. Appendix B describes a simple set of Windows inline functions that implement length-prefixed data exchange on a socket.

Server error messages follow a simple format: a forward slash, followed by a two-digit error code, a colon, and an English-language description of the error. The two-digit code shall assist applications in evaluating the error condition. It is not necessary for an application to use, or display, the error description.

1.4.3. Trace Data

Trace data is presented as a length-prefixed block of one-byte or two-byte unsigned integers. How the server extracts trace data from instrument output is explained below, where the /T command is described. If the trace vertical resolution is less than 256, the trace is represented as single-byte values; otherwise, two-byte values are used. In any case, a trace data block is length-prefixed and contains the following elements:

Byte 0:	Trace number (1-3)
Bytes 1-4:	Packed date-and-time
Bytes 5-6:	Trace width
Bytes 7-8:	Trace height
Bytes 9...:	Trace values

The trace header is defined in `sendrecv.h` as follows:

```
typedef struct _PACKEDDATETIME
{
      unsigned long nYear:6;  // 1998-2061 (Y2062 problem, hehe)
      unsigned long nMonth:4;
      unsigned long nDay:5;
      unsigned long nHour:5;
      unsigned long nMin:6;
      unsigned long nSec:6;
} PACKEDDATETIME;

typedef struct _TRACEHDR
{
      PACKEDDATETIME pdtStamp;
      unsigned short nHRes;
      unsigned short nVRes;
} TRACEHDR;
```

Conversion of packed date-time values is assisted by two helper functions, also declared in `sendrecv.h`:

```
inline void SystemTimeToPackedTime(SYSTEMTIME *pst, PACKEDDATETIME *ppdt);
inline void PackedTimeToSystemTime(PACKEDDATETIME *ppdt, SYSTEMTIME *pst);
```

Trace data blocks are sent to the designated UDP port on the client computer, as specified by the `/U` command. The trace data block always contains one complete trace (i.e., one screenful.)

Note that the same format is used to send the output of the background command (specified using `/B`) to the client. In this case, the trace number (byte 0) is set to zero, and the trace values are replaced by the output of the command that is specified via `/B`.

A client application that processes input on the UDP port would first examine byte 0 to determine whether the data represents a trace or the output of a background command. It would then extract the date and time stamp, and save it for use in subsequent `/K` (keepalive) commands.

1.4.4. Audio Data

Audio data is sent only when the client connects to a special “audio” instrument. In this case, no background commands or traces are processed. However, audio is continuously read from the server’s audio hardware and processed, compressed as needed, and sent to the client’s UDP port.

An audio data block is prefixed by a six-byte header:

Byte 0:	Trace number (always 0)
Bytes 1-4:	Packed date-and-time
Byte 5:	Flags
Bytes 6...:	Audio stream

Presently, the following bit flags (byte 5) are defined:

Bit 0: Set to true if any form of compression is in use
Bit 1: 2X compression in use

Bit 2: GSM compression is in use
Bit 3: Channel A (left) data present in data block
Bit 4: Channel B (right) data present in data block

Audio data consists of an optionally compressed 8-bit PCM stream, sampled at 8000 kHz. Two forms of compression are used: 2X compression effectively reduces the sample rate to 4000 kHz (at a significant loss of quality), whereas GSM compression applies a standard GSM compression algorithm at moderate data loss. Normally, 2X compression is not required, as GSM compression allows a single audio channel to be transmitted over an ordinary modem connection with enough bandwidth left over for trace data and instrument control.

1.5. ISOC Server Command Set

Below is a complete list of the commands that the ISOC server accepts.

> NOTE: syntactical elements are shown with a fixed-pitch font. Syntactical elements in italics are variable elements; e.g., parameters supplied by the client, or data provided by the server. Syntactical elements enclosed in square brackets are optional.

`/1:`*`command-string`*

Treat *command-string* as a single command and send it to the instrument. If the string ends with a question mark, wait for data from the instrument.

The server normally parses a command that is sent to the instrument, and treats any semicolons that are part of the string as command separators. I.e., the string "command1;command2;" is sent to the instrument in two separate transactions. The /1 command makes it possible to send such compound commands to the instrument all at once. Not all instruments can process multiple commands this way, but for those that do, this syntax may provide a small performance gain. Additionally, the syntax also permits the use of commands that have semicolons embedded in them.

`/a`*`identifier`*`[:`*`db`*`:`*`mode`*`]`

Request calibration data for the instrument identified by *identifier*. This command is used to obtain a block of binary data that represents calibration information for the selected instrument. Calibration information in the present implementation is available for ICOM receivers and the antenna rotator instrument.

The *db* and *mode* parameters are optional and are used with ICOM receivers only. ICOM receivers have separate calibration data sets with 1dB, 2dB, and 3dB accuracy for each of the instrument's various operating modes (demodulator+bandwidth setting.)

In response to this query for an ICOM instrument, the server sends one or more digit sequences, each consisting of 12 decimal digits, followed by 232 hexadecimal digits. Each of these sequences represents a calibration range. The first 12-digit number is the upper frequency limit of the range (the lower limit is either 0 or the upper limit of the previous range; ranges are

listed in incremental order.) The 232 hexadecimal digits represent 116 one-byte numbers; these represent the ICOM signal levels (in ICOM units) corresponding with -10 to +105 dBm.

The response antenna rotators is a fixed-length 32-byte binary sequence representing 16 2-byte numbers in big-endian (Intel) encoding. The sequence contains the following parameters:

Azimuth offset (in tenth of degrees)
Azimuth resolution
Azimuth minimum (hardware)
Azimuth minimum (software)
Azimuth maximum (hardware)
Azimuth maximum (software)
Azimuth minimum
Azimuth maximum

The same set of parameters is then repeated for elevation.

The hardware minima and maxima represent the absolute limits of the instrument (in degrees). Software minima and maxima are site specific limits that should not be exceeded. The 'minimum' and 'maximum' values are calibration parameters; they correspond with the hardware minimum and maximum in instrument units. The instrument is assumed to have a linear response within these limits.

This command may return the following error if no calibration data is available:

```
/31: No calibration data
```

/b*interval*,*command-string*

Execute *command-string* every *interval* milliseconds. The string may contain one or more commands to be sent to the instrument. Commands are terminated by a question mark or semicolon; a question mark indicates that the instrument is to be polled for a response, which is then communicated to the client via the UDP interface.

The command may respond with the following error indications:

```
/08:not connected
/15:insufficient memory
```

This command can also be used without parameters, to query the current background command setting.

/c*instrument-identifier*

Connect to the instrument associated with *instrument-identifier*. If the command is not successful, the following errors may appear:

```
/02:connect failed
/09:already connected
```

```
/10:in use
/13:Cannot create UDP thread
/14:unknown instrument
/31:no memory
```

/d

Disconnect the current instrument without terminating the client-server session. May report the following errors:

```
/03:disconnected
/08:not connected
/12:disconnect failure (warning only)
```

/e*peername*

Set the peer name (i.e., tell server the name under which the client wishes to be known.)

/fc*ID*

Report audio channel. If successful, the command returns a line containing a single number (0 or 1) of the audio channel associated with the instrument identified by *ID*. May return the following error:

```
/14:unknown instrument
```

/fd*ID*

Get the default audio input tuned with the device identified by *ID*. If successful, the command returns a single line identifying the tuned device. May also return the following error:

```
/19: no default input
```

/fs*ID*

Report audio device. If successful, the command returns a line containing a single number identifying the device associated with the instrument identified by *ID*. May return the following error:

```
/14:unknown instrument
```

/g*n*

Add or don't add GPS header. GPS header is added if $n = 1$, not added otherwise.

/h

Report monitor center name. If no monitor center name has been configured on the server, returns the following error:

```
/32:unknown monitor center
```

`/i`

Report the interface identifier for the currently connected instrument. Most useful in case the same instrument is accessible via different interface types (e.g., RS-232 vs. GP-IB), utilizing a different command set.

If there is no current instrument, the following error is reported:

```
/08:not connected
```

`/mcommand`

Usage logging commands.

`/jc`

Query list of valid reason codes, as configured by this server's administrator. The result is a single line of text containing a tab-deliminted list of English/French reason codes with a vertical bar character ('|') separating the two. The string is terminated by a newline character.

`/jluser-ID:reason-code:GDOC`

Set the user identifier, reason code, and GDOC code for usage tracking.

`/jxlog-text`

Logs arbitrary text in the usage tracking log. Used by `ISOCSCAN.EXE`.

`/ktimestamp`

Keep-alive. This command is used to reset the server's command loss timer. The server uses the command loss timer to determine if it is necessary to reduce the data rate, and also to detect, and terminate, a 'hung' connection. The *timestamp* parameter to /K is an 8-digit hexadecimal value indicating a timestamp. A value of zero is equivalent to the current time, and allows the command loss timer to be reset. Without regular keep-alive packets or other traffic from the client, a server will always shut down a connection that has been idle too long.

Because /K commands are issued by clients frequently, they are not usually logged, unless their logging is enabled by the /- command.

`/l`

List instruments. This command lists all instruments that the server knows about in the following format:

```
/98:identifier|type|English-name|French-name|
```

The part between the semicolon and the terminating vertical bar (both inclusive) is repeated once for each instrument. New newline or carriage return characters are inserted.

The fields are as follows:

- Instrument identifier (an arbitrary string)
- Instrument type (presently one of ADV, ESN, HPS, SND, or UNK)
- English instrument name
- French instrument name
- Instrument user (DNS name of computer, if any, that is presently using the instrument.)

For instance, the server may return the following string, listing four instruments of which one (HPS) is in use (newlines have been inserted for formatting purposes, and are not part of the server's response):

```
/98:ADV|ADV|Advantest Spectrum Analyzer|Analyseur de spectre ADVANTEST|
:ESN|ESN|Rohde&Schwarz ESN Test Receiver|Récepteur ESN R&S|
:HPS|HPS|Hewlett-Packard Spectrum Analyzer|Analyseur de spectre|localhost
:SND|SND|Windows Multimedia Sound|Services de son digitalisés|
```

`/m`*command*

Matrix commands. Several subcommands are available to control matrix operation.

The virtual switch matrix model provided by the server implements the concept of an *input* signal source. Each instrument may have one ore more inputs associated with it; when the server is configured via the ISOC server configuration tool, these inputs and the corresponding switch matrix commands are listed.

The server also provides the concept of an input *connector*. This is used, for instance, with ICOM receivers, some of which have two or three input connectors in the back, to be used for different frequency ranges. On these instruments, in addition to selecting a signal source, a client program must also select the connector to which this signal source will be connected.

The server also supports audio switching. In this case, the "instrument" is the audio card on the server. Since this card has two inputs (the left and right channels of the stereo signal) support is provided in the command set to allow these two to be managed separately.

`/m?`

Query matrix availability. The server responds with a 1 if the switch matrix is available, and a 0 if it isn't. Matrix availability is checked in real time; in other words, if a switch matrix is connected to the system, or if one that has been previously connected is powered up, it is not necessary to restart the server.

`/m`*n*

Connect to connector port *n*. This command is valid for instruments with multiple connectors defined. In response to this command, the server sends to the switch matrix the command string that is associated with the specified connector.

