Integrated Spectrum Management System

ISOC for Windows

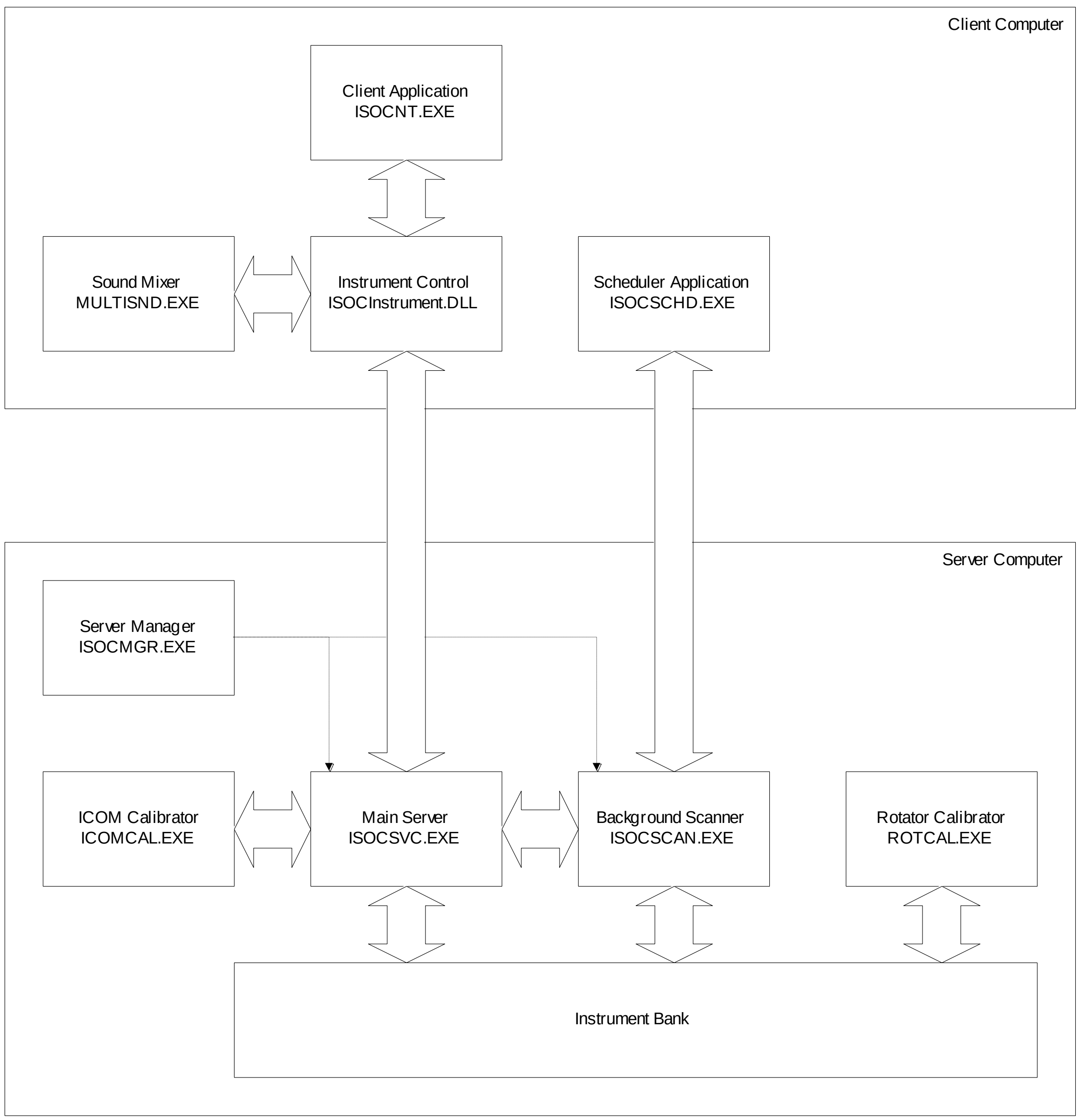
Developers' Roadmap

Last Updated: 3/27/2014 10:07:00 AM

1.1. Overview

The Integrated Spectrum Observation Center (ISOC) is an application suite of several client and server programs and auxiliary components. The purpose of this application suite is to provide flexible remote access to instrument suites using standard network protocols.

The following figure shows an overview of the main ISOC components and their relationships to each other.



The instrument bank consists of a selection of radios, spectrum analyzers, and auxiliary instrumentation that is connected to a controlling computer via a variety of interfaces: GP-IB, RS-232, ICOM's CI-V. A special type of instrument is the PC audio port, used to capture, and digitize, up to two monaural audio channels and stream audio to client computers.

Instruments are controlled by the main ISOC server, ISOCSVC.EXE. In addition to providing an instrument interface, ISOCSVC.EXE also arbitrates access by providing a mechanism through which instruments are reserved for use and later released.

Another key server component is ISOCSCAN.EXE, which performs background scanning functions. Background scanning provides the ability to perform scheduled measurements in an unattended fashion and store the results in data files that are made available for download.

Three auxiliary support applications perform ICOM radio calibration, calibration of the antenna rotator instrument, and server management.

On the client side, all interactive communication with instruments is performed via the ISOCInstrument control. This COM control component provides an interactive interface for sending and receiving data, the ability to display a visual graph (instrument 'trace') and the ability to play back an audio stream. This latter capability utilizes an external component, MULTISND.EXE, which is a simple COM server performing digital audio mixing, in effect permitting the user to listen to multiple audio sources simultaneously.

Yet another client application, ISOCSCHD.EXE, communicates with the background scanner server. This application lets the user schedule and manage background scanner sessions.

Note: This is an evolving document that will continue to be updated during the project's life cycle.

Executables:

`DFREG.EXE`	Direction finder Registry setup
`ICOMCAL.EXE`	Command-line calibrator program for ICOM receivers
`ICOMDUMP.EXE`	Calibration result dump utility for ICOM receivers
`IPSWEMU.EXE`	IP switch matrix emulator
`ISOCCONF.EXE`	ISOC server configurator utility
`ISOCCRON.EXE`	User interface to ISOC background scanner
`ISOCDF.EXE`	Interactive ISOC DF application
`ISOCGPS.EXE`	GPS monitor
`ISOCMGR.EXE`	ISOC service application manager
`ISOCNT.EXE`	Interactive ISOC application
`ISOCSCAN.EXE`	ISOC background scanner service
`ISOCSVC.EXE`	Main ISOC instrument control server
`ROTCAL.EXE`	Antenna rotator calibrator

Dynamic-link libraries:

`ANTCRON.DLL`	Scanner setup dialog for antenna devices used by: `ISOCCRON.EXE`
`ANTSCAN.DLL`	Scanner module for antenna devices used by: `ISOCSCAN.EXE`
`CIVSVC.DLL`	CI-V communications library used by: `ISOCSVC.EXE`, `ISOCCONF.EXE`
`CRCSE.DLL`	CRC Spectrum Explorer interactive support used by: `ISOCNT.EXE`
`DC44X.DLL`	DC44X tone decoder interactive support used by: `ISOCNT.EXE`
`DC4CRON.DLL`	Scanner setup dialog for DC-4XX tone decoders used by: `ISOCCRON.EXE`
`DC4SCAN.DLL`	Scanner module for DC-44X tone decoders used by: `ISOCSCAN.EXE`
`DF.DLL`	DF user interface components used by: `ISOCNT.EXE`
`DF7KSVC.DLL`	Doppler DDF 7001 binary protocol driver used by: `ISOCSVC.EXE`
`DFCRON.DLL`	Scanner setup dialog for DF used by: `ISOCCRON.EXE`
`DFLIB.DLL`	DF-specific functions library used by: `ISOCLIB.DLL`
`DFSCAN.DLL`	Scanner module for DF

