Web Based Data Acquisition

December 1, 2008

Henry Grimmick
FGCU
CEN Data Acquisition and Control Systems
Professor Janusz Zalewski
# Table of Contents

1. Introduction 5

2. Problem Description 11

3. Web Access to the Xbee Sensor Array System 13
   3.1. Modification to the Xbee Sensor Array System Database to Support Advanced Web Reporting 21
   3.2. Modifications to the Xbee Sensor Array Web Based Reporting System 25
   3.3. Addition of Web Based Control Functions to the Xbee Sensor Array System 27

4. Web Access to the DataQ DI-158 28
   4.1. Testing the DataQ DI-158U 29
   4.2. Sample Data Analysis Using DataQ Provided Tools 31
   4.3. DataQ SDK and Access to the DI-158U With Visual Dataflex 33
   4.4. Using the DataQ SDK with Visual Dataflex 34
   4.5. Communicating With the DataQ DI-158U Using a Dataflex Win32 Application 35

5. Users Guide 37
   5.1. Component Listing 37
   5.2. Installation Overview 38
   5.3. Code Development Overview 39

6. Conclusion 40

References 41

Appendix A: Sample Java Code to Implement Connection to the Faircom SQL Database via the Internet 42
Appendix B: Sample HTML Page to Load Reporting Applet and Connect to the Faircom SQL Database

Appendix C: DataQ DI-158U Connection Diagram

Appendix D: DataQ ActiveX Object Wrapped in a Dataflex Class

Appendix E: Dataflex Source Code for Communication with the DataQ DI-158U
Glossary of Terms

1. **ISDN**: ISDN (Integrated Services Digital Network) is a set of CCITT/ITU standards for
digital transmission over ordinary telephone copper wire as well as over other media.
Transfer rates are up to 128Kbps
1. Introduction

Simply defined, data acquisition is the sampling of the real world processes to generate data that after being manipulated by a computer would give us insight into the behavior of these processes. [1] The methods of sampling data ranges from low technology solutions such as a user manually recording the current temperature in a log, to medium technology solutions such as circular or strip chart mechanical recorders to high technology solutions such as computer based sensor reading and control systems.

With the development of low cost personal computers in the early 1980’s, high tech data acquisition capabilities became available to the masses for the first time. And with that new data acquisition capability came the ability to control devices automatically. Processes which required constant, expensive human monitoring around the clock could now be monitored and controlled electronically, significantly reducing the operational costs and increasing the efficiency of the business. [Fig 1.1]

![Fully Customizable Data Acquisition & Control System](image)

**Fig 1.1 Sample Data Acquisition & Control System [6]**
The early 1990’s introduced the Internet to the public for the first time. This allowed the low cost transfer of data, at near instantaneous speeds, between two computer systems thus eliminating the need for slow and costly communication methods such as dial-up modems, and ISDN communications. For the first time systems not directly connected to each other could communicate among themselves and exchange data back and forth. This new communication medium opened up a new frontier in data acquisition as systems could be monitored and controlled from remote locations.
In the previous project paper Serial Based Wireless Sensor Arrays for Use in Construction, Land Surveying and Agriculture [2] we introduced a system of wireless sensor arrays which collected various soil, plant moisture and atmospheric readings on a periodic basis and submitted these readings to an SQL database system.

Figure 1.2 summarizes the wireless sensor array system developed in this project. A central Control Computer located in a main facility houses both the data acquisition program and the SQL database engine. Attached to this central computer is the Xbee Pro Base Unit which is a 2.4ghz transceiver. [7] The Base Unit communicates with a single Xbee Pro Remote Unit at a time by switching communication channels to that of the desired Remote Unit housing a sensor. With the proper antennas communication between the Xbee Pro Base and Remote Unit can be achieved from up to one mile away. Unlimited communication range can be achieved by adding additional Xbee transceiver units in repeater mode. Once communications between the Base and Remote unit has been established, a virtual RS-232 serial port is emulated allowing for direct communication to a serial device attached to the Xbee Pro Base unit. For this system the 1-Wire DS9097U serial to 1-Wire Network adapter was attached to each Xbee Pro Base unit. Attached to each DS9097U are various, weather, soil moisture and plant moisture sensors which used the 1-Wire communication protocol.
At regularly scheduled intervals [Fig 1.3], the data acquisition software located on the Central Computer would initiate communications with the desired Xbee Pro Remote unit, open the serial link between the base and remote units, activate the 1-Wire DS9097U serial to 1-Wire network adapter, query each of the sensors attached to the DS9097U and record these values to the SQL database for future analysis.