/**mc***input*[:*c*]

Connect signal source identified by *input* to the current instrument, or disconnect from all inputs if *input* is a blank string. The optional *c* parameter is used to select channel A or channel B for the audio input device.

This command may report the following errors:

```
/17: undefined input
/20: input in use
/22: no input connected
/23: no matrix
```

If there is no error, the server responds with the identifier of the signal source that was just connected to the instrument, or the number 0 to indicate that all signal sources have been disconnected.

/**md**

List default input. Associated with each instrument is a default (hardwired) input that represents the signal the instrument receives when no switch matrix is available, or when the instrument has no switch matrix settings.

The default input is reported in the following format:

identifier|*English-name*|*French-name*

/**mi**

List inputs for the current instrument. For the currently connected instrument, all available inputs are listed in the following format:

identifier|*English-name*|*French-name*[|*user*]

If the instrument has no inputs, the following error is reported:

```
/24: Instrument has no inputs
```

/**mp**

List connector port parameters for the current instrument. Connector ports are listed in the following format:

Upper-frequency|*identifier*|*English-name*|*French-name*

/**ms**

Send notification to the current user of the instrument that a matrix position is requested. No error returns.

/**mt***input-identifier*

Show tune-with association for the specified input. Each input may optionally have a "tune-with" association with an instrument. This implies that when this input is selected for a virtual instrument, the "tune-with" associated instrument should be tuned when the user changes tuning settings. For instance, if the intermediate frequency (IF) output of a receiver is selected as the input of a spectrum analyzer, the spectrum analyzer should be tuned to a fixed frequency, and any tuning actions by the user should be transmitted to the receiver instead of the spectrum analyzer.

The tune-with associated instrument and parameters are listed in the following format:

Instrument;swap;frequency

The swap parameter is 1 if the signal is 'reversed' in the frequency domain, as it is done on the IF output of ESN receivers, for instance. The frequency parameter is the signal frequency (e.g., the intermediate frequency of a receiver) that the current instrument should be tuned to if this input is connected.

This command may produce the following messages if an error occurs:

```
/17:undefined input
/18: no tune-with association
```

/**my***input-identifier*

Show the antenna type name associated with the specified input. Standardized antenna type names are recorded during background scanning sessions in the standard format output file.

/**n[f]**

Query the long name of the instrument. Default response is the English name; the character F can be used to request the French name. If no device is presently connected, the following error is reported:

```
/08:not connected
```

/**o**

Request obstruction list. Obstruction lists are site-specific lists of obstructions that are used in the present implementation with the antenna rotator instrument. The obstruction list is provided in the following format:

```
Obstruction-name-1|Parameter-block-1|[Name-2|Pblock-2|...]
```

The parameter block for each obstruction consists of eight hexadecimal digits, representing 2 4-digit numbers. Each of the numbers is represented using big-endian byte encoding (e.g., 1000_{dec}=$3E8_{hex}$ is encoded as E803.) These two numbers represent the start and end angle of the obstruction in tenth of degrees.

`/pcommand-code`

Power management. Powers up or down the instrument that the client is presently connected to. The *command-code* part is a single character that may have the following values:

```
0:    Off
1:    On
2:    Boot (power cycle)
?:    Status query
```

If the command fails, one of the following errors is reported:

```
/10:in use
/14:unknown instrument
/25:no power settings
/26:no power bar
```

`/rchannel[threshold,file-name]`

Record audio. This command is applicable only in the case of audio devices. Once the command is issued, recording commences with an audio threshold value *threshold*, with the audio stream saved in Windows .WAV file format to *file-name*. The command confirms the successful commencement of audio recording with the following reply:

```
/00:OK
```

If no device is presently connected, the following error is reported:

```
/08:not connected
```

`/sflags`

Set sound transmission switches to *flags*. The value *flags* is a bitwise combination of the following values:

1: Compress the stream
2: Use rate-halving compression
4: Use GSM compression
8: Enable left audio channel
16: Enable right audio channel

If no instrument is presently connected, or if the instrument is not a sound instrument, the command returns the following error:

```
/08:not connected
```

`/sXinstrument-identifier`

Send notification to the current user of the instrument that the instrument is requested. X can be the character C (indicating a request of interactive use) or the character B (indicating a request

for background scanning use.) Currently, clients will display a message when a background use request is received, but not otherwise.

The command does not return a response.

`/tID,interval,offset,type,i-width,i-height,t-width,t-height,mode,command`

Trace command. This command instructs the server to periodically send *command* to the instrument to obtain a block of data representing a graphical trace, process this trace, and communicate the result to the client.

The command utilizes the following parameters:

ID: Trace identifier. The server supports up to three traces. For instance, whereas one trace may show instantaneous updates (clear write) the other two may be configured to show the signal minimum and maximum.

interval: The repeat interval (in milliseconds) with which the trace command will be re-executed.

offset: The trace is assumed to be a binary data block that is part of the instrument's response to the trace command. This parameter determines how many bytes are to be ignored (header bytes).

type: The type of the trace data that is reported by the instrument. 0 indicates byte data; 1 indicates big-endian words; 2 indicates little-endian words (i.e., word data in reverse byte order.)

i-width: The width of the trace as reported by the instrument (i.e., the number of trace data points.)

i-height: The height of the trace as reported by the instrument

t-width: The desired width of the trace to be sent to the client

t-height: The desired height of the trace to be sent to the client

mode: The resampling mode parameter determines how N data points in the trace reported by the instrument are converted into M data points for transmission to the client. For every point in the target set, there exist one or more (*k*) points in the source set. A mode parameter of 1 means "sample" mode; we pick the one data point in the source set that is closest to the desired horizontal position in the target set. A mode parameter of 2 means that the *k* points in the source set are averaged. A mode parameter of 3 means that their minimum is computed; 4 means maximum. Finally, a mode parameter of 0 ("binary bins") means that for every pair of points in the target set, the minimum and maximum are identified in the source set, and they are assigned to the target points in the (left-to-right) order in which they appear in the source set.

For example, if the source set contains six data points and the target set contains two data points, these five algorithms yield the following values:

	Source	25 15 11 9 2 1
0.	Minimax	25 1
1.	Sample	15 2
2.	Average	17 4
3.	Minimum	11 1
4.	Maximum	25 9

The trace command may report the following errors in case of failure:

```
/08:not connected
/11:syntax error
/15:insufficient memory
/16:not supported
```

Here is a specific example demonstrating the use of this command. Suppose we are communicating with an HP8594E Spectrum Analyzer. This analyzer can be configured to send trace data in the form of 400 two-byte values, each representing a value between 0 and 8000, using the following instrument commands:

```
TDF A;MDS W;
```

After this command is sent, we can obtain traces from the instrument using the TRA? command. The trace data begins on the fourth byte of the data block received in response to this command. If we wish to obtain the trace every 150 milliseconds in the form of 200 1-byte values, we need to issue the following command to the ISOC server:

```
/T1:150,4,2,400,8000,200,200,0,TRA?
```

The format of the resulting trace data that is sent to the client's UDP port is discussed in section 1.4.3.

`/`**u***port-number*

Instruct the server to send UDP packets to port *port-number* on the client. Use of the /U command is required before the client can receive background command results, graphical trace updates, or streaming audio.

If no instrument is currently connected, the following error is reported:

```
/08:not connected
```

`/`**v***command*

Manage server-side variables. Allows the client to set, and read the values of, variables stored by the server. These parameters are persistent and stored in the Registry. Several subcommands are available.

/**vr*varname***

Retrieve the value of the variable identified by ***varname***. May report the following errors:

```
/08:not connected
/15:insufficient memory
/35:error accessing variables
/36:variable not found
```

/**vs*varname*:*value***

Sets the value of the variable identified by ***varname*** to ***value***. May report the following errors:

```
/08:not connected
/11:syntax error
/15:insufficient memory
/35:error accessing variables
/36:variable not found
```

/**w***timeout*

Change timeout period. This command is especially relevant with instruments controlled via the GP-IB interface. By default, the ISOC uses a 1 second timeout when reading from the instrument; for commands that take a long time to execute (e.g., calibration), the client may opt to change this timeout value before issuing the command. The *timeout* value is in milliseconds.

If no instrument is presently connected, issuing this command results in the following error:

```
/08:not connected
```

/**x**

Disconnect the current instrument and terminate the client-server connection. May report the following errors:

```
/03:disconnected
/04:goodbye
/08:not connected
/12:disconnect failure (warning only)
```

/**z***command*

Set line terminator string. Used by the TCP/IP and RS-232 drivers. Several subcommands are available.

/**zr*string***

Sets the line terminator string to ***string*** for received data. The server will use this string to determine when a logical line ends in the data stream received from the instrument. The

standard ‘C’ notation is accepted for the carriage return (‘\r’) and line feed (‘\n’) symbols. No error returns. Example usage:

```
/zr\r\n
```

/zsstring

Sets the line terminator string to ***string*** for data to be sent to the instrument. The server will use this string to terminate lines of command sent to the instrument. The standard ‘C’ notation is accepted for the carriage return (‘\r’) and line feed (‘\n’) symbols. No error returns. Example usage:

```
/zr\r\n
```

/ztnum

Set send timeout delay to ***num*** milliseconds. The server will wait this amount of time between sending data lines to the instrument. No error returns.

/z#C

Sets “block character” to *C*. If defined, it enables a special data format for data received from the instrument: if a line begins with this character, the next character is assumed to be an ASCII digit that determines the length (0-9 characters) of the rest of the data line. No error returns.***Explain.... May report the following errors:

/-

Enable or suppress logging of /K (keep-alive) commands.

/?

Query server status. In the present implementation, this command returns a constant message indicating a healthy server:

```
/99:still alive
```

1.6. A Typical Client-Server Session

The following example demonstrates the use of many of the ISOC server commands in a typical client server session.

In this session, a client connects to the ISOC server and starts a session with the HP8594E Spectrum Analyzer instrument.

The session begins by establishing a TCP socket connection to port 25449 on the ISOC server. Once two-way communication is established, the server sends a 2-byte authentication code to the client, to which the client must respond appropriately.

Before connecting to a specific instrument, the client may request a list of instruments using the /L command:

```
/L
```

In response to this command, the server sends the list of instruments to the client. For instance, if only one instrument is present on the server, the response string will look like this:

```
/98:HPS|HPS|HP8594E Spectrum Analyzer|Analyseur de spectre HP8594E|
```

To actually connect to the instrument, the client would issue the following command:

```
/cHPS
```

The client may wish to receive background command results and graphical trace data on a UDP port. For instance, if the UDP port number is 27001, the client would send the following command next to the server:

```
/u27001
```

Next, the client initializes the spectrum analyzer instrument. The commands sent are instrument-specific:

```
TDF A;MDS W;CONTS;MKACT 1;MKFCR 0;MKFC OFF;MKTRACK OFF;
HD;AUTO;LG;DEMOD OFF;SPEAKER OFF;
```

Since these command strings are not preceded by a forward slash, they are forwarded to the instrument.