	used by: `ISOCSCAN.EXE`
`DFSVC.DLL`	DF communications library used by: `ISOCSVC.EXE`, `ISOCCONF.EXE`
`ESMB.DLL`	ESMB receiver interactive support used by: `ISOCNT.EXE`
`ESMBCRON.DLL`	Scanner setup dialog for ESMB receivers used by: `ISOCCRON.EXE`
`ESMBSCAN.DLL`	Scanner module for ESN receivers used by: `ISOCSCAN.EXE`
`ESNCRON.DLL`	Scanner setup dialog for ESN receivers used by: `ISOCCRON.EXE`
`ESNSCAN.DLL`	Scanner module for ESN receivers used by: `ISOCSCAN.EXE`
`FLEX.DLL`	FLEX tone decoder interactive support used by: `ISOCNT.EXE`
`GPIBSVC.DLL`	GPIB communications library used by: `ISOCSVC.EXE`, `ISOCCONF.EXE`
`GPSSCAN.DLL`	Scanner module for GPS used by: `ISOCSCAN.EXE`
`ICOM.DLL`	ICOM receiver interactive support used by: `ISOCNT.EXE`
`ICOMCRON.DLL`	Scanner setup dialog for ICOM receivers used by: `ISOCCRON.EXE`
`ICOMSCAN.DLL`	Scanner module for ICOM receivers used by: `ISOCSCAN.EXE`
`IFRCRON.DLL`	Scanner setup dialog for IFR receivers used by: `ISOCCRON.EXE`
`IFRSCAN.DLL`	Scanner module for IFR receivers used by: `ISOCSCAN.EXE`
`ISOCInstrument.DLL`	ISOC virtual instrument control used by: `ISOCNT.EXE`
`ISOCLIB.DLL`	ISOC common functions library used by: all ISOC components
`ISOCUI.DLL`	ISOC user interface common components used by: `ISOCNT.EXE`
`OARSCAN.DLL`	Scanner module for OAR receivers used by: `ISOCSCAN.EXE`
`ROTCRON.DLL`	Scanner setup dialog for antenna rotators used by: `ISOCCRON.EXE`
`ROTSCAN.DLL`	Scanner module for antenna rotators used by: `ISOCSCAN.EXE`
`RS232SVC.DLL`	RS-232 communications library used by: `ISOCSVC.EXE`, `ISOCCONF.EXE`

`RSFSP.DLL`	Rohde & Schwarz FSP receiver support used by: `ISOCNT.EXE`
`RSIBSVC.DLL`	RSIB communications library used by: `ISOCSVC.EXE`, `ISOCCONF.EXE`
`RSSCAN.DLL`	Scanner module for R&S DF equipment used by: `ISOCSCAN.EXE`
`SECRON.DLL`	Scanner setup dialog for Spectrum Explorer used by: `ISOCCRON.EXE`
`SESCAN.DLL`	Scanner module for Spectrum Explorer used by: `ISOCSCAN.EXE`
`SMHCRON.DLL`	Scanner setup dialog for SMH signal generators used by: `ISOCCRON.EXE`
`SMHSCAN.DLL`	Scanner module for SMH signal generators used by: `ISOCSCAN.EXE`
`SNDCRON.DLL`	Scanner setup dialog for audio recording used by: `ISOCCRON.EXE`
`SNDSCAN.DLL`	Scanner module for background sound recording used by: `ISOCSCAN.EXE`
`SOUNDSVC.DLL`	Digital sound communications library used by: `ISOCSVC.EXE`, `ISOCCONF.EXE`
`TBTCK.DLL`	Telonic/Berkeley TCK filter support library used by: `ISOCNT.EXE`
`TCKCRON.DLL`	Scanner setup dialog for Telonic/Berkeley TCK filter used by: `ISOCCRON.EXE`
`TCKSCAN.DLL`	Scanner module for Telonic/Berkeley TCK filter used by: `ISOCSCAN.EXE`
`TCPIPSVC.DLL`	TCP/IP communications library used by: `ISOCSVC.EXE`, `ISOCCONF.EXE`

<u>External components:</u>

`KNOBCtrl.dll`	ActiveX "knob" control
`Lame_enc.dll`	The Lame MP3 encoder library (open source, but uses technology that is patented in some countries)
`METER.dll`	ActiveX "meter" control
`Rsib32.dll`	Rohde & Schwarz RSIB support library
`SCHEDULE.dll`	ActiveX "weekly grid" control
`MULTISND.EXE`	ActiveX sound mixer server
`MULTIWnd.dll`	Multiple window manager library
`SEInterface.dll`	Visual Basic interface code for Spectrum Explorers
`zlib.dll`	Open-source compression library

3.1. ISOCSVC.EXE

3.1.1. Overview

The primary component of the ISOC system on the server side is ISOCSVC.EXE. This server listens for incoming TCP connections and provides a simple, interactive interface for communicating with instruments. Its role is twofold. First, it acts as a 'protocol translator', for lack of a better term, allowing software to connect to instruments with a variety of physical interfaces via a TCP/IP network connection. Second, it acts as an arbitrator, managing access to instruments from multiple clients.

There is one thing ISOCSVC.EXE doesn't do: it doesn't provide a virtualized instrument interface. In other words, it doesn't provide services such as duplicating instrument state information or providing a generic 'superset' of features for instruments such as spectrum analyzers and receivers. Client programs are required to use the raw command set that the instrument manufacturer provides for programmatic instrument control. This approach made it possible for the server to be relatively small, robust, and reliable, without significantly (if at all) increasing client complexity.

That said, the server does provide more than a mere bi-directional interface for instrument control. It provides a means to schedule commands to be executed in the background (e.g., repetitive level measurements or trace readouts), and it provides a means to collect trace data and compress it for optimized transmission. In particular, the server utilizes the connectionless UDP protocol to continuously transmit stream data, such as trace updates, recurring background command execution results, and audio.

Conceptually, the ISOCSVC.EXE server operates as follows. Upon startup, it listens for incoming connections on a predefined TCP port. When a connection is established, a separate thread is spawned which will manage all aspects of that connection.

This separate thread listens for incoming commands on the TCP socket. The thread operates in one of two states: connected and not connected. When connected, the thread is associated with a specific instrument to which commands are forwarded. Instruments may be managed in synchronous mode (GP-IB instruments) or in asynchronous mode (RS-232 and CI-V instruments.) The difference is that asynchronous instruments may send data at any time, whereas synchronous instruments only send data when it is so requested by a specific command.

When the server receives a command from a remote client over the TCP socket, it first checks for the presence of the slash character ('/') as the first character of the command. If the slash character is present, the command is assumed to be a command for the server itself; if there is no slash character, the command is forwarded to the instrument if the server is in the connected state.

Although this capability alone would be sufficient to fully control an instrument, there are problems with performance. The overhead associated with sending commands over a TCP socket becomes pronounced over slower networks, especially if either the client, or the server, (or both) are connected via an ordinary modem line. Because of this, the server provides a background command execution capability. Simply put, a client has the ability to request the repetitive execution of a command in the background; results are communicated to the client using an alternate channel of communication, for performance reasons as well as to ensure that the results of foreground and background commands can be easily distinguished.

The channel of communication for background command results is a connectionless UDP socket. If a client wishes to receive data via UDP, it communicates the port number to the server via a special command. Subsequently, the server will use this port number to send background results. The main advantage of using UDP is that this connectionless protocol does not suffer from ever increasing timeouts like a TCP socket if the underlying IP layer is unreliable (high packet loss.) It must be kept in mind, however, that the delivery of UDP packets is not guaranteed by the network; therefore, the background command execution capability of the server is best used for repetitive measurements where it's not a major problem if occasionally, measurement results are lost.

The background UDP channel is used for more than communicating command results to the client. The server has a special feature that allows it to obtain an 'instrument trace' in the background, optionally resample the trace, and send the result to the client. The 'instrument trace' is a binary representation of the graphical display found on spectrum analyzers and some receivers. The server provides a generic capability to extract the trace from the instrument's response, resample it to a specific horizontal and vertical resolution, and transmit the result as a UDP packet. This mechanism allows fast, response trace updates even via slow communication lines.

Lastly, the background UDP channel is also used for transmitting streaming audio data.

3.1.2. Modular Construction

The ISOCSVC.EXE server supports a modular construction. Port driver modules exist as separate DLLs that can be compiled and distributed independently from the main server.

DLLs that the server must load at startup time are specified in a semicolon-separated list that is stored in the Registry: `HKLM\Software\Industry Canada\ISOC for Windows\ServerDLLs`. The default set of DLLs with which the package is distributed is `"GPIBSVC.DLL;SOUNDSVC.DLL;TCPIPSVC.DLL;RSIBSVC.DLL;DFSVC.DLL"`.

DLLs must be created as "MFC Extension" DLLs using Visual C++. A typical server DLL implements a class derived from CPort, which provides the functionality for a specific port type (e.g., GP-IB). DLLs are expected to export the following set of functions (all functions are declared with `extern "C"` and `__declspec(dllexport)` to ensure proper linkage):

`PCPORT Create(int nIF, int nType)`

This function is called with nIF set to the interface type and nType set to the instrument type. Interface types are enumerated in ISOCLIB.H (CISOCDev::IF). Modules are free to define additional interface type values as needed; however, care must be taken to ensure that no two modules reuse the same interface type value. Instrument type values are not predefined but loaded from the Registry. The function CISOCDev::FindType() can be used to obtain the numeric value that corresponds with a specific instrument type identifier string.