Fig 1.3 Logical Flow of the Sensor Network Software
This future analysis was envisioned to be achieved by allowing remote access to the database by a third party GIS system where the data would be queried on a regular basis to help provide the agricultural operator with a complete picture of the operation to assist in determining irrigation needs and crop performance.
2. Problem Description

While the Xbee based data acquisition system is stable and reliable, it is only a self-contained one-way system which pulls data from the remote sensor array and records the readings from the sensor units to a SQL database. There is no facility to control the flow of the readings from each sensor array nor is there any ability to present the data to the system user in a meaningful format. All access to the data was designed on a pull basis from a third party GIS system where it would be integrated with other related data and then presented to the user. However as the data needed to be pulled from the system and manipulated by a third party, there is a realistic probability that once the data finally makes it back to the field operator, it is outdated as field conditions may have changed significantly during the time it took to process the report.

What is needed in this system is the addition of a real-time reporting and system status module. [Figure 2.1] Additionally this module should be accessible from the Internet not only from the standpoint of convenience for the operator, such that they do not need to physically go directly to the central computer system, but also for instantaneous access to the data wherever the operator may be located.

Figure 2.1 Web Based Reporting and Control Flow
In the first half of this document we describe the development and implementation of an Internet based status and control module for the Xbee Wireless Sensor Array System. This module provides the operator with a series of status reports such as status by array, trend by array, status of the entire system, average readings by the system and trend by the entire system. Additionally, this module allows for some basic control of the system such as controlling the interval between array readings and the ability to change which array module is read next.

In the second half of this document we look at a more traditional data acquisition product from DataQ Instruments. [3] The DataQ DI-158 is a low cost USB based data acquisition device which contains four analog channels supporting up to 200 volts, four digital bits for general I/O, and two 0 to 1.25 volt digital to analog output ports. We address the steps required to communicate with this device and acquire data using the DATAQ Active X control library provided by the device manufacturer. This device is accessible from the Internet using the data and Internet aware programming language DataFlex [4].

In summary, both implementations are expected to prove the usefulness of data acquisition and presentation over the Internet possibly with control functions included.
3 Web Access to the Xbee Sensor Array System

3.1 Basic Internet Access

In this section we look at the steps required to connect to, report from and implement some basic control of the Xbee Sensor Array System via the Internet. As previously mentioned, the Xbee Sensor Array System is connected to a SQL database in the Central Computer. The primary function of the database in the project is to record sensor data for future analysis, however with some slight modification the database can also be used as a pass-through mechanism to initiate some control of the Xbee Sensor Array System.

The database used in this project is the CtreeSQL database engine from Faircom Corporation. [3] This database product was chosen as it is platform independent, requires a very small footprint, designed for both embedded and large scale deployments, and does not require a complex administration as in products from Oracle and Microsoft. In addition to offering a robust SDK where a developer can produce their own database engine driver to the Faircom server, Faircom Corporation provides both a Windows client SQL driver and a JDBC driver, contained in a .JAR file, which is accessed by both the local and web based client.
The following steps provide details on reporting from the Xbee Sensor Array System via the Web using JDBC [Figure 2.2].

**Step 1.** To have connectivity to the Faircom SQL database via JDBC it is necessary to install the JDBC driver file, ctreeJDBC.jar, on the web server machine which will also host the Java reporting and control applet. Incorrect installation information provided in the initial documentation from the vendor may cause some delays in implementing a successful installation and connection to the Faircom database via JDBC.