The client may wish to inquire as to whether a switch matrix is present:

```
/M?
```

The presence of the switch matrix is indicated by a response string containing the number 1. A 0 indicates that matrix functionality is not available. Even in this case, the client may opt to inquire about any default inputs that may be defined for this instrument:

```
/MD
```

To actually obtain a graphical trace repeatedly from the spectrum analyzer, the client must instruct the server to periodically send a `TRA?` query to the instrument and interpret the result. This is done using the following command string:

```
/t1:150,4,2,400,8000,200,200,0,TRA?
```

This command string provides access to one of the most powerful functions of the ISOC server. The command letter T identifies this as a TRACE command; the digit that follows indicates that this will be trace 1. The server can process three independent trace commands simultaneously.

The first number after the colon, 150, indicates that this command should be processed every 150 milliseconds. In other words, network and instrument performance permitting, approximately six traces per second will be obtained from the instrument.

The next several numbers tell the server how the instrument's binary response is to be interpreted. The Hewlett-Packard spectrum analyzer sends the trace response in the form of a 400-byte data block representing 400 samples across the trace window. The actual data block is preceded by a four-byte header, which the ISOC server ignores. The data format is little-endian (most significant byte first.) Each data point is represented by a two-byte number with a range of 0 to 8000. Here, we ask the server to resample this data with 200 sample points, each with an amplitude range of 0 to 200. The resampling method requested is the "binary bins" or "minimax" method. To summarize, the following numerical parameters are used:

150 .... Sample interval in milliseconds
4 ..... Byte offset (start of data block within the instrument's response)
2 ..... Instrument data format is little-endian
400 .... Number of sample points
8000 .. Sample amplitude range
200 .... Number of sample points requested after resampling
200 .... Sample amplitude range requested after resampling
0 ..... Resampling method requested is binary bins

Immediately after this command is issued, the server begins to obtain traces, sending the resampled trace result to the UDP port previously specified using the /U command. Of course, the client application must listen for incoming data on this port, and have the capability to interpret and process graphical trace data.

The client can then proceed to set or query instrument settings. For instance, to obtain the current center frequency setting of the spectrum analyzer, the client would issue the CF? command and wait for a numerical response.

In order to ensure proper client operation, it may be necessary to obtain instrument settings that define how the instrument interprets command parameters or formats its responses. For instance, the AUNITS? command can be used to obtain the current amplitude units in use by the instrument.

Of particular interest on the HP spectrum analyzer is the INZ? command, which can be used to obtain the current input impedance setting. This setting is essential in order to accurately convert between amplitude units such as dBm and dBμV.

This sample session demonstrates the essential features of the ISOC server. More complex sessions may include switch matrix operations, utilizing the power bar to reset an instrument, or using interconnected instruments for "click-and-tune" functionality.

It should be noted that much of this functionality (in particular, client authentication, length-prefixed command processing, instrument commanding, trace display, and digital audio playback) is automated through the ISOCInstrument control. Whenever possible, it is recommended that instead of directly connecting to the ISOC server, applications should make use of the services of this OLE/COM control component.

1.7. Rate Adaptive Algorithm

Because the server expects connections from clients over an unreliable network, it needs to be protected against network connections that go down or clients that fail. Because connections can be mode over low-bandwidth networks, it is important to ensure that the connection is not overloaded by excess amounts of data.

Both these goals are accomplished by the rate-adaptive algorithm built into the server. This algorithm requires cooperation by the client and thus may have implications on client program design.

In essence, the server expects the client to send commands regularly. If the client is not heard from for an extended period of time, the server assumes that the connection is lost and terminates the session.

The server also expects the client to regularly send `/K` commands with the most recent time stamp (if any) received as part of a UDP packet. The server uses this to calculate packet turnaround time and, if necessary, reduce the data rate.

If no commands are received at all for 15 seconds, the server “throttles down”, reducing the data rate. This reduction in the data rate may mean less frequently executed trace commands, increased audio compression, or the complete suppression of audio data.

If no commands are received in 4 consecutive 15 second periods, the server terminates the connection. The server also throttles down if the turnaround time (as measured through the /K command) is excessive.

The server can also “throttle up” if turnaround time is low, as measured through any `/K` commands received. Experiments show that this algorithm is sufficient to maintain good data flow even on severely restricted communication channels (such as a low-bandwidth cellular connection.)

1.8. A Simple Example

The following simple C++ program connects to an ISOC server and obtains the list of instruments available on that server.

Note that this example only demonstrates the very essentials of communicating with an ISOC server, and does not make use of more advanced options such as communicating with an instrument, using background commands and trace or audio processing, or using "keepalive" packets. Nevertheless, it can be used as a starting point, a basis for developing ISOCSVC-compatible client applications.

To compile this program, type:

```
CL -GX ISOCTEST.CPP WSOCK32.LIB ISOCDev.lib
```

Note that this assumes that `ISOCDev.h`, `sendrecv.h`, and `ISOCDev.lib` are present in the current directory. When executing the program, `ISOCDev.dll` must reside in the current directory.

```
#include <iostream>
#include "ISOCDev.h"
#include "sendrecv.h"

void main(void)
{
      SOCKET s;
      SOCKADDR_IN sin;
      WSADATA wsaData;
      int nLen;
      char pBuf[4096];

      WSAStartup(0x0101, &wsaData);

      s = socket(AF_INET, SOCK_STREAM, 0);
      if (s == INVALID_SOCKET)
      {
            cerr << "Could not create socket." << endl;
            exit(1);
      }
      sin.sin_family = AF_INET;
      sin.sin_port = htons(25449);
      sin.sin_addr.s_addr = 0x100007F;    // 127.0.0.1 = localhost
      if (connect(s, (LPSOCKADDR)&sin, sizeof(sin)))
      {
            cerr << "Unable to connect." << endl;
            closesocket(s);
            exit(1);
      }

      Sleep(250); // Without some delay, sends that follow immediately
                  // fail. The 250 ms is for the benefit of dial-up W95
                  // connections that appear to require that.

      if (!ISOCAuthenticateClient(s))
      {
            closesocket(s);
            cerr << "Authentication failure." << endl;
            exit(1);
      }

      sendstr(s, "/L\n");
```

```
	nLen = recvwithlength(s, pBuf, sizeof(pBuf), 0);
	if (nLen == 0)
	{
		cerr << "No data received." << endl;
		closesocket(s);
		exit(1);
	}
	cout << "Instrument list received:" << endl;
	cout.flush();
	cout.write(pBuf, nLen);
	cout << endl;
	closesocket(s);
	exit(0);
}
```

The ISOC background scheduler is a server application that is capable of performing unattended scans at scheduled times using ESN and ICOM receivers. It is also capable of recording audio at scheduled times.

Similarly to the main ISOC server, the ISOC background scheduler is also a server application that listens for incoming TCP socket connections. The port number in use by the background scheduler is 25450.

When an incoming connection is detected, the background scheduler expects the client to authenticate itself with the same authentication method that is used by the main ISOC server. Once authentication is completed successfully, the background scheduler is ready to respond to queries.

Unlike the main ISOC server, the background scheduler does not utilize a length-prefixed protocol for transmitting or receiving commands or data. Instead, a plain ASCII protocol with human-readable commands and responses is used. Therefore, except for the initial authentication sequence, it is possible to control the background scheduler using a text-only tool such as a telnet program.

2.1. Schedule Entries

The ISOC background scheduler maintains a list of schedule entries. Each schedule entry specifies an instrument, a set of instrument settings, input and output files, and times. For instance, a typical schedule entry may look like the following:

```
ESN|2000/12/18|0000/00/00|300000000000000000000000000000000000000000|
FMBCAST|FMSCAN|10|ANT1|3|FM Monitoring|HOST22|;
```

As can be seen, a schedule entry consists of several fields, which are delimited by the vertical bar character. Here's a list of these fields from the above example and their interpretation:

ESN:	The name of the instrument used for the background operation. Must be an ESN or ICOM radio, or an audio input device.
2001/12/18:	The date when this background schedule becomes active
0000/00/00:	The date after which this schedule is no longer active (all zeroes mean run forever)
300.000:	42 hexadecimal digits (0..F) each representing one of 168 one hour period in the week. This schedule will run from 0000 to 0200 every Monday.
FMBCAST:	The name of the source file containing the list of frequencies to be monitored
FMSCAN:	The name of the target file containing the scan results
10:	Attenuation
ANT1:	Name of signal source
3:	Bandwidth selection (instrument-specific)

FM Monitoring:	Name of this session
HOST22:	Name of the client host where this schedule entry originated
;	Extra initialization commands. A semicolon indicates no extra commands

This format is used throughout the background scheduler, both for storing schedule information in the Registry, and for exchanging schedule information with client programs.

2.2. Input Files and Ranges

When the background scheduler operates a radio, it sets the radio to scan a series of frequencies as specified in an input file. Two input file formats are recognized: .LST and .SST files.

Of the two, .LST files are the simpler; they simply contain a list of frequency values (in units of MHz), one per line. The other file format, .SST files, are inherited from a previous version of the ISOC system and contain, in addition to frequencies, header information and instrument settings, all of which are ignored by the background scheduler. It is recommended that newly created frequency lists be stored in the simpler, .LST format.

Files of a third type, .SCL files, contain lists of .SST and .LST files, one per line.

The background scheduler recognizes a special type of input specification as defining a frequency range. When the input specification begins with the colon character (which is never part of a valid file name) it is assumed to specify a range in the following format:

```
:start-frequency;end-frequency;step-size
```

For instance, scanning the FM broadcast band using 200 kHz steps can be accomplished using the following input specification:

```
:88100000;106900000;200000
```

Finally, the input specification may contain a single blank character. This is used, for instance, when the background scheduler records audio and no frequency list is required.

2.3. Output Files

The result of background scanning with ESN and ICOM radios is a file with the .ESN extension. The file format is inherited from an older version of the ISOC system. It contains the result of one or more scanning sessions in binary format. Specifications for the .ESN file format are available separately.

For audio recordings, the output file format is a monaural .WAV file with 8000 8-bit samples per second. Additional fields in the .WAV file make it possible to identify the start and stop time of VOX-controlled recording sessions.

The server never overwrites existing files. New files are created using the root name specified in the schedule entry, to which the current date is appended.

2.4. Background Scheduler Command Set

a

Adds an entry to the schedule. The server replies with a string containing the words "OK" or "FAIL" depending on whether or not the operation was successful.

as*id*|*start-date*|*end-date*|*hour-bits*|*host*

Add a new job to the schedule. The start date (yyyy/mm/dd format), end date, and 42 hexadecimal digits representing 168 hours of the week must be specified, along with the name of the submitting host.

Example (long lines broken up to fit the page):

```
AESN Test|1999/01/11|1999/12/31|23:59|
000000FFFFFFFFFFFFFFFFFFFFFF000000000000|ISOC3
```

at*id*|*type*|*instrument*|*input*|*output*|*post-proc*|*att*|*ant*|*BW*|*init-string*

Add a task to an existing job. The task id contains the schedule and task identifiers, separated by a colon. The task type, instrument, input and output files, post-processing batch file, default attenuation, antenna, and bandwidth settings, as well as an optional instrument initialization string must be specified.