`void Startup()`

Called to allow the module to initialize its operations. For instance, the CI-V module uses this function to start its listening threads for RS-232 ports on which CI-V interfaces are present.

`void Cleanup()`

Called to allow the module to clean up before program termination.

`void AddIcon(CImageList *pImageList)`

Called to allow the module to add icons and associate them with specific instrument types. These icons will be used in ISOCCONF.EXE when the instruments are listed.

`bool OnInput(int nType)`

Called by ISOCCONF.EXE when the user clicks the Inputs button. If a custom Inputs dialog is shown, the function should return true. If this function returns false, the default Inputs dialog is displayed by the program.

`const char *GetIFName(int nType)`

Should return a language-independent string representing the interface name. E.g., "RS-232".

`bool Save()`

Allows the module to save any extra configuration information in the Registry when ISOCCONF.EXE terminates. (The function isn't called if the user chooses not to save any changes.) Should return true on success.

`CDialog *SetupDlg(int nDev, int nIF, unsigned char *pSettings)`

Creates the subdialog that is displayed with the instrument setup dialog by ISOCCONF.EXE. The subdialog should provide fields specific to the interface that is implemented by the module. For instance, the RS-232 module subdialog provides fields for the bit rate, parity, and stop bit settings of the serial interface.

3.1.3. Switch Matrix Support

As mentioned previously, the ISOCSVC.EXE server does not provide a virtualized instrument model. A specific exception was made for switch matrices such as the "Racal Switch Matrix" instrument.

The reason for this exception is simple. Whereas other instruments are accessed by clients in exclusive mode (i.e., only one client can access the instrument at any given time) the switch matrix is a shared resource. The same switch matrix is used for connect the inputs and outputs of multiple instruments simultaneously. In order to prevent situations where multiple clients may send conflicting commands to the switch matrix (which, apart from being annoying, may also cause equipment failure) the switch matrix is, in fact, virtualized, and a special command set is provided for client programs to access switch matrix resources specific to the instrument to which they are connected.

3.1.4. Power Bar Support

Similarly to the case of the switch matrix, the remote control power bar is also a shared resource. For this reason, power bar support is also implemented on the server level, so rather than giving any client exclusive access to the power bar itself, the server provides instrument-level functionality for instruments that can be powered down remotely.

3.1.5. Usage

Detailed description of the ISOCSVC.EXE command set can be found in the ISOC for Windows Application Programming Interfaces manual. Here is a brief overview of a typical session between a client program and ISOCSVC.EXE:

1. The client establishes a TCP connection to the designated port on the server, and performs authentication

2. The client uses the /L command to acquire a list of available instruments

3. The client uses the /C command to connect to a specific instrument

4. The client sends instrument-specific commands and reads responses

5. Optionally, other server commands (escaped with a leading slash, '/') are used to change operating parameters, set up background commands, etc.

6. The client disconnects from the instrument using the /D command and optionally, closes the socket using /X.

3.1.6. Software Operation

The ISOCSVC.EXE program is based on a Windows NT Service sample application from Microsoft.

The main() function is implemented in a Microsoft file, SERVICE.C. Depending on the way the application was started, main() invokes the function ServiceStart() in MAIN.CPP. This is where the real work begins.

The ServiceStart() function, after completing some trivial initializations, enters an infinite loop in which it waits for incoming connections on a TCP socket using a call to the accept() socket library function. When an incoming connection is established, ServiceStart() creates a new CTCPThread object and spawns a new thread, using the (static) function CTCPThread::ThreadProc as the thread function.

The CTCPThread class (declared in SOCKET.H, implemented in SOCKET.CPP) implements the high-level command functionality for ISOCSVC.EXE. At its heart is the CTCPThread::ReadTCP() member function that reads, and processes, commands received from the client.

Commands fall into one of three categories:

- Server commands are escaped with a leading forward slash ('/') character
- Instrument commands are terminated with a semicolon (';') character. When such a command is sent to the instrument, the instrument is not expected to respond
- Queries are terminated with the question mark ('?') character. After a query, an attempt is made to read a response from the instrument and send it back to the client.

This synchronous command model is compatible with the operation of instruments connected via the GP-IB (HP-IB) interface. With other interfaces, certain adjustments were necessary to fit the model with the instruments' operation, but to date, the model was more than adequate to carry out full instrument commanding.

3.1.6.1. Server Command Processing

When a new connection to the server is established, the server creates a CTCPThread object and starts a new thread. This new thread will run CTCPThread::ReadTCP(), a function that reads commands from the TCP socket.

When a command is received, it is first determined if it is escaped by a leading forward slash character, in which case it is processed by the server itself. Server commands are processed by the CTCPThread::ProcessCommand() function.

In addition to the primary thread, a secondary thread is started when a connection is requested to an instrument. The controlling function for this thread is CUDPThread::ReadUDP(). This is a pure virtual function; specific implementations exist in classes derived from CUDPThread. In CInstrThread, ReadUDP performs the processing of instrument traces and background commands. In CSoundThread, ReadUDP reads and processes digitized sound data.

The classes CTCPThread and CUDPThread, and classes derived from these, are defined and implemented in SOCKET.H and SOCKET.CPP.

3.1.6.2. Instrument Commanding

Any command received by CTCPThread::ReadTCP that is not escaped with a leading forward slash is assumed to be a command destined for the instrument that this session is associated with.

When such a command is received, it is parsed by a simplified parser that checks for the presence of terminator characters: semicolons and question marks. Individual commands are extracted and processed one by one. According to conventions borrowed from GP-IB, a command is assumed to be a query if it ends with a question mark.

Commands and queries are executed using member functions of the class CPort. This class provides a generalized representation of a communication port that connects the server to an instrument. The CPort class supports RS-232 and GP-IB connections, as well as instruments connected via the CI-V protocol. The latter require a helper class, CCIVPort (despite the name, it is not derived from CPort), in order to arbitrate between multiple CI-V instruments connected via a single RS-232 port.

Additional support classes include CPowerPort and CSoundPort, which provide functionality specific to the remote control power bar and the Windows sound device.

These classes are defined in PORT.H, CIV.H, POWER.H, SOUND.H, and implemented in PORT.CPP, CIV.CPP, POWER.CPP, and SOUND.CPP, respectively.

The core functions in the CPort class are Connect(), Disconnect(), Transact(), and MultiTransact().

3.1.6.3. Switch Matrix Commanding

Switch matrix commanding is accomplished through the CMatrix class (defined and implemented in MATRIX.H and MATRIX.CPP). A single object of type CMatrix is created upon server startup; the constructor for this class is responsible for loading switch matrix configuration information from the Registry. The server communicates with this object using the Open and Close member functions which, in turn, utilize the Command function to send commands to the matrix.

When a client sends a matrix command to the server, the command is parsed by CTCPThread::DoMatrix(), which calls member functions of CMatrix to carry out the command. An interesting example is the implementation of the /M? (matrix present?) server command: it sends a single space character to the switch matrix, checking for a GP-IB error. This one-character command does nothing; however, if no matrix is present, the command fails, thus providing a reliable (and fast) method for detecting the presence of the matrix.

3.1.6.4. Audio Capture

The ISOCSVC.EXE server can capture digital audio from a standard Windows multimedia audio device. Two channels of audio can be captured at 8000 8-bit samples per second. Any client can request one or both channels of audio; multiple clients can receive audio streams at the same time.

In order to facilitate the transmission of digitized audio over slow communication lines, two methods of compression are used. One method simply halves the sampling rate; the other uses a public domain GSM compression library for high-efficiency compression. When both compression methods are used, audio of (barely) acceptable quality can be transmitted over a PPP connection as slow as 9600 bits per second.

The audio capture device is distinct from the audio input device; this device type is essentially a dummy device used to allow the user to select audio input sources via the switch matrix. Whereas the audio capture device is used in non-exclusive mode, audio input devices can be opened by only one client at any given time.

In addition to sending audio to a remote client, the server is also capable of recording audio in a .WAV file, using an annotated format that is compatible with the shareware application RecAll PRO.

3.2. The ISOCLIB.DLL support library

All ISOC components, including the ISOCSVC.EXE program, make extensive use of the ISOCLIB.DLL library. This library provides support functions in three areas: instrument and matrix configuration data structures, helper functions, and units conversion.

3.2.1. Data Structures

The ISOCLIB.DLL library relies on the Standard Template Library (STL) to implement several related lists for switch matrix operation.