![Image of communication flow between client & server](image)

**Step 2.** It is necessary to create the java applet which will use the JDBC driver and connect to the Faircom SQL database, (see Appendix A). The sub steps required to connect to the Faircom SQL server using the JDBC driver are as follows:

1. Initialize the JDBC SQL database driver contained in the ctreeJDBC.jar file.
   ```java
   Class.forName("ctree.jdbc.ctreeDriver");
   ```

2. Create the connection and login string to the database engine.
   ```java
   conn = DriverManager.getConnection("jdbc:ctree:6597@localhost:myDatabase", "ADMIN", "ADMIN");
   ```
Upon initialization and a successful login to the Faricom SQL server the user will receive notification of the successful connection in the Java Console window on the local PC. [Fig 2.3]

Fig 2.3 Java Console showing successful connection to the Faircom SQL Server
**Step 3.** In the connection process it is necessary to build and send a SQL statement to the Faircom SQL server for processing. The first section of this project addresses only reporting from the Xbee Sensor Array System. The reporting SQL statement is formatted as follows:

```java
ResultSet rs = stmt.executeQuery("select * from sensordata");
```

Once the applet code is completed, the applet and the ctreeJDBC.jar file need to be called from within the same HTML page. [Fig 2.4]

![Simple HTML page to call applet and load the Faircom JDBC driver.](image)

Fig 2.4 Simple HTML page to call applet and load the Faircom JDBC driver.
Upon successful compilation of the sensorWeb applet file, the ctreeJDBC.jar, and the calling html file must all be loaded to the web server. [Fig 2.5] In addition, the web server machine must be hosting the Faircom SQL server database engine locally or must be able to connect to another machine running the Faircom SQL server process as defined in the connection and login string and described in Step 2. The web server used in the project is the Microsoft Internet Information Server (IIS) which is installed on an XP Professional workstation.

![Fig 2.5 Contents of c:\inetpub\wwwroot\XbeeSensorWeb](image)

Fig 2.5 Contents of c:\inetpub\wwwroot\XbeeSensorWeb
**Step 4.** From any local or remote PC which has access to the web server machine, load the applet via the HTML page described in the prior section. This will load the applet and ctreeJDBC.jar SQL driver file. [Fig 2.6]

![Fig 2.6 Successful load of the Java applet in Internet Explorer.](image)

In this test applet, the connection to the SQL server and the sensor status query functions are initiated manually by the user clicking on the corresponding button. Future revision of this applet will automatically initiate the connection to the Faircom SQL database and provide the user with a number of query and control options.

To initiate a connection to the Faircom SQL server the user simply presses the “Connect” button, as stated prior. A successful connection is logged into the Java Console window. [Fig 2.7]
Fig 2.7 Successful connection to the Faircom SQL server by pressing the Connect button.

Once the successful connection is established, the user can simply click the “Query/Draw” button to send the pre-defined report query to the SQL server. Upon completion, like any other SQL query, the result set is returned back to the user and displayed in the browser. [Fig 2.8]
Fig 2.8 Successful query result returned back to the user’s browser.
3.2 Essential Modifications to the Basic System

3.2.1 Modification to the Xbee Sensor Array System Database to Support Advanced Web Reporting

In section 3 it was discovered that the database structure and reporting logic used did not necessarily return the last value recorded to the Faircom SQL database by the Xbee Sensor Array System. This was due to the fact that there was no unique database key defined in the database table `sensordata`. The original table `sensordata` was structured as illustrated in Table 3.1.