Example:

```
AESN Test:ESN Rcvr|ESN|BAND1.LST|BAND1.ESN|POSTPROC.BAT|0|ANT1|9000|;
```

Note that the *init* parameter must always have a value. If no initialization string is needed, use a semicolon (';').

c*command*

Catalog operations. Subcommands are used to enumerate input and output files stored on the server.

cb

List batch files. All existing .BAT and .CMD files are listed, complete with their filename extension.

cf

List French-language help files. All existing .MAF files are listed, complete with their filename extension.

ch

List English-language help files. All existing .MAN files are listed, complete with their filename extension.

ci

List input files. All .LST, .SST, and .SCL files are listed, complete with their filename extension.

cl

List log files. All existing .LOG files are listed, complete with their filename extension.

cm

List MP3 audio files. All existing .MP3 files are listed, complete with their filename extension.

co

List output files. All existing .ESN files are listed, complete with their filename extension.

cs

List all files. All existing files are listed, complete with their filename extension.

cw

List waveform audio files. All existing .WAV files are listed, complete with their filename extension.

cx

List Excel files. All existing .XLS files are listed, complete with their filename extension.

c*

List all files of a known type. All existing files are listed, complete with their filename extension.

d

Delete a schedule entry.

ds*id*

Delete a job. All tasks of the selected job will also be removed.

dt*id*

Delete a task. The *id* parameter must contain the job and task name separated by a colon.

e*instrument*

List default (hardwired) input for the requested instrument. The response contains the identifier, English name, and French name of the default input, separated by the vertical bar character. If there is no default input, the string FAIL is returned. For instruments with multiple default inputs, the response takes the following form:

```
0|[Multiple hardwired connectors]|[Plusieurs connections câblées]
```

f*instrument*

List filter values for the requested instrument. This command is necessary because radios (in particular, ESN receivers) may have non-standard IF filters installed. A typical response to this command may appear as follows:

```
3000,6000,9000,20000,120000,250000
```

Applications would typically use this command to generate an on-screen control (e.g., list box) in which available filter values are presented to the user.

Please note that the filter set returned represents the last value queried from the instrument; i.e., no query is performed in response to this command.

i

List available instruments for background operation. Instruments are listed, one per line, using the following format:

identifier|*type*|*English-name*|*French-name*

After the last instrument, the string "OK" appears on a line by itself, indicating the end of the list.

For instance, if the server has a single ESN instrument, the server may respond as follows:

```
ESN1|ESN|Rohde&Schwarz ESN Test Receiver|Récepteur ESN R&S
OK
```

k

Keep-alive command. The server responds with the current date and time:

```
OK 2001/01/04 10:18:48
```

l

List all scheduled sessions. Sessions are listed one per line in the following format:

identifier:*rate*|*schedule-entry*

The word OK by itself on a line indicates the end of the list has been reached.

The identifier is a numeric value that uniquely identifies schedule entries. Identifiers are valid for the current session only; if the client disconnects from the background scheduler server, a different set of identifiers may be used upon reconnection.

The list format is almost identical to the format used in the A command. The only difference is that every line has an extra element at the beginning: a numeric value. If this is –1, the scan is not currently executing; if it is 0 or a positive number, the scan is active. A positive number represents a near real-time measurement of the scanning rate, in units of channels per second.

For example, in response to the L command the server may output the following:

```
1:329|ESN|1999-01-11|23:00|1999-01-11|23:59|TEST.SCL|TEST|0|0|9000|;
2:-1|ESN|1999-01-12|23:00|1999-01-12|23:59|TEST.SCL|TEST|0|0|9000|;
```

m

Get monitoring center (server) name.

q

Quit (terminate current client session.)

r*instrument*

List available inputs for the selected instrument. The list contains one entry per line in the following format:

identifier|*English-name*|*French-name*

After the last input, the word OK appears by itself on a line to indicate that the end of the list has been reached.

s

Query current status of schedule entry.

ss*id*

Query a job's status.

st*id*

Query a task's status.

t*command*

File transfer commands. Files can be uploaded to, or downloaded from, the server. A simple length-prefixed protocol is used, with no error correction or flow control facility; it is assumed that error-free transmission is provided for by the underlying TCP socket mechanism.

The (binary) file content is preceded by a 4-byte value indicating the number of bytes to follow. The length is Intel-encoded (big-endian.)

td

Download a file to the client. Upon receipt of this command, the server commences sending the file. An error prior to transmission is indicated by a zero length value sent. An error during transmission results in the TCP socket being closed.

th

Type (list) named file in postproc directory.

tl

Obtain length of named file (NB: This command is not yet implemented.)

tr

Remove named file.

tt

Type (list as text) named file. The contents of the file will be listed on output.

tu

Upload a file to the server. Immediately after this command, the client must begin transmitting the file, prefixed by a four-byte length value. Errors during reception cause the server to close the TCP socket.

u*identifier*

DEPRECATED: List default input for a specific instrument.

v*identifier*

Verify a task. Presently, verification checks if the task's frequency list a) contains duplicate entries, or b) longer than the number of frequencies allowed by the instrument.

x*identifier*

Expunge a task (terminate running measurements.)

y*identifier*

Resume a task that was interrupted.

2.5. A Simple Example

The following program connects to the ISOCSCAN server on the machine on which it executes, obtains and prints the list of scheduled scans, and then terminates. It demonstrates the basic operations of the ISOCSCAN server.

To compile this program, type:

```
CL -GX SCANTEST.CPP WSOCK32.LIB ISOCDev.lib
```

Note that this assumes that `ISOCDev.h`, `sendrecv.h`, and `ISOCDev.lib` are present in the current directory. When executing the program, `ISOCDev.dll` must reside in the current directory.

```
#include <iostream>
#include "D:/VIKTOR/CLIENTS/INDUSTRY/ISOC/ISOCNT/ISOCDev/ISOCDev.h"
#include "D:/VIKTOR/CLIENTS/INDUSTRY/ISOC/ISOCNT/sendrecv/sendrecv.h"

void main(void)
{
      SOCKET s;
      SOCKADDR_IN sin;
      WSADATA wsaData;
      int nLen;
      char pBuf[4096];

      WSAStartup(0x0101, &wsaData);

      s = socket(AF_INET, SOCK_STREAM, 0);
      if (s == INVALID_SOCKET)
      {
            cerr << "Could not create socket." << endl;
            exit(1);
      }
      sin.sin_family = AF_INET;
      sin.sin_port = htons(25450);
      sin.sin_addr.s_addr = 0x100007F;    // 127.0.0.1 = localhost
      if (connect(s, (LPSOCKADDR)&sin, sizeof(sin)))
      {
            cerr << "Unable to connect." << endl;
            closesocket(s);
            exit(1);
      }

      Sleep(250); // Without some delay, sends that follow immediately
                  // fail. The 250 ms is for the benefit of dial-up W95
                  // connections that appear to require that.
```

```
	if (!ISOCAuthenticateClient(s))
	{
		closesocket(s);
		cerr << "Authentication failure." << endl;
		exit(1);
	}

	send(s, "L\n", 2, 0);
	cout << "Schedule:" << endl;
	while (1)
	{
		nLen = recv(s, pBuf, sizeof(pBuf), 0);
		if (nLen == 0)
		{
			cerr << "No data received." << endl;
			closesocket(s);
			exit(1);
		}
		for (char *p = pBuf; p != NULL && *p != '\0';)
		{
			char *q = strchr(p, '\n');
			if (q) *q = '\0';
			cout << p << endl;
			if (!strcmp(p, "OK")) goto DONE;
			p = q;
			if (p) p++;
		}
	}
DONE:
	closesocket(s);
	exit(0);
}
```

The complex functionality of the ISOC instrument server is made more accessible through the ISOCInstrument.dll.

This component is an ActiveX control that provides the following functionality:

1. Automatic connection to servers and instruments
2. Processing and visual presentation of trace data
3. Receiving and processing background command results
4. A simple set of methods that can be used from Visual C++, Visual Basic, Web scripts, and other programming environments.

There are several ways for an application program to utilize the capabilities of this control.

First, an application may use a hidden control for instrument communication. Any information displayed will be displayed by the application itself, not by the control, which remains invisible to the user.

Second, an application may make the control visible, allowing it to display trace data.

Third, the application may keep the control invisible and use it for playing back digital audio.

These three modes of operation are not mutually exclusive, but since a single copy of the ISOCInstrument control can only be connected to one instrument at a time, performing multiple functions simultaneously generally requires multiple copies of the control.

Note that it is entirely normal for an application to utilize multiple copies of the control. This is generally the case if an application provides a means to connect to multiple instruments simultaneously.

Examples presented in this section use the syntax of Visual Basic, a language especially suited for controlling ActiveX objects. However, the ISOCInstrument control can be utilized from any programming language that is compatible with ActiveX technology, including Visual C++, Borland Delphi, and more.

3.1. Modes of Operation

The ISOCInstrument control can be used in three different modes of operation: in server control mode, in graphical trace mode, and in audio mode.

3.1.1. Server Control Mode

In server control mode, none of the ISOCInstrument control's advanced features are used. Instead, the control is used merely as a convenient tool for communicating with the ISOC server. This mode can be used, for instance, to obtain a list of instruments from the server.

In this mode, an application would utilize the `ServerConnect` method to connect to a server, and the `Transact` property to exchange information with it. Optionally, an application may also use the `Send` method and the `Receive` property, in conjunction with the `Receive` event.

If using it in this mode, an application will likely keep the ISOCInstrument control invisible.

The following Visual Basic code fragment is an example for this type of use. It obtains the list of instruments from the specified server and stores it in the string variable `SrvLst`:

```
Dim ISOC As Object, SrvLst as String
Set ISOC = CreateObject("ISOCInst.ISOCInst.1")
Call ISOC.ServerConnect("isoc.ic.gc.ca", 25449)
SrvLst = ISOC.Transact("/L" + chr(10))
```

Information about the `/L` command and the server's response can be found in section 1.3.6. After the above code fragment is executed, `SrvLst` may contain a string similar to the following (newlines, inserted here for printability, are not part of the server's response):

```
/98:ADV|ADV|Advantest Spectrum Analyzer|Analyseur de spectre ADVANTEST|
:ESN|ESN|Rohde&Schwarz ESN Test Receiver|Récepteur ESN R&S|
:HPS|HPS|Hewlett-Packard Spectrum Analyzer|Analyseur de spectre|localhost
:SND|SND|Windows Multimedia Sound|Services de son digitalisés|
```

3.1.2. Graphical Trace Mode

In graphical trace mode, an application would make the ISOCInstrument control visible, use it to connect to a specific instrument on a server, and request a graphical trace from that instrument. The steps an application must perform to obtain a trace are as follows:

1. Connect to the server
2. Connect to the desired instrument
3. Set up trace parameters
4. If not yet shown, display the control

For instance, a the following Visual Basic code fragment configures an ISOCInstrument control (`ISOC1`) in a dialog box to connect to an HP Spectrum Analyzer on a remote server and display a trace 4 times a second:

```
Call ISOC1.ServerConnect("localhost", 25449)
Call ISOC1.Connect("HPS")
Call ISOC1.Send("TDF A;MDS W;" + Chr(10))
ISOC1.TraceOffset = 4
ISOC1.TraceType = 2
ISOC1.TraceWidth = 400
ISOC1.TraceHeight = 8000
```

```
ISOC1.DisplayWidth = 200
ISOC1.DisplayHeight = 200
ISOC1.TraceResample = 0
ISOC1.DisplayMode(0) = 1
ISOC1.HorizontalGraticule = 8
ISOC1.VerticalGraticule = 10
ISOC1.GraticuleColor = &H20000C0
ISOC1.ForeColor = &H200FFFF
ISOC1.BackColor = &H20080C0
ISOC1.CursorColor = &H2FF0000
ISOC1.ShowTime = True
ISOC1.TraceCommand = "TRA?"
ISOC1.TraceInterval = 250
```

Note that it is a good idea to leave updating the `TraceCommand` and `TraceInterval` properties to the end, to ensure that the control does not attempt to acquire traces from the instrument when not all parameters are correctly set yet.