CISOCDevList is a list of CISOCDev objects, each of them representing a virtual instrument on the server.

CISOCMatList is a list of CISOCMat objects. Each object represents a signal source that can be connected up via the switch matrix.

CISOCInpList is a list of CISOCInp objects. A CISOCInp object represents a signal source for a specific instrument and the associated switch matrix command. Each virtual instrument has an associated CISOCInpList object containing the list of valid inputs for that instrument.

CISOCConList is a list of CISOCCon objects. A CISOCCon object is a connector; this is used with instruments such as the ICOM R-9000 receiver that has multiple signal source connectors on the back panel. Each CISOCCon object contains the necessary switch matrix commands to connect a signal to the respective connector. Each instrument as an associated CISOCConList that represents the instrument's list of back-panel connectors.

In addition to these structures, the ISOCLIB.DLL library also defines the following structures:

- PACKEDDATETIME is a "packed" date/time stamp format used when transmitting trace data blocks from the server to the client;
- TRACEHDR is the header block for trace data.

Lastly, the ICOMCalibration class is used to create ICOM receiver calibration data blocks and save these to the Registry.

3.2.2. Helper Functions

The ISOCLIB.DLL also provides several helper functions that are used throughout the ISOC suite. These functions can be loosely bundled into several categories.

3.2.2.1. Authentication Functions

Two complementary functions are provided to facilitate client-server authentication.

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

3.2.2.2. SYSTEMTIME Operators

This group consists of several comparison and arithmetic operators for the SYSTEMTIME type. These operators simplify date/time arithmetic.

```
bool operator<(const SYSTEMTIME &st1, const SYSTEMTIME &st2)
bool operator>(const SYSTEMTIME &st1, const SYSTEMTIME &st2)
bool operator<=(const SYSTEMTIME &st1, const SYSTEMTIME &st2)
bool operator>=(const SYSTEMTIME &st1, const SYSTEMTIME &st2)
bool operator==(const SYSTEMTIME &st1, const SYSTEMTIME &st2)
__int64 operator-(const SYSTEMTIME &st1, const SYSTEMTIME &st2)
const SYSTEMTIME operator+(const SYSTEMTIME &st, int n)
const SYSTEMTIME &operator+=(SYSTEMTIME &st, int n)
const SYSTEMTIME operator-(const SYSTEMTIME &st, int n)
const SYSTEMTIME &operator-=(SYSTEMTIME &st, int n)
```

3.2.2.3. Data Conversion

Data conversion functions assist in the conversion between standard Windows and ISOC-specific data types.

```
bool SystemTimeToPackedTime(SYSTEMTIME *pst, PACKEDDATETIME *ppdt)
bool PackedTimeToSystemTime(PACKEDDATETIME *ppdt, SYSTEMTIME *pst)
```

3.2.2.4. Data Communication

Data communication functions are utilized throughout the ISOC suite. Their main purpose is to facilitate length-prefixed data transmissions on a communications socket.

```
int sendsz(SOCKET s, const char *pszText, int flags = 0)
int sendprintf(SOCKET s, char *pszFormat, ...)
int sendwithlength(SOCKET s, const char *buf, unsigned short len, int flags,
   const struct sockaddr FAR *to, int tolen)
int sendstr(SOCKET sockfd, const char *p, struct sockaddr *pAddr, int
   nAddrLen)
int recvlength(SOCKET s, int flags)
int recvlendata(SOCKET s, char FAR *buf, int len, int flags)
```

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

3.2.2.5. Registry Functions

Registry functions provide some additional functionality for moving, copying, and deleting registry key subtrees.

```
LONG RegWipeKey(HKEY hKey, LPCTSTR lpszSubKey)
LONG RegCopyKey(HKEY hKey, LPCTSTR lpszSubKey, HKEY hNewKey, LPCTSTR
   lpszName)
LONG RegMoveKey(HKEY hKey, LPCTSTR lpszSubKey, HKEY hNewKey, LPCTSTR
   lpszName)
```

3.2.2.6. Miscellaneous Helper Functions

```
bool IsWindows95()
char hex2char(int n)
int char2hex(char c)
bool GetWeekBase(SYSTEMTIME *st)
bool IsValidDate(SYSTEMTIME &st)
void DrawButton(HDC dc, RECT rect, int nType, UINT itemState)
```

3.2.3. Unit Conversion

Six functions are provided for easy parsing and formatting of frequency, level, and time values:

```
double ParseFrq(const char * pszFrq)
double ParseLvl(const char * pszLvl, int nAUnit, double fZ)
double ParseTime(const char * pszTime)
FormatFrq(char * pszFrq, double fVal, FRQ_UNIT nUnit, int nDec)
FormatLvl(char * pszLvl, double fVal, LVL_UNIT nUnit, int nDec)
FormatTime(char * pszTime, double fVal, TIME_UNIT nUnit)
```

Permissible unit values are declared in UNITS.H. The ParseLvl function takes an extra parameter (fZ) that represents the input impedance of the device for which the conversion is performed; without this value, it would not be possible to convert, for instance, between dBμV and dBm.

3.3. ISOCSCAN.EXE

3.3.1. Overview

The second main server component after ISOCSVC.EXE is ISOCSCAN.EXE, the background scanning service application.

The purpose of ISOCSCAN.EXE is quite simple: maintain a set of schedule entries, activate instruments at scheduled times, and collect scan data. The implementation, however, differs depending on the type of instrument in use.

Presently (ISOCSCAN.EXE is about to undergo a revision) the background scanner supports scanning on two types of instruments: the ESN receiver and the ICOM receiver. In addition, ISOCSCAN.EXE also supports scheduled recording of audio.

3.3.2. Modular Construction

Like ISOCSVC.EXE, the ISOCSCAN.EXE server is now also modularized. Instrument-specific scanner implementations live in separate DLLs which are loaded by the main server at startup time. These DLLs can be compiled and distributed separately from the main application suite.

DLLs that the server must load at startup time are specified in a semicolon-separated list that is stored in the Registry:
`HKLM\Software\Industry Canada\ISOC for Windows\ScannerDLLs`. The default set of DLLs with which the package is distributed is

```
"ANTSCAN.DLL;DC4SCAN.DLL; ESMBSCAN.DLL;ESNSCAN.DLL;ICOMSCAN.DLL;\
 IFRSCAN.DLL;SMHSCAN.DLL;SNDSCAN.DLL;SESCAN.DLL;ROTSCAN.DLL;\
 TCKSCAN.DLL;DFSCAN.DLL;GPSSCAN.DLL;OARSCAN.DLL;RSSCAN.DLL".
```

DLLs must be created as "MFC Extension" DLLs using Visual C++. A typical server DLL implements a class derived from CPort, which provides the functionality for a specific port type (e.g., GP-IB). DLLs are expected to export the following set of functions (all functions are declared with `extern "C"` and `__declspec(dllexport)` to ensure proper linkage):

```
class CScan *Create(class CScanSession *pThis,
     CISOCDevList::iterator iInstrument, const char *pszESN)
```

This function is called to create a CScan-derived object that will perform the scanning.

```
void Validate(CISOCDevList *pList)
```

This function is called when the server starts. It allows the module to validate the presence of any instruments that the server will recognize. For instance, the ESN scanner module's Validate function removes any ESN receivers that share the GP-IB bus with other instruments, as such receivers cannot be used for background scanning (which, in case of the ESN, requires exclusive access to the GP-IB bus.)

```
bool Support(int nType)
```

Returns true if the instrument type is supported. The numeric value is not predefined, but determined at runtime when instrument types are loaded from the Registry. The string representation of the instrument type can be retrieved using CISOCDev::FindType().

```
const char *HeaderString(int nType, const char *pszINI)
```

Returns a string header that is stored in the ESN result file. The ESN file format is a format predefined by Industry Canada.

3.3.3. Usage

Detailed description of the ISOCSCAN.EXE command set can be found in the ISOC for Windows Application Programming Interfaces manual. Here is a brief overview of a typical session between a client program and ISOCSCAN.EXE:

1. The client establishes a TCP connection to the designated port on the server, and performs authentication

2. The client obtains a list of schedule entries

3. The clients sends updated schedule entries to the server

4. The client monitors the progress of a scan by sending the appropriate queries

5. The client terminates the connection

3.3.4. Software Operation

The ISOCSCAN server has two distinct functions:

1. Executing background scans at scheduled times

2. Servicing client connections

The ISOCSCAN application is based on the generic Microsoft Windows NT service application example. Its main starting function is ServiceStart(), found in MAIN.CPP. This function performs the necessary initializations and then enters an infinite loop, awaiting incoming client connections. When a client connection is established, it is serviced by CScanSession::Interact(), a function that is called from within a separate thread of execution in order to free up the main thread for processing other incoming connections.