Table 3.1 Structure of database table `sensordata`

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>sensornumber</td>
<td>VARCHAR</td>
<td>5</td>
</tr>
<tr>
<td>sensortype</td>
<td>VARCHAR</td>
<td>15</td>
</tr>
<tr>
<td>sensorreading</td>
<td>VARCHAR</td>
<td>10</td>
</tr>
</tbody>
</table>

This original database structure was practical for collecting sensor reading in chronological order and outputting to Excel or other system for analysis. However it was not possible to provide meaningful query capabilities to the web based reporting component of the Xbee Sensor Array System. To perform more complex reporting and analysis a unique record identifier and timestamp were added to the database structure.
A unique record identifier was added to the system so that reporting functions such as
displaying the last record, last X records, last X records entered by sensor Y and other more
complex queries could be performed against the database. Determining the proper logic for
generating the unique record identifier posed a bit of a challenge and the following three
methods were evaluated:

- Implement a counter stored and updated in the database
- Implement a counter based on the number of cycles that the system performed
- Implement a counter based upon a date/time timestamp

Implementing a counter stored and updated in the database was possible and would provide a
unique record identifier for each new record inserted into the database however the following
problems existed with this solution:

- Required the addition of a new table to the system database.
- Increased space required for the database.
- More complex record insertion during data collection as the control record would
  have to be read, incremented, and saved each time a new sensor data record was
  inserted into the database.
- Significantly increased disk reads and writes due to manipulating the additional
  table and records.
- Provisions would have to be added to rollback the counter in the event of a failed
  insert in order to maintain data integrity.

Implementing a counter based on the number of cycles that the system performed was ruled
out as this would reset the system counter to zero each time the Xbee Sensor Array System
was restarted. This would result in duplicate data and the operator would not be able to
determine the chronological order of the data in the database.

Implementing a counter based on a date/time timestamp was ultimately decided on and was
added to the system in the following manner:

- A new field `sensortimestamp` was added to the `sensordata` table. See
  Table 3.2
- After a successful sensor read a timestamp was created by calling the Java
  function `System.currentTimeMillis()`

`System.currentTimeMillis()` returns the current time in milliseconds based upon the
difference, measured in milliseconds, between the current time and midnight, January 1,
1970 UTC. This is useful as the data returned is stored in the database as a large unique
integer thus eliminating the possibility for duplicate data and insuring the ability to report
data in chronological order from the database.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>sensortimestamp</td>
<td>VARCHAR</td>
<td>40</td>
</tr>
<tr>
<td>SensorNumber</td>
<td>VARCHAR</td>
<td>5</td>
</tr>
<tr>
<td>Sensortype</td>
<td>VARCHAR</td>
<td>15</td>
</tr>
<tr>
<td>sensorreading</td>
<td>VARCHAR</td>
<td>10</td>
</tr>
</tbody>
</table>
The addition of the `sensortimestamp` field insured that a unique record identifier was available in the database to allow the user advance SQL reporting functions, however the data written to the record identifier field was meaningless to the user of the system. For example the value 1226206504484 is the number of milliseconds between midnight, January 1, 1970 UTC and 11:55 pm, November 8, 2008. While 1226206504484 is definitely a unique integer it is meaningless as a human readable timestamp and requires the reporting user to perform complex functions in SQL to transform the integer to a meaningful date. As the `sensortimestamp` data was already in memory, converting it to a human readable value was possible by using the Java timestamp function. This converted the integer from 1226206504484 to the string value 2008-11-08 23:55:04.484. Additionally another field `sensordatetime` was added to the database table `sensortimestamp` to store the human readable timestamp data. (See Table 3.3)

Table 3.3 Modified structure of database table sensordata supporting human readable date field `sensordatetime`
3.2.2 Modifications to the Xbee Sensor Array Web Based Reporting System

With the addition of the unique record identifier field `sensortimestamp` to the `sensordata` database table, it is possible to perform queries against the database based upon more advanced selection criteria. Figure 3.2.1 shows the modified sensorWeb applet program performing a query against the last record written to the database.

![Fig 3.2.1 Modified sensorWeb Applet to Report Last Record Written to the Database](image)

---

Page 24 of 81
The sensorWeb applet was further modified to improve the usability and readability of program and prepare for the addition of control functions for the Xbee Sensor Array System. Figure 3.2.2 shows the result of this modification, as it is displayed in the user's browser.