3.1.3. Audio Control Mode

In audio control mode, an application would create an invisible ISOCInstrument control, use it to connect to a sound source, and instruct it to play back streaming audio. The steps required to obtain streaming audio playback are as follows:

1. Connect to the server
2. Connect to the sound source
3. Set the sound control parameter
4. Set the volume

The following Visual Basic code fragment demonstrates how a control named ISOC1 can be made hidden and used to deliver streaming audio at one quarter of the maximum volume:

```
ISOC1.Visible = False
Call ISOC1.ServerConnect("isoc.ic.gc.ca", 25449)
Call ISOC1.SoundConnect("SND")
ISOC1.SoundFlags = 13
ISOC1.Volume = 16384
```

> You must use the SoundConnect method when connecting to an audio playback virtual instrument. Although the Connect method will also succeed, no audio will be heard.

Note that successful execution of this code requires that compatible sound hardware be installed on both the client and the server.

3.2. A Web Script Example

The example in figure 1 (shown at the end of this section) is a simple HTML page that utilizes the ISOCInstrument.dll to display a visual representation of the ESN receiver and provides graphical trace updates, audio, and a simple form-based control interface. This Web page can be

used with the Internet Explorer browser; other browsers may require plug-ins in order to support embedded custom controls.

This Web script creates two ISOCInstrument objects: the first (named RSESN) controls the ESN receiver, while the second (SND) does audio playback. Other elements in the page exist merely to provide a minimalistic, but useful, graphical user interface.

The Web page begins with the standard HTML preamble, followed immediately with a Visual Basic script:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="VBScript">
<!--
Dim FRval, SPval
Sub window_onload()
      call RSESN.ServerConnect("isoc.gc.ca", 25449)
      call RSESN.Connect("ESN")
```

The function that begins here is executed immediately after the Web page is loaded into the browser. It references objects like RSESN that are defined later in the page. It calls methods of RSESN to establish connection to the (hypothetical) ISOC server at isoc.gc.ca, and to open an instrument named ESN (presumably, an ESN receiver.) Next, we send some initialization commands to the receiver itself:

```
      call RSESN.Send("H OF;LE:F AS;ME:C L,FR;SPE 50,ON;MO IF;" + chr(10))
```

These commands, separated by semicolons, are sent in unaltered form to the receiver, setting up the receiver with operating parameters for working with our software.

The next set of commands establish settings for the background trace. This includes sending the background trace command to the ISOC server, and setting up the trace parameters:

```
      RSESN.TraceCommand = "SW:B?"
      RSESN.TraceOffset = 4
      RSESN.TraceType = 1
      RSESN.TraceWidth = 0
      RSESN.TraceHeight = 8000
      RSESN.DisplayWidth = 200
      RSESN.DisplayHeight = 200
      RSESN.TraceResample = 0
      RSESN.DisplayMode(0) = 1
      RSESN.TraceInterval = 1000
```

The trace command, `SW:B?`, causes the receiver to respond with a binary data block representing the trace display. This command will be sent to the receiver once every 1000 milliseconds, as established by the `TraceInterval` setting. Other settings establish the format of the trace data, and the desired size of the visible trace display.

Lastly, we set up a few cosmetic settings. The meaning of these is self-explanatory:

```
      RSESN.HorizontalGraticule = 8
      RSESN.VerticalGraticule = 10
      RSESN.GraticuleColor = &h020000C0
      RSESN.ForeColor = &h0200FFFF
      RSESN.BackColor = &h02004040
      RSESN.CursorColor = &h02FF0000
      RSESN.ShowTime = true
```

And some more boring stuff: we call two subroutines (defined later) in order to update the displayed values for the frequency and span:

```
      call SetFR(0)
      call SetSP(0)
```

That's it: we're done setting up the ESN receiver. We have, however, a second object to configure: one that will provide sound playback. Like RSESN, SND is an object defined later in the HTML file. Here we initialize it as follows:

```
      call SND.ServerConnect("isoc.gc.ca", 25449)
      call SND.SoundConnect("SND")
      SND.SoundFlags = 13
      SND.Volume = 65535
```

Before we leave, we do one more thing: we set the focus to a field (defined later) where the user can enter instrument commands:

```
      call COMMAND.focus()
end sub
```

The next subroutine is invoked whenever the RSESN object receives data from the server. Any text received is added to the contents of the field BUFFER.

```
Sub RSESN_Receive()
      buf = RSESN.Receive
      BUFFER.value = BUFFER.value + buf
end sub
```

The next subroutine is a helper function that is invoked whenever a mouse event occurs within the ESN receiver's display area. This subroutine draws a frequency or span marker depending on whether or not the Control key is depressed when the user clicks within the trace area.

```
Sub UpdateMarker()
      MK_SHIFT = 4
      if (RSESN.KeyFlags And 4) = 4 Then
            bSpan = true
      Else
            bSpan = false
      End If
      nXPos = RSESN.MousePosX
      RSESN.CursorPos = nXPos
      RSESN.SpanCursor = bSpan
      If bSpan Then
```

```
            fSpan = SPval - SPval / 100 * nXPos
            If fSpan < 0 Then
                  fSpan = -fSpan
            End If
            fSpan = fSpan / 1000
            strCaption = "sp: " + CStr(CLng(fSpan)) + " kHz"
      Else
            fFrq = FRval - SPval / 2 + SPval / 200 * nXPos
            fFrq = fFrq / 1000
            fLvl = RSESN.CursorLevel
            fLvl = fLvl * 80 / 200
            strCaption = "f: " + CStr(CLng(fFrq)) + " kHz" + chr(10)
            strCaption = strCaption + "l: " + CStr(CInt(fLvl)) + " dB"
      End If
      RSESN.Caption = strCaption
end sub
```

This subroutine is called in response to three mouse events:

```
Sub RSESN_LButtonDown()
      RSESN.ShowText = true
      call UpdateMarker
end sub
Sub RSESN_MouseMove()
      call UpdateMarker
end sub
Sub RSESN_LButtonUp()
      RSESN.CursorPos = -1
      RSESN.ShowText = false
end sub
```

We also need a function that sends any commands the user enters to the server:

```
Sub Send(buf)
      call RSESN.Send(buf + chr(10))
      call COMMAND.focus()
end sub
```

Lastly, two helper functions update the center frequency and span values, respectively. If these functions are called with a value of 0, they query the instrument for the current setting.

```
Sub SetFR(newFR)
      If newFR > 0 Then
            call RSESN.Send("FR " + newFR + ";" + chr(10))
      End If
      FRval = RSESN.Transact("FR?" + chr(10))
      FR.value = FRval
end sub
Sub SetSP(newSP)
      If newSP > 0 Then
            call RSESN.Send("SPA " + newSP + ";" + chr(10))
      End If
      SPval = RSESN.Transact("SPA?" + chr(10))
      SP.value = SPval
end sub
```

```
-->
</SCRIPT>
```

That's it for the scripts; the rest is plain HTML. First, we have some header text, followed by a series of radio buttons in a form; clicking each of these buttons causes the corresponding method of the RSESN object to be called.

```
<TITLE>Test Page</TITLE>
</HEAD>
<BODY>
<H1>Test Page: ISOC ActiveX Instrument Control</H1>
<H3><EM>RS ESN Virtual Instrument</EM></H3>
<P>This page contains the ISOC ActiveX Instrument Control, preconfigured to
connect to the RS ESN instrument.</BR></P>
<SMALL>Resampling method: <INPUT LANGUAGE="VBScript" TYPE="radio"
 ONCLICK="RSESN.TraceResample = 0" NAME="MODE" CHECKED>Peaks
 <INPUT LANGUAGE="VBScript" TYPE="radio"
 ONCLICK="RSESN.TraceResample = 1" NAME="MODE">Sample
 <INPUT LANGUAGE="VBScript" TYPE="radio"
 ONCLICK="RSESN.TraceResample = 2" NAME="MODE">Average
 <INPUT LANGUAGE="VBScript" TYPE="radio"
 ONCLICK="RSESN.TraceResample = 3" NAME="MODE">Minimum
 <INPUT LANGUAGE="VBScript" TYPE="radio"
 ONCLICK="RSESN.TraceResample = 4" NAME="MODE">Maximum</SMALL><BR>
<SMALL>Trace mode: <INPUT LANGUAGE="VBScript" TYPE="radio"
 ONCLICK="RSESN.DisplayMode(0) = 1" NAME="TRACE" CHECKED>Clear Write
 <INPUT LANGUAGE="VBScript" TYPE="radio"
 ONCLICK="RSESN.DisplayMode(0) = 2" NAME="TRACE">View
 <INPUT LANGUAGE="VBScript" TYPE="radio"
 ONCLICK="RSESN.DisplayMode(0) = 3" NAME="TRACE">Maximum
 <INPUT LANGUAGE="VBScript" TYPE="radio"
 ONCLICK="RSESN.DisplayMode(0) = 4" NAME="TRACE">Minimum
 <INPUT LANGUAGE="VBScript" TYPE="radio"
 ONCLICK="RSESN.DisplayMode(0) = 5" NAME="TRACE">Average</SMALL><BR>
```

Next, we have the controls themselves! The RSESN control is defined with a size of 400 by 300 pixels; this provides a good trace area for most screen resolutions. The SND control is defined, in contrast, with a size of 0x0; this control therefore remains invisible, which is exactly our intention, since we don't want a visual trace display to be associated with the sound playback.

```
<DIV ALIGN="LEFT">
<TABLE BORDER="2" STYLE="border: medium groove rgb(128,128,128)">
  <TR>
    <TD>
      <OBJECT ID="RSESN" WIDTH="400" HEIGHT="300" BORDER="3"
       CLASSID="CLSID:8494107C-0FE7-11D2-8E06-00E02910AE47"
       CODEBASE="ISOCInstrument.cab#Version=1,3,0,1">
      Sorry, your browser doesn't appear to support ActiveX.</OBJECT>
    </TD>
    <TD>
      <OBJECT ID="SND" WIDTH="0" HEIGHT="0" BORDER="0"
       CLASSID="CLSID:8494107C-0FE7-11D2-8E06-00E02910AE47"
       CODEBASE="ISOCInstrument.cab#Version=1,3,0,1">
      Sorry, your browser doesn't appear to support ActiveX.</OBJECT>
```

```
    </TD>
  </TR>
</TABLE>
</DIV>
<P>
```

Lastly, a few additional controls are provided for setting the frequency and span, and for sending commands directly to the server.