Background scanning functionality is encapsulated within the CScanSession class. Each CScanSession object corresponds with a background task. Background execution is initiated by starting a secondary thread with the CScanSession::Scheduler function. This function waits until the time of the next scheduled background task, and then initiates the background task by creating yet another thread with the CScanSession::Scan member function.

Whereas CScanSession objects represent all scheduled sessions, CScan objects are used to carry out the actual scanning task. The CScan class provides generic functionality for parsing input files, building frequency lists, and connecting to the ISOCSVC.EXE server for reserving, and communicating with actual instruments. Classes derived from CScan contain implementations specific to one instrument type or another. Since significant differences exist in the way scanning is carried out on different instrument types, these are discussed separately below.

3.3.4.1. ESN Scanning

The ESN receiver has a special mode of operation where it scans a preloaded list of frequencies at the maximum speed the radio hardware permits. Results are communicated via the GP-IB bus with the ESN receiver acting as bus controller. This mode of operation requires that the ESN receiver be the only instrument on the bus; for this reason, many ISOC server installations use two GPIB cards on the receiver, one of which is dedicated to the ESN receiver.

Because of this special mode of operation, ESN scanning is not performed by ISOCSVC.EXE. A connection is made to the ISOCSVC server, but it is only to reserve the instrument, in order to ensure that no other user attempts to access it while a scan is being performed. The background scanner server, ISOCSCAN.EXE, communicates with the ESN receiver via the GPIB bus on its own.

ESN scanning is carried out by the CESNScan::Scan() function. This function sets up the receiver by transmitting the frequency list and other settings, initiates scan operation, and then passes GPIB bus control to the receiver itself. The function then enters a loop in which it reads data from the receiver and saves it to the results file.

3.3.4.2. ICOM Scanning

Background scanning with the ICOM receiver differs from ESN scanning in two important ways. First, scanning is carried out with the server computer in control; scan frequencies and level readout commands are sent interactively. Second, the scanner server doesn't communicate with the instrument directly; all communication takes place using the ISOCSVC.EXE instrument server.

A problem specific to ICOM receivers is the unpredictable settling time of the receiver during fast scanning. The solution to this problem is a "hack" in the ISOCSVC.EXE server itself, in its CCIVPort::ProcessIC() member function. A simple algorithm checks for any large level fluctuations and if such fluctuations are detected, the level readout is repeated. If necessary, multiple level readouts are attempted, although the acceptable fluctuation is increased with every readout, in order to ensure that a successful readout is obtained within a reasonable time.

3.3.4.3. Audio Recording

Audio recording is carried out entirely by the ISOCSVC.EXE server. The role of the background scanning server is reduced to merely sending the appropriate commands to ISOCSVC.EXE to initiate and/or stop an audio recording session. Correspondingly, the function CSoundScan::Scan() is essentially a placeholder that merely waits for the scan to end.

3.4. Auxiliary Programs

In addition to the two main service applications, ISOCSVC.EXE and ISOCSCAN.EXE, several auxiliary programs are provided to manage an ISOC server.

3.4.1. The ISOC Server Configurator

The ISOC Server Configurator, ISOCCONF.EXE, provides a graphical user interface for setting up instruments. Its user interface elements correspond closely with data structures defined in the ISOCLIB.DLL library. It also loads and accesses server DLLs, which provide the implementations for port-specific subdialogs that are displayed when an instrument is configured.

The main dialog ("ISOC Server Configurator") contains a list of all instruments on the server. This list corresponds with the CISOCDevList class in ISOCLIB.DLL. Details for each instrument can be viewed using the Edit button; these details correspond with the data content of CISOCDev objects.

The Inputs button invokes the ISOC Signal Sources dialog. The list herein is the list of all available signal sources on the server's switch matrix, represented in ISOCLIB.DLL by the CISOCMatList class. Details of individual signal sources can be viewed by clicking Edit; the dialog that appears shows data fields that correspond with the data content of CISOCMat objects.

For each instrument displayed in the Instrument Configuration dialog, you can click the Inputs button that shows the matrix commands used for connecting a specific signal source to this instrument. This list corresponds with the CISOCInpList object associated with the instrument in ISOCLIB.DLL.

Lastly, for some instruments (notably ICOM receivers) a similar list exists that defines the instrument's input connectors. This list can be viewed by clicking the Connectors button, and it corresponds with the CISOCConList object associated with the instrument.

The ISOC Server Configurator also allows the configuration of site-specific parameters, such as the site's name, logging preferences, or the GP-IB parameters of the switch matrix. All information is saved in the Windows Registry. No attempt is made to cause running server components (namely, ISOCSVC.EXE and ISOCSCAN.EXE) to re-read the Registry; to effect a re-read, these server components must be restarted.

3.4.2. ICOM Calibrator

The ICOM calibrator is a 32-bit command-line utility that carries out a calibration sequence on ICOM receivers. Instrument commanding is carried out through ISOCSVC.EXE, so this main ISOC server must be running during calibration. Calibration consists of two distinct steps. First, the instrument's behavior is characterized in the frequency domain, and a set of representative frequencies is created. Second, for each frequency in the list, level measurement is performed to identify the ICOM level (a value between 0 and 255) with a known signal level. The resulting data set is saved in the Windows Registry and is used subsequently for all operations involving the receiver.

3.4.3. Rotator Calibrator

The Rotator Calibrator is an interactive utility with a simply GUI that allows the user to determine the A/D converter values associated with the end positions of the horizontal and vertical actuators in the rotator. It is assumed that the rotator's behavior is linear (i.e., that the actual rotation angle is a linear function of the A/D converter readout value.)

In addition to this calibration function, the rotator calibrator also allows the user to specify rotator parameters, such as its operating range, offsets, and obstruction lists. All information is saved in the Windows Registry, and is used during all subsequent operations involving the rotator.

The Rotator Calibrator communicates with the rotator directly through the RS-232 port.

3.4.4. The ISOC Service Manager

The ISOC Service Manager is a very simple utility that is used to selectively start or stop ISOCSVC.EXE and ISOCSCAN.EXE, when these applications run in the background as Windows NT services.

3.4.5. The Installation Support Library

The ISOC Installation Support Library contains extensions used in conjunction with InstallShield Express during application removal. Specifically, the library is used to remove ISOC service applications from the Windows NT Registry, and to wipe all Registry settings (including settings not created during installation) from under HKCU\SOFTWARE\Industry Canada\ISOC for Windows.

On client computers, two ISOC applications are installed: the ISOC client (ISOCNT.EXE) and the ISOC scheduler (ISOCSCHD.EXE). The ISOC client is the primary interface to remote ISOC servers that the instruments hosted there. The ISOC scheduler provides a GUI for managing background scanning tasks that are executed by a background scan server.

4.1. Bilingual Operation

All ISOC client applications are capable of operating in both English and French. This is accomplished the following way:

1. French-language versions of all user interface elements (resources) are created and maintained along with the English-language elements. The multi-language resource file editing capabilities of Visual Studio are used for this purpose.

2. Custom compilation steps are included to create French-language versions of the applications' resource files. These files are saved using the .FRC filename extension.

3. A Registry setting (that can be altered by selecting the Language option from the View menu in ISOCNT.EXE) determines if English, French, or the system default language should be used.

4. In each application that supports multiple languages, if the language setting is other than English, the .FRC file is loaded with a call to the system function LoadLibrary(), and activated with a call to AfxSetResourceHandle().

5. Applications support separate English and French-language Help files. During application startup, the name of the correct Help file is determined from the current language setting.

4.2. The ISOC client application

The ISOC client application appears to the user as a multiple-document interface (MDI) window, within which individual windows are displayed representing remote instruments. The client-server model of the ISOC suite permits a single copy of the ISOC client to connect to instruments on multiple servers simultaneously.

The user interacts with the ISOC client by selecting the Connect command from the File menu. This invokes a dialog (ISOC Servers) where the user can connect to a specific server (selected by IP address) and list the available instruments there. When a specific instrument is selected and the Connect button is clicked, an MDI child window representing the selected instrument is opened.

4.2.1. Document-View Architecture

Internally, the ISOC client follows the document-view paradigm of Microsoft Foundation Classes (MFC) applications. The current settings of a virtual instrument are represented by a document object, which allows, among other things, saving these settings to disk. Visual presentation of the instrument is performed by the corresponding view object.