![Image of modified sensorWeb applet](image.jpg)

Figure 3.2.2 Modified sensorWeb.html
3.2.3 Addition of Web Based Control Functions to the Xbee Sensor Array System

In this section we describe the addition of web based control functions to the Xbee Sensor Array System. In periods of active or abnormal weather conditions it may be desirable to modify the polling speed of the system to collect data at an increased rate. To support this desired behavior we needed to modify the sleep time between cycles on the fly. Historically the sleep value was hard coded to 2.5 seconds.

To add control of the sleep timing to the Xbee Sensor Array System, the following changes were made to the Xbee Sensor Array System:

- A new data table named `controlparams` was added to the database.
- The Xbee system control program was modified to default to a 2.5 second sleep value and on each cycle, check to see if a new sleep value was written to the database.
- The user applet was modified to display the current sleep value.
- The user applet was modified to provide the user with four additional sleep values options to be set by depressing a button. Sleep value options are hard set in the system as values too low will cause timing issues and failed reads.
4. Web Access to the DataQ DI-158

In this section we look at data acquisition using the DataQ DI-158 USB data acquisition device [Fig. 4.1]. The DI-158 is a low cost USB data acquisition device which offers:

- a. 12-bit measurement accuracy for analog samples
- b. Sampling rates up to 14,400 samples per second.
- c. Four ±10V fixed differential ports
- d. 200V peak sample voltage on the analog ports.
- e. 2 Digital to Analog converter ports
- f. Four Digital I/O ports

The DataQ DI-158 can acquire analog data ranging from 0-200V on its 4 analog ports. In addition the two 0-5V digital to analog ports allow the developer to send control signals to analog devices like variable speed motor control relays, lights and alarms. The four TTL level output can sent signals to LED’s to fire or control other high voltage processes by connecting the TTL level outputs to a digital relay.

Fig 4.1 DataQ DI-158U USB Data Acquisition Device [3]
Like in most PC based devices, the first step regarding software operation is to install the manufacturer’s device driver. It is highly recommended that all firewalling and anti-virus software be disabled during this process as the software accesses local machine devices and ports that could be falsely interpreted as virus like behavior. During the installation process, sample software and hardware configuration and test utilities are installed. Once the software is installed, the user is prompted to attach the device to the PC’s USB port for detection and configuration. This completes the driver and hardware installation and the product is now ready to use.

4.1 Testing the DataQ DI-158U

In this section we look at establishing communications with the DI-158U using the manufacturer provided test utility. To familiarize the user with the product DataQ provides a number of programs and utilities to confirm proper install of the device driver and hardware, read data from the four analog channels of the DI-158U and analyze the data collected.

Step 1. This step is to insure that the device has been installed correctly and is accessible by the local PC. To perform this step the user activates the DataQ Instruments Hardware Manager.

Upon initialization, the DataQ Hardware Manager will scan the local PC USB and serial ports for attached devices. Fig 4.1.1 shows the software after locating the installed DI-158U.
<table>
<thead>
<tr>
<th>Description</th>
<th>Model</th>
<th>Status</th>
<th>Serial#</th>
<th>IP Address</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>48A57DBB</td>
<td>DI-158-U</td>
<td>Available</td>
<td>48A57DBB</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Fig 4.1.1 DataQ Hardware Manager after finding attached DI-158U
Step 2. This step is to use the manufacturer provided data acquisition software to insure that the DI-158U is installed and working properly. DataQ includes a demo version of the professional data acquisition software Windaq with each of their products. This demo version is limited to 240 samples per second whereas their professional version and SDK support up to 14400 samples per second. To enter the Windaq-lite the user clicks on the “Start Windaq” button located on the DataQ Hardware Manager panel. This launches the Windaq-lite already in data acquisition mode. [Fig 4.1.2] For the purpose of this test a standard 9V alkaline battery was attached to the first analog channel shortly after software activation in order to confirm that not only the device was accessible but also that the analog channels were functioning. The result is shown in Fig. 4.1.2

![DI-158 Acquisition 48A57DBB](image)

Fig 4.1.2 Windaq-lite acquiring sample from 9V battery attached to analog port #1.
4.2 Sample Data Analysis Using DataQ Provided Tools.