```
Frequency: <INPUT TYPE="text" SIZE="12" NAME="FR" STYLE="font-family:
monospace"
>
<INPUT LANGUAGE="VBScript" TYPE="submit" VALUE="Set"
 ONCLICK="SetFR(FR.value)" NAME="SETFRE">
Span: <INPUT TYPE="text" SIZE="12" NAME="SP" STYLE="font-family: monospace">
<INPUT LANGUAGE="VBScript" TYPE="submit" VALUE="Set"
 ONCLICK="SetSP(SP.value)" NAME="SETSPA"><BR>
<INPUT TYPE="text" SIZE="44" NAME="COMMAND" STYLE="font-family: monospace">
<INPUT LANGUAGE="VBScript" TYPE="submit" VALUE="Send"
 ONCLICK="call Send(COMMAND.value)" NAME="SEND"><BR>
Result:<BR>
<TEXTAREA ROWS="6" COLS="50" NAME="BUFFER"></TEXTAREA>
</P>
</BODY>
</HTML>
```

That's it. Only about 160 HTML lines, yet this page already provides comprehensive control of an ESN instrument and audio playback.

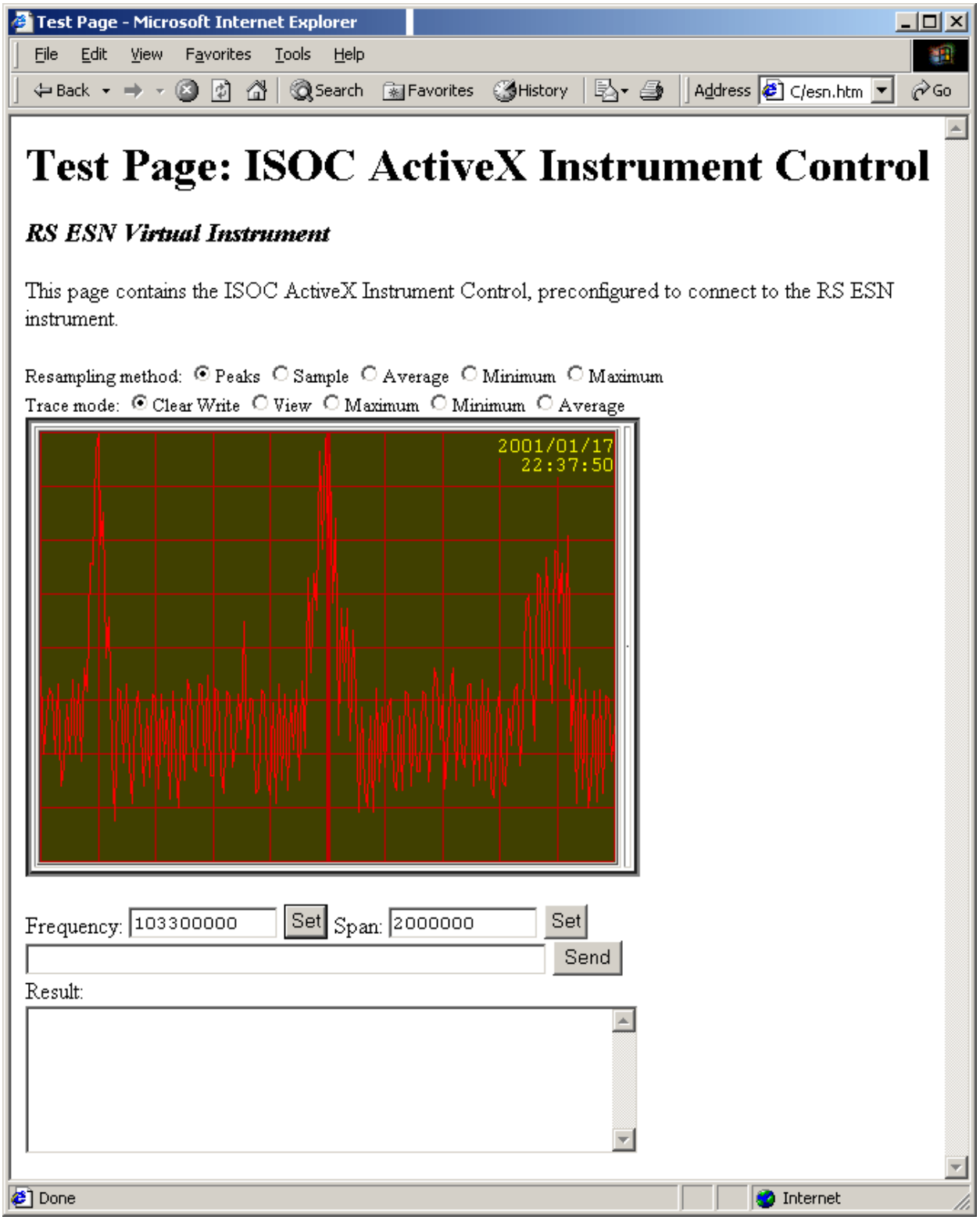

Test Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History Address C/esn.htm Go
Test Page: ISOC ActiveX Instrument Control
RS ESN Virtual Instrument
This page contains the ISOC ActiveX Instrument Control, preconfigured to connect to the RS ESN instrument.
Resampling method: Peaks Sample Average Minimum Maximum
Trace mode: Clear Write View Maximum Minimum Average
2001/01/17
22:37:50
Frequency: 103300000 Set Span: 2000000 Set
Send
Result:
Done
Internet

Figure 1.

3.3. Quick Reference

The following tables provide an overview of the properties, methods, and events defined by the ISOCInstrument.dll control.

Property Name	Type	Parameters	Description
BackColor	in/out	OLE_COLOR	trace background color
ForeColor	in/out	OLE_COLOR	text color
Caption	in/out	BSTR	trace annotation text
CursorColor	in/out	OLE_COLOR	cursor (local marker) color
GraticuleColor	in/out	OLE_COLOR	graticule color
MarkerColor	in/out	long, OLE_COLOR	color of specified (remote) marker
TraceColor	in/out	long, OLE_COLOR	color of specified trace
ShowText	in/out	BOOL	Annotation visible flag
ShowTime	in/out	BOOL	Timestamp visible flag
VerticalGraticule	in/out	long	Number of vertical graticules
HorizontalGraticule	in/out	long	Number of horizontal graticules
MarkerPos	in/out	long, long	Horizontal position of specified marker
MarkerTrace	in/out	long, long	Trace that specified marker is attached to
MarkerLevel	out	long, long	Vertical level of specified marker
DeltaMarker	in/out	long, BOOL	Delta marker flag for specified marker
CursorPos	in/out	long	Horizontal position of cursor (local marker)
CursorTrace	in/out	long	Trace that cursor is attached to
SpanCursor	in/out	BOOL	Cursor span mode flag
CursorLevel	out	long	Current level at cursor position
MousePosX	out	long	Current mouse horizontal position
MousePosY	out	long	Current mouse vertical position
TraceCommand	in/out	BSTR	Background trace command
TraceWidth	in/out	long	Instrument horizontal trace resolution
TraceHeight	in/out	long	Instrument vertical trace resolution
DisplayWidth	in/out	long	Desired horizontal trace resolution
DisplayHeight	in/out	long	Desired vertical trace resolution
DisplayHOffset	in/out	long	Trace display horizontal displacement
DisplayVOffset	in/out	long	Trace display vertical displacement
TraceResample	in/out	long	Trace resample mode
TraceType	in/out	long	Instrument trace data type
TraceOffset	in/out	long	Trace data offset in instrument data block
TraceInterval	in/out	long	Trace command execution interval
DisplayMode	in/out	long	Trace display mode
Invert	in/out	BOOL	Horizontal flip flag
Pause	in/out	BOOL	Trace display freeze
AverageInterval	out	long	Measured trace update interval
Jitter	in/out	long	Audio playback jitter compensation

SoundRate	out	long	Measured sound data rate
SoundFlags	in/out	long	Sound configuration settings
SoundLevel	out	long	Measured sound volume level
Volume	in/out	long	Audio playback volume
Transact	out	BSTR, BSTR	Send command, retrieve response atomically
Receive	out	BSTR	Retrieve server data
Keyflags	out	long	Key flags at time of mouse event
BackgroundInterval	in/out	long	Repeat rate for background command
BackgroundCommand	in/out	BSTR	Background command string
BackgroundReceive	out	BSTR	Retrieve background command result
MarkerPeak	out	long, long, long, BOOL, long	Compute marker peak position using parameters

Notes:

- *There are four markers, identified by a marker number. Delta markers operate in reference to marker 1.*

- *There are four traces. The TraceCommand, TraceWidth, TraceHeight, DisplayWidth, DisplayHeight, TraceType, TraceOffset, and TraceInterval properties correspond with the respective parameters of the ISOC server's /t command.*

- *The marker peak position for the selected marker is computed using two parameters: peak type and marker excursion. If the Boolean parameter is set, the marker will be moved to the computed peak position.*

Method Name	**Parameters**	**Description**
ServerConnect	BSTR, long	Connect to server at specified address and port number
ServerDisconnect		Disconnect from server
Connect	BSTR	Connect to specified instrument on server
Disconnect		Disconnect from instrument
SoundConnect	BSTR	Connect to specified audio instrument on server
Send	BSTR	Send command string
Draw	hdc,long,long,long,long, BOOL	Draw trace display into device context using specified rectangel coordinates, in color or black-and-white

Notes:

- *The Draw method is used to create a copy of the trace for printing or clipboard transfers.*

Event Name	**Description**
LButtonDown	The left mouse button was pressed within the trace area
LButtonUp	The left mouse button was released

LButtonDblClk	The left mouse button was double-clicked
RButtonDown	The right mouse button was pressed within the trace area
RButtonUp	The right mouse button was released
MouseMove	The mouse was moved
Receive	Data was received from the server via the main TCP socket
BackgroundReceive	Data was received from the server via the background UDP socket

Notes:

- *The mouse is captured when the left button is depressed, so key up and mouse movement events may be generated even when the mouse is no longer in the trace area.*

- *The ISOC instrument control generates events under the following circumstances:*
  1. *The user performs a mouse action within the visible trace area*
  2. *Data is received from the server*

3.4. Methods Reference

3.4.1. The ServerConnect Method

```
ServerConnect(BSTR bstrAddress, long lPort)
```

The `ServerConnect` method connects to an ISOCSVC server on the specified port number. The port number must be between 0 and 65535 and is usually set to 25449, the default port number used by the ISOCSVC server. Example:

```
ServerConnect("isoc.ic.gc.ca", 25449)
```

3.4.2. The ServerDisconnect Method

```
ServerDisconnect()
```

The `ServerDisconnect` method is used to terminate an existing connection to a server. It is recommended, but not necessary, to call this function prior to destroying an ISOCInstrument control. After calling this function, it is possible to call `ServerConnect` again to connect to another (or the same) ISOCSVC server.

3.4.3. The Connect Method