4.2.1.1. Virtual Instrument Document Classes

The document classes representing virtual instruments are relatively simple. A typical document class, such as that of the HP-8594E spectrum analyzer (CHP8594EDoc) contains several data members representing instrument parameters, and a customized Serialize() member function for saving these settings to a disk file.

Although remote programming is not presently supported, the document classes are designed with programmability in mind. A future version of the ISOC may operate as an Automation server, permitting external programs (e.g., scripts written in Visual Basic) to control instruments through these document objects.

Currently, the following document objects are defined in the ISOC client:

CADVR3261ADoc	Advantec R-3261A spectrum analyzers
CDummyDoc	Dummy instruments (instruments with no functionality other than input selection) such as test antennae or audio inputs
CHP8594EDoc	Hewlett-Packard 8594E spectrum analyzers
CIFRCOM120BDoc	IFR COM-120B receiver/analyzer
CISOCNTDoc	Generic instrument support (for debug purposes)
CPCRDoc	PC-R1000 receiver support (incomplete, experimental)
CRotatorDoc	Antenna rotators
CRSESNDoc	Rhode & Schwarz ESN receivers
CRSSMHDoc	Rhode & Schwarz SMH signal generators
CSERDDoc	Support for Spectrum Explorer control via Remote Desktop
CSoundDoc	Audio output

Additionally, document objects defined in external DLLs (see below the subsection on modular design) include:

CCRCSEDoc	Support for the CRC Spectrum Explorer
CFLEXDoc	Support for FLEX tone decoders
CICOMDoc	ICOM R-8500 and R-9000 receivers
CDC44XDoc	OptoElectronics DC-440 and DC-448 tone decoders

CRSESMBDoc	Rohde & Schwarz ESMB support
CRSFSPDoc	Rohde & Schwarz FSP spectrum analyzer
CTBTCKDoc	Telonic-Berkeley TCK filter

Objects of these types are not created directly. Instead, they are constructed using the MFC document template mechanism. Construction occurs in CServersDlg::OnConnect() in response to the user's clicking the Connect button in the ISOC Servers dialog, after the correct template is identified and selected.

4.2.1.2. Virtual Instrument View Classes

All the visual interface elements for virtual instruments are encapsulated within the corresponding view class. In the current ISOC implementation, two distinct types exist: those derived directly from CFormView and those derived from CMultiForm.

CFormView-derived view classes present a simple "flat" interface model. In the case of more complex interfaces, such as the HP-8594E spectrum analyzer, a tabbed dialog approach is used to unclutter the visual area. Subdialogs are derived from the class CTabDlg that manages integration of the subdialogs with the main form view area. This solution, however, was found less than satisfactory by end users, so a new approach was developed.

This new approach makes use of a special class, CMultiForm, which itself is derived from CFormView. CMultiForm, in conjunction with another class, CSubForm (itself derived from CDialog) provide a simple generic "MDI within MDI" style user interface that made it possible to implement the GUI for various areas of instrument functionality in the form of individual sub-windows that can be moved within the client area of the instrument view. CMultiForm/CSubForm also support the automatic saving and reloading of window positions.

Some instrument views are also derived from CInstrumentDlg using multiple inheritance. CInstrumentDlg encapsulates some functionality related to the "tune-with" capability of the ISOC client application. In future releases, more generic (common to all instruments) functionality may be migrated from the individual instrument view classes to CInstrumentDlg.

The present set of ISOC view classes is as follows:

CADVR3261AView	**Advantec R-3261A spectrum analyzer**
CADVR3261AControlDlg	Advantec "Control" tab
CADVR3261AMarkerDlg	Advantec "Marker" tab
CADVR3261ASetupDlg	Advantec "Setup" tab
CADVR3261AStateDlg	Advantec "State" tab
CADVR3261ACalibrateDlg	Advantec "Calibrate" subdialog ("State" tab)
CADVR3261AFileDlg	Advantec "File" subdialog ("State" tab)

CADVR3261ASoundDlg	Advantec "Sound" subdialog ("State" tab)
CADVR3261ATraceDlg	Advantec "Trace" tab
CCRCSEView	**CRC Spectrum Explorer**
CDC44XView	**OptoElectronics DC-440/DC-448 tone decoders**
CCRCSEHistogramDlg	CRC SE "Histogram" panel
CCRCSEModsDlg	CRC SE "Modulation" panel
CCRCSEMsrmntsDlg	CRC SE "Measurements" panel
CCRCSEOpsDlg	CRC SE "Operations" panel
CCRCSESettingsDlg	CRC SE "Settings" panel
CDummyView	**Dummy instrument (antenna selector only)**
CHP8594EView	**Hewlett-Packard 8594E spectrum analyzer**
CHP8594EControlDlg	HP-8594E "Control" tab
CHP8594EMarkerDlg	HP-8594E "Marker" tab
CHP8594ESetupDlg	HP-8594E "Setup" tab
CHP8594EStateDlg	HP-8594E "State" tab
CHP8594ECalibrateDlg	HP-8594E "Calibrate" subdialog ("Control" tab)
CHP8594EDemodDlg	HP-8594E "Demod" subdialog ("Control" tab)
CHP8594EFileDlg	HP-8594E "File" subdialog ("Control" tab)
CHP8594ETraceDlg	HP-8594E "Trace" tab
CICOMView	**ICOM R-8500 and R-9000 receivers**
CICOMMainDlg	ICOM "Main" subdialog
CICOMMeterDlg	ICOM "Meter" subdialog
CICOMScanDlg	ICOM "Scan" subdialog
CICOMStateDlg	ICOM "State" subdialog
CISOCNTView	**Generic instrument view (debug only)**
CSetupDlg	Generic setup subdialog
CTraceDlg	Generic trace subdialog
CPCRView	**ICOM PC-R1000 view (incomplete, experimental)**
CPCRMainDlg	
CPCRMarkerDlg	
CPCRMeterDlg	
CPCRScopeDlg	
CPCRSettingsDlg	

CPCRTracesDlg	
CRotatorView	**Antenna rotator view**
CRotatorLvlDlg	Antenna rotator level subdialog
CRotatorMainDlg	Antenna rotator main subdialog
CRSESNView	**Rhode & Schwarz ESN receiver**
CRSESNMainDlg	ESN "Main" subdialog
CRSESNMeterDlg	ESN "Meter" subdialog
CRSESNSpecialDlg	ESN "Special" subdialog
CRSESNStateDlg	ESN "State" subdialog
CRSESNTraceDlg	ESN "Trace" subdialog
CRSESNAppearanceDlg	ESN "Appearance" tab ("Trace" subdialog)
CRSESNMarkerDlg	ESN "Marker" tab ("Trace" subdialog)
CRSESNSettingsDlg	ESN "Settings" tab ("Trace" subdialog)
CRSESNTracesDlg	ESN "Traces" tab ("Trace" subdialog)
CRSESNTTDlg	ESN "Time Traces" subdialog
CRSSMHView	**Rhode & Schwarz SMH signal generator**
CSoundView	**Sound output virtual instrument**

4.2.1.3. Document/View Operation

In the ISOC virtual instrument implementation, most of the work is actually performed by the view class or classes associated with subdialogs within the view.

Most activity is centered around a single CISOCInstrument object, which represents an ISOCInstrument ActiveX control. This object is defined as a dialog control and it is created automatically when the dialogs are constructed from templates by the MFC framework. View and dialog member functions frequently reference the Send() and Transact() member functions of this object to exchange data with the remote server.

Another recurring feature in virtual instrument implementations is the use of a common architecture for instrument parameters. Parameters are defined as an array of CParam objects, CParam being a local subclass of the document class associated with the instrument. This array of CParam objects is referenced in several places, including:

- The Serialize() member function of the document class, for saving instrument settings to disk;
- The DoDataExchange() member function of the view class, for exchange parameters between the GUI and the remote server;

- The GetParam() member function of the document class, and the related GetPar(), InvalidatePar(), and InvalidatePars() member functions in the view class, for parameter extraction.

Another recurring feature of the virtual instrument view classes is the use of a CKnobCtrl object and a corresponding array of CScroll objects. This array references a pair of controls: an edit control representing a value, and a (usually hidden) selector radio button. Using this control arrangement, a single rotating knob can be used to "tune" several scrollable parameters within the user interface.

Putting these features together, here's a brief overview of how a typical virtual instrument operates:

1. The view is created, causing the MFC framework to load and initialize any dialog templates. This implicitly creates an ISOCInstrument ActiveX control.