```
Connect(BSTR bstrID)
```

The `Connect` method is used to establish a connection to a specific instrument. If a list of instruments is required, it can be obtained by sending the `/L` ISOCSVC command (section 1.3.6) to the server using the `Transact` property.

3.4.4. The SoundConnect Method

```
SoundConnect(BSTR bstrID)
```

The `Connect` method is used to establish a connection to an audio playback instrument.

3.4.5. The Disconnect Method

```
Disconnect()
```

The `Disconnect` method is used to terminate a connection to an instrument. It is recommended, but not necessary, to call this method prior to calling `ServerDisconnect`. After a call the `Disconnect`, the `Connect` method can be called to connect to another (or the same) instrument.

3.4.6. The Send Method

```
Send(BSTR bstrData)
```

The `Send` method is used to send a command to the server. If the command produces any output, the `Receive` event will be triggered, allowing the calling program to process output asynchronously. However, it is generally not recommended to use `Send` in this case, because there is no easy way for the application program to identify which portion of the server response corresponds with which command sent. It is therefore better to use the `Transact` property.

3.4.7. The Draw Method

```
Draw(unsigned long hdc, long nLeft, long nTop, long nRight, long nBottom,
BOOL bColor)
```

The `Draw` method causes the control to draw a copy of the current trace into the device context represented by `hdc`. It is important that `hdc` represent a valid device context in the process context in which the control runs.

The trace display will be drawn with current annotation and graticule settings into the rectangle specified by `nLeft`, `nTop`, `nRight`, and `nBottom`. If `bColor` is true, the trace will be drawn with the same color as it is drawn on screen; otherwise, it will be drawn in black on a white background.

This method is intended primarily to facilitate copying the trace display to the clipboard or printing the trace display. Setting the `bColor` flag to true is appropriate if the target device context represents a bitmap. If it is a device context for a printer or a metafile device context, it is recommended to turn off the `bColor` flag, especially if the control is set to draw the trace display on a dark background, to avoid excessively dark printing.

3.5. Properties Reference

The properties of the ISOCInstrument control the object's visual appearance, audio output, and provide a means for interactive transactions with the ISOCSVC server.

3.5.1. The CursorColor Property

```
OLE_COLOR CursorColor
```

The `CursorColor` property is used to set, or read, the color of the cursor. The cursor is the thin vertical line that appears when the user clicks and drags the mouse within the trace area.

3.5.2. The GraticuleColor Property

```
OLE_COLOR GraticuleColor
```

The `GraticuleColor` property is used to set, or read, the color of horizontal and vertical graticule lines.

3.5.3. The MarkerColor Property

```
OLE_COLOR MarkerColor(long nMarker)
```

The `MarkerColor` property is used to set, or read, the color of markers. The marker affected is determined by the `nMarker` parameter, which must have a value between 0 and 3.

3.5.4. The TraceColor Property

```
OLE_COLOR TraceColor(long nTrace)
```

The `TraceColor` property is used to set, or read, the color used to display a particular trace. The trace is identified by the `nTrace` parameter, which must have a value between 0 and 2.

3.5.5. The ShowText Property

```
BOOL ShowText
```

The `ShowText` property is a read/write flag that controls whether any text annotations are shown as part of the trace display.

3.5.6. The ShowTime Property

```
BOOL ShowTime
```

The ShowTime property is a read/write flag that controls whether the time/date stamp is displayed in the upper right corner of the trace display. If enabled, the date and time displayed represents the last date and time stamp received from the server.

3.5.7. The VerticalGraticule Property

```
long VerticalGraticule
```

The `VerticalGraticule` read/write property controls the number of vertical graticules shown. The maximum value is 20; setting `VerticalGraticule` to 0 turns them off completely.

3.5.8. The HorizontalGraticule Property

```
long HorizontalGraticule
```

The `HorizontalGraticule` read/write property controls the number of horizontal graticules shown. The maximum value is 20; setting `HorizontalGraticule` to 0 turns them off completely.

3.5.9. The MarkerPos Property

```
long MarkerPos(long nMarker)
```

The `MarkerPos` property is used to set, or read, the horizontal position of the marker identified by the `nMarker` parameter. This parameter must have a value between 0 and 3. The property must have a value between 0 and the horizontal resolution of the trace display; if the marker is positioned outside the trace display, it will be turned off.

3.5.10. The MarkerTrace Property

```
long MarkerTrace(long nMarker)
```

The `MarkerTrace` property can be used to set, or read, the identifier of the trace to which the marker identified by the `nMarker` parameter is attached. The `nMarker` parameter must have a value between 0 and 3; the property must have a value between 0 and 2.

3.5.11. The MarkerLevel Property

```
long MarkerLevel(long nMarker)
```

The `MarkerLevel` read-only property can be used to determine the current level of the specified marker. This value is generated from trace values and is thus dependent on the trace resolution. If the trace resolution is lower than the instrument resolution, this value may not always agree with the marker level reported by the instrument. The marker level will be between 0 and the vertical trace resolution; the `nMarker` parameter must have a value between 0 and 3.

3.5.12. The DeltaMarker Property

```
BOOL DeltaMarker(long nMarker)
```

The `DeltaMarker` property is used to determine whether the marker specified by `nMarker` is a delta marker or a regular marker. A delta marker is displayed using a different color and symbol. The `nMarker` parameter must have a value between 0 and 3.

3.5.13. The CursorPos Property

```
long CursorPos
```

The `CursorPos` property is used to set the horizontal position of the cursor (the vertical line that is normally used to track the mouse position while the user holds down the left mouse button.)

3.5.14. The CursorTrace Property

```
long CursorTrace
```

The `CursorTrace` property is used to identify the trace to which the cursor is attached. The value of the `CursorLevel` property depends on this setting. The `CursorTrace` property can have a value between 0 and 2.

3.5.15. The SpanCursor Property

```
BOOL SpanCursor
```

The `SpanCursor` property is used to control the cursor's appearance. If `false`, the cursor is displayed as a single vertical line, with a short horizontal line segment indicating signal level at the cursor position. If `true`, the cursor is displayed as two vertical lines indicating a span centered around the middle of the trace area, and no level indicator is shown.

3.5.16. The CursorLevel Property

```
long CursorLevel
```

The `CursorLevel` read-only property is used to read the signal level at the cursor position. Because this value depends on the horizontal trace resolution, if the trace resolution is different from the instrument resolution, this value may differ from values reported by the instrument.

3.5.17. The MousePosX Property

```
long MousePosX
```

The `MousePosX` property is used to report the horizontal mouse position during mouse capture. The mouse position is reported using trace units.

3.5.18. The MousePosY Property

```
long MousePosY
```

The `MousePosY` property is used to report the horizontal mouse position during mouse capture. The mouse position is reported using trace units.

3.5.19. The TraceCommand Property

```
BSTR TraceCommand
```

The `TraceCommand` property reflects the trace command that is used to obtain traces from the instrument in the background. It corresponds with the *`command`* parameter of the /T command of ISOCSVC (section 1.3.8).

3.5.20. The TraceWidth Property

```
long TraceWidth
```

The `TraceWidth` property specifies the width of the trace on the instrument. This property corresponds with the *`width`* parameter of the /T command of ISOCSVC (section 1.3.8).

It is possible to specify a trace width of 0, in which case the actual width is calculated from the amount of data received by the instrument in response to the trace command.

3.5.21. The TraceHeight Property

```
long TraceHeight
```

The `TraceHeight` property specifies the height of the trace on the instrument. This property corresponds with the *`height`* parameter of the /T command of ISOCSVC (section 1.3.8).

3.5.22. The DisplayWidth Property

```
long DisplayWidth
```

The `DisplayWidth` property specifies the width of the trace on the display. If a display width of 200 is specified, a total of 201 data points (0-200) will be sent by the server. The server converts trace data from the instrument's resolution to the resolution specified via this property. This property corresponds with the *`packwidth`* parameter of the /T command of ISOCSVC (section 1.3.8).

3.5.23. The DisplayHeight Property

```
long DisplayHeight
```

The `DisplayHeight` property specifies the height of the trace on the display. If a trace height of 200 is specified, trace values between 0 and 200 will be sent by the server. The server converts trace data from the instrument's resolution to the resolution specified via this property. This property corresponds with the *`packheight`* parameter of the /T command of ISOCSVC (section 1.3.8).

3.5.24. The DisplayHOffset Property

```
long DisplayHOffset
```

The `DisplayHOffset` property specifies the amount by which the trace should be shifted horizontally when it is displayed. This feature can be used, for instance, to shift the display when the user is interactively controlling an instrument (e.g., changing the frequency) without the latency involved with sending commands to the instrument itself.

3.5.25. The DisplayVOffset Property

```
long DisplayVOffset
```

The `DisplayVOffset` property specifies the amount by which the trace should be shifted vertically when it is displayed. This feature can be used, for instance, to shift the display when the user is interactively controlling an instrument (e.g., changing the reference level) without the latency involved with sending commands to the instrument itself.

3.5.26. The TraceResample Property

/Ttrace,interval,offset,type,width,height,packwidth,packheight,repack,command

```
long TraceResample
```

The `TraceResample` property specifies the method used by the server when converting from the instrument resolution to the requested resolution. This property corresponds with the *`repack`* parameter of the `/T` command of ISOCSVC (section 1.3.8).

3.5.27. The TraceType Property

```
long TraceType
```

The `TraceType` property specifies the data type used by the instrument to return trace data. This property corresponds with the *`type`* parameter of the `/T` command of ISOCSVC (section 1.3.8).

3.5.28. The TraceOffset Property

```
long TraceOffset
```

The `TraceResample` property specifies the byte offset of the start of trace data within the packet returned by the instrument in response to the trace command. This property corresponds with the *`offset`* parameter of the `/T` command of ISOCSVC (section 1.3.8).

3.5.29. The TraceInterval Property

```
long TraceInterval
```

The `TraceResample` property determines how frequently trace data is requested from the instrument. Its value represents the time, measured in milliseconds, between subsequent traces. This property corresponds with the *`interval`* parameter of the `/T` command of ISOCSVC (section 1.3.8).

3.5.30. The DisplayMode Property

```
long DisplayMode(long nTrace)
```

The `DisplayMode` property specifies how a trace is displayed.

The ISOCInstrument control obtains a single trace from the instrument but can display up to three traces on screen. The `DisplayMode` property can be used to determine which of the three traces are to be displayed, and the mode of data display. In effect, this simulates the capability of most instruments to display multiple traces, trace averages, maximums, and minimums.

3.5.31. The Invert Property

```
BOOL Invert
```

The `Invert` property is used to horizontally flip the trace display.

3.5.32. The Pause Property

```
BOOL Pause
```

The `Pause` property is used to temporarily suspend the updating of the trace display. While it is set to true, traces received from the instrument are ignored.

3.5.33. The AverageInterval Property

```
long AverageInterval
```

The `AverageInterval` read-only property reflects the average time between the receipt of subsequent traces, calculated as a decaying average.

3.5.34. The Jitter Property

```
long Jitter
```

The `Jitter` property controls audio buffering. The higher the setting, the more audio data is buffered before playback begins, increasing audio latency but better protecting the application against problems caused by unreliable network connections.

Note that a minimal amount of buffering is always performed, so it is appropriate to set this property to zero when the network connection is of good quality.

3.5.35. The SoundRate Property