2. During initialization (usually in OnInitialUpdate) a connection is established to the remote ISOC server using member functions of the ISOCInstrument control, and an instrument is reserved. If any of these steps fail, the view is destroyed and an error message is displayed.

3. The instrument is reset and its current settings are queried, using the DoDataExchange() member function of the view (and its child windows, if any) to populate all GUI elements. At this point, the view is ready for user interaction.

4. Member functions are invoked in response to the user's actions. These functions communicate with the remote ISOC server using the Send() and Transact() member functions of the ISOCInstrument object. They also invalidate any parameters that need to be re-queried from the server; the re-query takes place in DoDataExchange().

5. The ISOCInstrument object may provide a visual interface (trace view) that is updated in response to data received in the background. If additional background data is received, this is processed by the OnBackgroundReceive() member function of the view.

4.2.2. Modular Construction

Like the ISOC server components, the ISOC client application can also utilize instrument support modules that are in the form of external DLLs. These external modules implement the document and view classes that are required to support a specific instrument or family of instruments. These support DLLs can also be used to construct derived instrument types.

The modular design is facilitated in part by the ISOCUI library that includes much of the code that is common to all instruments. Individual instrument document and view classes are usually classes derived from CInstrumentDoc and CInstrumentDlg, which are defined in this library.

The archetype instrument support module is ICOM.DLL. The code in this module was split from the main ISOCNT executable in order to facilitate the reuse of the ICOM code for the purposes of building DF support without unnecessary code duplication. Although no effort was made to split most older instrument support implementations from the main ISOCNT code branch, support for newer instruments was implemented in the form of DLL modules. Currently, the following instrument types are supported this way:

- Rohde & Schwarz FSP spectrum analyzer
- Telonic-Berkeley TCK filter
- ICOM receivers
- DC-44X tone decoders;
- FLEX tone decoders
- Rohde & Schwarz ESMB receiver
- DF instruments.

The list of DLLs that are to be loaded when ISOCNT.EXE starts is stored in the Registry under `HKLM\Software\Industry Canada\ISOC for Windows\ClientDLLs`. The default set of DLLs is `"RSFSP.DLL;TBTCK.DLL;ICOM.DLL;DC44X.DLL;FLEX.DLL;ESMB.DLL;DF.DLL"`.

4.2.3. Additional Support Classes

In addition to classes representing documents, views, and subdialogs, additional classes exist to represent other areas of the ISOC application user interface.

4.2.3.1. MFC Framework Support

In addition to two classes generated by the MFC AppWizard, CMainFrame and CChildFrame, a third class, CFixedSizeFrame, provides modified frame window behavior for instruments that are represented by a fixed size window.

4.2.3.2. ActiveX Controls

The current implementation utilizes three ActiveX controls. CISOCInst represents ISOCInstrument controls; CMETERCtrl represents V-U meter objects, and CKnob represents knob controls. Meters and knobs are not ISOC-specific components, but full source code for these components is provided.

4.2.3.3. The ISOC Servers Dialog

The ISOC Servers dialog appears in response to the user selecting the Connect command from the Site menu. This dialog allows the user to connect to a server of choice and obtain the list of available instruments there. In order to facilitate this, the dialog template associated with this control contains an invisible ISOCInstrument object; all communication with the server is performed through this object.

4.2.3.4. Miscellaneous Dialogs

The CAboutDlg class represents the application's "About" dialog that displays copyright and version information. The CCalibrateDlg class represents a dialog that is invoked when the user requests instrument calibration and the selected instrument is temporarily disabled. This dialog is usually displayed so as to simulate modal behavior, but rather than disabling the entire application window, only interactions with the specific instrument are prevented.

4.2.3.5. Debugging

The CDebugDlg provides a command-line interface and it is used for debugging purposes only.

4.2.3.6. Configuration Dialogs

The CColorOptionsDlg is displayed for instruments for which color selection for the visible trace area is permitted.

The CLangDlg is displayed when the user selects the Language command from the View menu, and allows the selection of English, French, or the system default language for the application's user interface.

4.3. The ISOC scheduler

The ISOC Scheduler provides a two-level user interface that lets the user monitor and manage scheduled background scanning tasks on a remote ISOCSCAN.EXE server.

4.3.1. Description

The main dialog of the ISOC Scheduler, represented by the CISOCSCHDDlg class, provides a list of all scheduled tasks on the selected server. The list displayed here corresponds with the list obtained from ISOCSCAN.EXE using the 'L' command (for a list of ISOCSCAN.EXE commands, please consult the ISOC for Windows Application Programming Interfaces manual.) Clicking the Edit button for any of the entries listed here invokes the "Edit existing schedule entry" dialog that provides a user interface for entering/updating schedule information.

This secondary dialog is actually a tabbed dialog; the three pages are labeled Setup, Schedule, and Files. The Setup page (CSetupDlg class) allows selecting the instrument for scheduled background operation. Depending on the type of instrument selected, the area labeled Instrument Parameters may contain an additional subdialog (CESNDlg along with CESNAdv, CICOMDlg, and CSNDDlg) that contains settings specific to that instrument.

The Schedule tab (CScheduleDlg) contains a custom ActiveX control that represents a weekly grid of 7×24 hours. The user can use the mouse to conveniently select any combination of hours that will then be used in the schedule.

The Files tab (CFilesDlg, along with CAddInputFilesDlg and CSelectOutputFileDlg) provides a means to select input and output files. In place of a set of input files, it is also possible to select a frequency range for scanning, in which case the list of scan frequencies will be automatically generated by the server.

4.3.2. Modular Design

The ISOC scheduler has also been converted to a modular design. Support for specific instrument types is implemented in the form of DLLs.

The list of DLLs that are to be loaded when ISOCCRON.EXE starts is stored in the Registry under `HKLM\Software\Industry Canada\ISOC for Windows\CRONDLLs`. The default set of DLLs is `"ANTCRON.DLL;DC4CRON.DLL;ESMBCRON.DLL;ESNCRON.DLL;ICOMCRON.DLL;\ IFRCRON.DLL;SMHCRON.DLL;SNDCRON.DLL;SECRON.DLL;ROTCRON.DLL;\ TCKCRON.DLL;DFCRON.DLL"`.

4.4. The ISOCInstrument.DLL control

The ISOCInstrument ActiveX control encapsulates the following core areas of ISOC client functionality:

1. Communication with an ISOC server, including instrument commanding
2. Background data reception
3. Display of graphical traces
4. Sound playback

Detailed information about the ISOCInstrument control and its usage can be found in the ISOC for Windows Application Programming Interfaces manual.

4.5. Non-ISOC Specific Components

The ISOC application suite uses a set of non-ISOC specific components that are part of the source distribution.

4.5.1. MULTISND.EXE

MULTISND.EXE is a simple, stand-alone COM server that provides digital audio mixing. In place of a lengthy explanation, here's a simple example program that uses MULTISND.EXE to play back a machine-generated sound. Note that you can run several copies of this program (perhaps with altered parameters) at the same time, with MULTISND.EXE properly mixing the resulting audio (that being the whole purpose of this simple client-server mechanism.)

```
// To compile, type: CL HCL.CPP OLE32.LIB OLEAUT32.LIB

#include <windows.h>
#include <stdio.h>
#include <conio.h>

void main(void)
{
```

```
    CLSID clsid;
    LPUNKNOWN punk;
    LPDISPATCH pdisp;
    DISPID dispid;
    OLECHAR *pszProp = L"Play";
    DISPPARAMS dispparams;
    UINT uArgErr;
    VARIANTARG vArg;

    vArg.bstrVal = SysAllocStringByteLen(NULL, 5120);
    char *pB = (char *)vArg.bstrVal;

    for (int i = 0; i < 2560; i++)
    {
        pB[i] = pB[i + 2560] = ((i >> 8) << 3) * ((i % 13 > 8) ? 1 : 0);
    }

    OleInitialize(NULL);
    CLSIDFromProgID(OLESTR("MULTISND.MultiSound"), &clsid);
    CoCreateInstance(clsid, NULL, CLSCTX_SERVER, IID_IUnknown, (LPVOID
            *)&punk);
    punk->QueryInterface(IID_IDispatch, (LPVOID *)&pdisp);
    punk->Release();
    pdisp->GetIDsOfNames(IID_NULL, &pszProp, 1, LOCALE_SYSTEM_DEFAULT,
            &dispid);
    dispparams.cArgs = 1;
    dispparams.cNamedArgs = 0;
    dispparams.rgdispidNamedArgs = NULL;
    dispparams.rgvarg = &vArg;
    vArg.vt = VT_BSTR;

    for (i = 0; i < 100; i++)
    {
        if (_kbhit()) break;
        pdisp->Invoke(dispid, IID_NULL, LOCALE_SYSTEM_DEFAULT,
            DISPATCH_METHOD, &dispparams, NULL, NULL, &uArgErr);
        Sleep(300);
    }
    SysFreeString(vArg.bstrVal);
    pdisp->Release();
    OleUninitialize();
}
```

4.5.2. The METER control

The METER custom control provides a simple graphical "V-U meter" style display. The control has the following properties:

BackColor, FillColor, and ForeColor control the appearance of the control.

The Caption property represents the text that may appear inside the control area.

The Appearance parameter is a numeric value representing the meter's present position between the Minimum and Maximum values.

The Ticks and Marks parameters define the number of small and large markers displayed. The defaults are 30 ticks and 5 marks.

The Minimum and Maximum parameters represent the lower and upper end of the scale.

4.5.3. The KNOB control

The Knob control provides a simple user interface element that works in a fashion similar to a rotary knob on a radio instrument. Clicking the control with the mouse and performing a rotary motion can be used to adjust the control; when this occurs, messages are sent to the application indicating that the control position has changed.

The Knob control provides the following properties:

The control's appearance and color are controlled by BevelColor, SpotColor, BevelSize, SpotSize, BevelRimColor, TickColor, BevelShadowColor, SpotShadowColor, SpotRimColor, TickSize, BevelRimSize, SpotRimSize, and Ticks.

The control's positioning is controlled by the Resolution and Position parameters.

When the user interacts with the control, the control sends a Scroll event to the application. The even has a single parameter that is a signed "delta" value indicating the amount by which the user adjusted the control. The control also captures the mouse; the application can use the GetCapture() windows function to determine if the mouse is still being captured by the control.

4.5.4. The SCHEDULE control

The Schedule control displays a weekly grid of 168 hours. The user can interact with the control to select any combination of hours, or invert the selection for a specific day, range of days, hour, range of hours, or the whole week by clicking on the appropriate header areas.

The Schedule control provides only two properties:

The Hours property can be used to read the user's current selection. It is a bitmask of 24 bits; an extra parameter determines which day of the week is referenced.

The Language property can be used to cause the control to display day names in English or French.

When the user interacts with the control, the control generates an Update event that can be captured by the controlling application.

Source code for post 2.13 versions of the ISOC is packaged in the form of two Visual C++ solutions: most components are organized as projects in the solution “MASTER”, whereas DF components are projects in the solution “MASTER ISOCDF”.

Each major ISOC component is packaged as a separate Visual C++ project. The project directories are the following:

ICOMCAL	The ICOM Calibrator application
IPSWEMU	IP switch matrix emulator (not in MASTER.sln)
ISOCCONF	The ISOC server configuration utility
ISOCCRON	The ISOC Scheduler client application
ISOCCRON\ANDCRON	Antenna scan setup dialog library
ISOCCRON\DC4CRON	DC-4XX scan setup dialog library
ISOCCRON\ESMBCRON	ESMB scan setup dialog library
ISOCCRON\ESNCRON	ESN scan setup dialog library
ISOCCRON\ICOMCRON	ICOM scan setup dialog library
ISOCCRON\IFRCRON	IFR scan setup dialog library
ISOCCRON\ROTCRON	Rotator scan setup dialog library
ISOCCRON\SECRON	Spectrum Explorer scan setup dialog library
ISOCCRON\SMHCRON	SMH scan setup dialog library
ISOCCRON\SNDCRON	Background audio recording setup dialog library
ISOCCRON\TCKCRON	TCK filter scan setup dialog library
ISOCGPS	GPS monitoring utility
ISOCInstrument	The ISOCInstrument.DLL ActiveX control
ISOCLIB	The ISOCLIB.DLL support library
ISOCMGR	The ISOC Service Manager
ISOCNT	The ISOC for Windows client application
ISOCNT\CRCSE	CRC SE interactive support library
ISOCNT\DC44X	DC44X interactive support library
ISOCNT\ESMB	ESMB interactive support library
ISOCNT\FLEX	FLEX interactive support library
ISOCNT\ICOM	ICOM interactive support library
ISOCNT\ISOCUI	ISOC user interface library
ISOCNT\RSFSP	FSP interactive support library
ISOCNT\TBTCK	TCK interactive support library
ISOCSCAN	The ISOC background scanner service
ISOCSCAN\ANTSCAN	Antenna scanner library
ISOCSCAN\DC4SCAN	DC-44X tone decoder scanner library
ISOCSCAN\ESMBSCAN	ESMB receiver scanner library
ISOCSCAN\ESNSCAN	ESN receiver scanner library
ISOCSCAN\GPSSCAN	GPS scanner library

ISOCSCAN\ICOMSCAN	ICOM receiver scanner library
ISOCSCAN\IFRSCAN	IFR receiver scanner library
ISOCSCAN\ROTSCAN	Rotator scanner library
ISOCSCAN\SESCAN	Spectrum Explorer scanner library
ISOCSCAN\SMHSCAN	SMH signal generator scanner library
ISOCSCAN\SNDSCAN	Sound background recording library
ISOCSVC	The main ISOC service
ISOCSVC\CIVSVC	CI-V communications library
ISOCSVC\DF7SVC	Doppler DDF 70001 communications library
ISOCSVC\GPIBSVC	GPIB communications library
ISOCSVC\RS232SVC	RS-232 communications library
ISOCSVC\RSIBSVC	RSIB communications library
ISOCSVC\SOUNDSVC	Sound communications library
ISOCSVC\TCPIPSVC	TCP-IP communications library
LOG	Logging library (not a project)
ROTCAL	Calibration utility for the Antenna Rotator
SETUP	Setup projects
SETUPEN	English-language setup project
SETUPFR	French-language setup project

Additionally, the following directories contain DF projects:

DFREG	DF registry utility
ISOCCRON\DFCRON	DF scheduler UI library
ISOCDF	The ISOC DF application
ISOCLIB\DFLIB	DF utility library
ISOCNT\DF	DF user interface components
ISOCSCAN\DFSCAN	DF scanner library
ISOCSCAN\GPSSCAN	GPS scanner library
ISOCSCAN\OARSCAN	OAR scanner library
ISOCSCAN\RSSCAN	RS scanner library
ISOCSVC\DFSVC	DF communications library

These project directories must be present at the same level in the directory tree hierarchy in order for all components to compile correctly. (However, they can be in separate branches: e.g., ISOCNT for the main components, and ISOCDF for the DF components.) Additionally, the following directories are required for compilation:

GPIB:	GPIB library
GSM:	GSM sound compression library
LOG:	Logging functions
RSIB:	RSIB library
ULAW:	Additional sound compression libraries
ZLIB:	Additional compression libraries

Non-ISOC specific components include the following:

KNOBCTRL:	The rotating "knob" control
LED:	LED-like (7-segment) numerals (presently unused)
METER:	V-U meter control
MULTISND:	Sound mixer COM server
SCHEDULE:	Weekly schedule grid control

Non-ISOC components are provided in precompiled form, and if recompilation is needed, must be recompiled as per individual project instructions.

All ISOC and ISOCDF components can be compiled at once by rebuilding the “MASTER” and “MASTER ISOCDF” projects. A build failure may occur during rebuild, but a repeated build should be successful. In particular, it may be necessary to build “MASTER”, then “MASTER ISOCDF”, and then “MASTER” again, to ensure that the setup toolkits pick up the most recently compiled DF components.

Setup and Packaging

Rebuilding the “MASTER” and “MASTER ISOCDF” projects creates installation kits in the form of MSI files. While these can be installed “as is”, a further requirement of the ISOC project is to have password-protected executable installation executables.

Currently, these executables are created using the English-language x64 version of WinRAR (version 3.93), to which the French-language UI library (Default-FR.SFX) has been added manually, as per multilingual WinRAR setup instructions. The installation kits can be generated by following these steps:

1. Start WinRAR and navigate to the folder where the desired MSI file is located;
2. Select MSI file
3. Click Add
4. Under General tab
    a. Change extension to .exe
    b. Check Create SFX archive (should be autochecked already)
5. Under Advanced tab
    a. Click SFX options
        i. Under General tab
            1. Enter msi file name under Run after extraction
        ii. Under Modes tab
            1. Check Unpack to temporary folder
        iii. Under Module tab
            1. select Default-FR.SFX (for French installations only; module must be preinstalled in WinRAR folder)
    b. Click OK
    c. Click Set password
        i. Enter NewISOC
6. Click OK