```
long SoundRate
```

The `SoundRate` read-only property reflects the average number of bytes per second received as part of the audio stream, calculated as a decaying average.

3.5.36. The SoundFlags Property

```
long SoundFlags
```

The `SoundFlags` property controls sound output. When written, it causes the corresponding command to be sent to the server. When read, it reflects the settings in the most recently received sound packet. The parameter is an integer value that is a combination of single-bit values, as described in section 1.2.4.

3.5.37. The SoundLevel Property

```
long SoundLevel
```

The `SoundLevel` read-only property reflects the highest amplitude level found in the most recent audio packet. It can be used to drive a visual feedback user interface component, such as a VU-meter.

3.5.38. The Volume Property

```
long Volume
```

The `Volume` property determines the sound volume. Its value is between 0 and 65535.

3.5.39. The Transact Property

```
BSTR Transact(BSTR bstrData)
```

The `Transact` property is used to perform a query-response transaction with the instrument atomically (i.e., without being interrupted by another transaction, such as a background trace command.) The property is read-only; its single parameter, `bstrData`, is the query command that is sent to the instrument. This must be a single command terminated by a question mark to ensure proper operation.

Example (HP 8594E):

```
CF = Transact("CF?")
```

3.5.40. The Receive Property

```
BSTR Receive
```

The `Receive` read-only property is used to read any data received from the server and buffered by the ISOCInstrument control. It is normally called in response to a `Receive` event.

3.5.41. The KeyFlags Property

```
long KeyFlags
```

The `KeyFlags` property can be used to determine the status of the Shift and Control keys at the time of the most recent mouse event. Its value is a combination of `MK_CONTROL`, `MK_LBUTTON`, `MK_MBUTTON`, `MK_RBUTTON`, and `MK_SHIFT`, as described in the documentation for the `WM_LBUTTONDOWN` message in the Win32 Platform SDK reference.

3.5.42. The BackgroundInterval Property

```
long BackgroundInterval
```

The `BackgroundInterval` property controls the frequency of execution of the background command specified through the `BackgroundCommand` property.

3.5.43. The BackgroundCommand Property

```
BSTR BackgroundCommand
```

The `BackgroundCommand` property can be used to specify a command that will be repeatedly sent to the instrument in the background. For instance, it can be used to regularly update marker values by querying the instrument several times a second. When the command executed in the

background generates output, the application receives a `BackgroundReceive` event and can use the `BackgroundReceive` property to read the instrument's response.

3.5.44. The BackgroundReceive Property

```
BSTR BackgroundReceive
```

The `BackgroundReceive` read-only property is used to read the instrument's response after a `BackgroundReceive` event.

3.5.45. The ScaleBase Property

```
double ScaleBase
```

The `ScaleBase` property determines the bottom value of the vertical scale, which may be displayed to the left of the trace area.

3.5.46. The ScaleStep Property

```
double ScaleStep
```

The `ScaleStep` property determines the scale step size for horizontal graticules, as it may be displayed to the left of the trace area.

3.5.47. The ScaleType Property

```
long ScaleType
```

The `ScaleType` property determines how to display a vertical scale to the left of the display area. Possible values are:

0	No scale is displayed
1	Linear scale. Scale step is added to generate each subsequent value.
2	Exponential scale. Each scale value is multiplied by the scale step to produce the next value.
3	Logarithmic (10 dB) scale.
4	Power scale (1 step corresponds with 20 dB) scale.
5	Square-power scale. Scale step is squared prior to computing each scale value.

3.5.48. The MarkerLabel Property

```
BSTR BackgroundReceive(long nMarker)
```

The `MarkerLabel` property determines what text to display next to a marker.

3.5.49. Stock Properties

The ISOCInstrument control also uses a small subset of stock properties that govern the control's visual appearance.

3.5.49.1. The BackColor Property

```
OLE_COLOR BackColor
```

The `BackColor` property is used to set the background color of the trace area.

3.5.49.2. The ForeColor Property

```
OLE_COLOR ForeColor
```

The `ForeColor` property is used to set the color of any text displayed in the trace area.

3.5.49.3. The Caption Property

```
BSTR Caption
```

The `Caption` property controls the text displayed in the upper left corner of the trace area. The text can contain newline characters.

3.6. Events Reference

The ISOCInstrument control uses several events to communicate with the controlling application asynchronously.

3.6.1. The LButtonDown Event

The `LButtonDown` event is fired every time the user clicks the left mouse button within the display area of the control. When this happens, the control also begins capturing the mouse.

3.6.2. The LButtonUp Event

The `LButtonUp` event is fired if the left mouse button is released while the mouse cursor is within the display area of the control or if the mouse is being captured by the control In response to this event, the control stops capturing the mouse.

3.6.3. The LButtonDblClk Event

The `LButtonDblClk` event occurs if the user double-clicks the mouse within the control's display area.

3.6.4. The RButtonDown Event

The `RButtonDown` event indicates that the user clicked the right mouse button within the control's display area.

3.6.5. The RButtonUp Event

The `RButtonUp` event indicates that the user released the right mouse button within the control's display area.

3.6.6. The MouseMove Event

The `MouseMove` event indicates that the user moved the mouse while the mouse was being captured by the control.

3.6.7. The Receive Event

The `Receive` event indicates that data from the instrument has arrived.

3.6.8. The BackgroundReceive Event

The `BackgroundReceive` event indicates that data has arrived in response to a query executed in the background.

Applications that connect to the ISOCSVC server or the ISOCSCAN server must respond to a simple authentication request when the connection is established. Although this authentication does not use commercial-grade encryption, it is sufficient to prevent causal attempts of server break-in. The authentication mechanism may also be used in the future to prevent incompatible versions of client and server programs from attempting to communicate with each other.

A.1 Authentication Algorithm

Although security is not a major concern for the ISOC suite, the fact that these applications utilize the TCP protocol opens the possibility that servers may be accessed by unauthorized users from either the public Internet or from other internal networks. Therefore, a simple authentication mechanism has been developed to ensure that unauthorized users do not easily gain access to the services of an ISOC server.

The authentication mechanism relies on a 2-byte key stored in the Windows Registry. When a client connects to the ISOC server, the server responds with a random 4-byte query. The query is generated using the following algorithm:

$$Q = \left[(P \oplus K \oplus E) \wedge \mathrm{AAAA_h} \cdot 2^{16} + (P \oplus K \oplus E) \wedge 5555_\mathrm{h}\right] \vee \left[E \wedge 5555_\mathrm{h} \cdot 2^{16} + E \wedge \mathrm{AAAA_h}\right]$$

Where K is the 16-bit key from the Registry (`HKLM\Software\Industry Canada\ISOC for Windows\Key`); E is a random 16-bit 'obfuscation word' obtained using the system timer; P is a randomly generated 16-bit plaintext word; and the symbol $\oplus$ denotes the exclusive-or operation. For instance, if $K=4213_h$, $E=D28E_h$, and $P=02F8_h$, Q will be $D22492CF_h$.

The plaintext word can be recovered using the following formula:

$$P = \left(\frac{Q}{2^{16}} \wedge 5555_\mathrm{h} \vee Q \wedge \mathrm{AAAA_h}\right) \oplus \left(\frac{Q}{2^{16}} \wedge \mathrm{AAAA_h} \vee Q \wedge 5555_\mathrm{h}\right) \oplus K$$

The client must decrypt this query using the correct 2-byte authentication key, and return the result to the server. If the result decodes correctly, the client is accepted; otherwise, the server closes the unauthenticated connection after printing the following error message:

```
/66:Authentication failed
```

The query is sent to the client in big-endian byte order (least significant byte first) and the client is expected to respond the same way.

The authentication mechanism is encoded in a pair of subroutines. The operation of these subroutines also requires the presence of a Registry key.

A.2 Client Authentication

```
bool ISOCAuthenticateClient(SOCKET s)
```

The `ISOCAuthenticateClient` function performs client authentication on socket s. It should be called immediately after a TCP connection to an ISOCSVC or ISOCSCAN server has been successfully established.

Note that experimentation found that a short delay (250ms) is required in order for the authentication function to perform properly. If it is called immediately after opening the socket, on certain target systems it occasionally fails.

A.3 Server Authentication

```
bool ISOCAuthenticateServer(SOCKET s)
```

The `ISOCAuthenticateServer` function should be called by ISOC servers immediately after an incoming client connection is accepted. It performs the server-side of the authentication function.

A.4 Authentication Registry Key

The success of the `ISOCAuthenticate...` functions depends on the presence of a Registry value on both the authenticating client and the server. This is a 2-byte binary value under the following key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Industry Canada\ISOC for Windows\Key
```

The two-byte value is an encryption key. If it is not present on either the client or the server, or if the values on the client and the server are not identical, authentication will fail.

Note that this value is automatically installed by the ISOC for Windows installer.

TCP sockets provide a byte-oriented stream of data. Because various network layers may packetize data differently, there are no guarantees that a single packet created, and sent, on one end of the connection will arrive as a single packet on the other end. Thus, a single call to the socket library function `send` on one end may require multiple calls to the library function `recv` on the other end.

To solve this problem, the ISOC servers use length-prefixed transactions. Every block of information is preceded by a four-byte length header.

To facilitate length-prefixed transactions, the `sendrecv.h` header file defines three helper functions. Note that these functions are defined as inline functions; only their declarations are presented here in the text.

B.1 The recvwithlength Function

```
inline int recvwithlength(SOCKET s, char FAR* buf, int len, int flags);
```

The `recvwithlength` function receives a length-prefixed block of information on socket `s`, filling buffer `buf` with up to `len` characters. The `flags` parameter contains flag values that are defined for the `recv` function in the WinSock documentation.

Only the actual data (without the length prefix) is returned in the buffer pointed to by `buf`. If `len` is less than the number of bytes indicated in the prefix of the received packet, the remaining data is discarded.

Note that depending on network traffic, packet fragmentation, and packet size, one call to `recvwithlength` may result in multiple calls to the WinSock `recv` function.

B.2 The sendwithlength Function

```
inline int sendwithlength(SOCKET s, const char *buf, int len,
                          int flags, const struct sockaddr FAR *to = NULL,
                          int tolen = 0);
```

The `sendwithlength` function creates, and sends, a length-prefixed packet on socket `s`. The data that is to be sent is stored in the buffer pointed to by `buf`, and the number of bytes to be sent is specified in `len`. The `flags` parameter may contain values as defined for the WinSock `send` function. The `to` and `tolen` parameters also function identically to the similarly named parameters of the WinSock `send` function, and allow sends on a connectionless socket, for instance.

B.3 The sendstr function

```
inline int sendstr(SOCKET sockfd, const char *p,
                   struct sockaddr *pAddr = NULL, int nAddrLen = 0);
```

The `sendstr` function provides a convenient shorthand for sending zero-terminated strings as length-prefixed data. The function automatically calculates the length of the string using the `strlen` library function.

Because of the inefficiency of `strlen`, if the length of the string is known to the calling function and performance is an issue, it is advantageous to use the `sendwithlength` function to avoid an unnecessary recalculation of string length.