In addition to providing sample test and acquisition tools, DataQ also provides some simple data analysis tools including a waveform browser and calculator for common data analysis functions. To perform this, we launched Windaq-lite, set the sampling rate to 100 samples per second, enabled all four analog channels and during the data acquisition run a standard 9V battery randomly connected and disconnected to the first analog port. Windaq is recording the signals from the analog channels and saving them in a file. Once the data from Windaq-lite was saved, the vendor provided tool, Windaq Waveform Browser was loaded and our sample data file was opened. [Fig 4.2.1]
The Windaq Waveform Browser allows the user to scroll back and forth through the data, change scale, transform, invert, view the waveform in higher detail and provides standard statistical analysis of the collected data. [Fig 4.2.2]

![Sample statistical analysis from Windaq Waveform Browser](image)

Fig 4.2.2 Sample statistical analysis from Windaq Waveform Browser
4.3 DataQ SDK and Access to the DI-158U with Visual Dataflex

DataQ provides, free of charge, an ActiveX SDK which allows software developers access to the DataQ hardware from their application without having to develop the technology to communicate directly with the DataQ device. The DataQ SDK will function with any development language that supports communication with standard ActiveX objects such as Visual BASIC, Visual C++, National Instruments LabVIEW, CEC TestPoint, Borland C++ Builder, Excel and many others. The SDK outlines the steps required and methods to call in order to establish communications, set desired properties and record data from the DataQ hardware. The flowchart in Fig. 4.3.1 outlines the steps required to communicate and acquire data from the DataQ devices.

![Flowchart](image)

Fig 4.3.1 Communication and acquisition steps required for DataQ hardware.
On program instantiation the developer must set a number of properties in the ActiveX object to begin communications with the DI-158U. The properties are:

- **ComDeviceDriver**: This sets or returns the device driver which defines the data acquisition device. By default, this method assumes the DI-158U is attached to the system USB port and loads the appropriate communication driver.

- **ComADChannelCount**: Setting this property is required to tell the Active X object how many of the analog channels are going to be enabled.

- **ComSampleRate**: Sets or returns the per-channel sample rate. Readings from 1 to 14,400 samples per second are supported.

- **ComEventPoint**: Sets or returns the EventPoint to determine how often NewData event will be fired. NewData is an event that fires when the number of available new data points exceeds the number specified in the EventPoint property.

- **ComStart**: Starts the Dataq device and begins to sample data.

Once the NewData Event is fired, the following events occur:

- **ComGetData**: Acquires all the available data from all enabled channels and returns the result in a string variable.

To stop acquiring data, the developer makes a call to the following function:

- **ComStop**: Stops the Dataq device and closes the device driver.
4.3.1 Using the DataQ SDK with Visual Dataflex

Multiple languages exist with native support for ActiveX objects, however few languages exist that are also data aware languages. To communicate with databases, languages such as Java, C++, .NET and Visual Basic require the developer to import non native database communication into their project and the developer must learn and implement code specific to the database being used such as SQL. Visual Dataflex [Fig 4.4.1] offers a different approach to this issue as being a data aware language. The entire language was developed around the native Dataflex database and should the developer desire to connect to another database such as MySQL, MS-SQL, Oracle, Faircom or IBM DB2, zero code changes are required, the developer simply includes the desired database driver file with the application and connectivity to the desired database is achieved.

Fig 4.4.1 Visual Dataflex Studio with test DataQ project loaded.
Additionally Visual Dataflex allows the developer to import an ActiveX object into their development environment and automatically translate and wrap the ActiveX object in a Dataflex class file making it immediately available to the developer using standard and familiar Dataflex syntax. [Appendix D] Visual Dataflex supports development of two types of applications, standard Win32 Windows applications and Web based applications using the Microsoft IIS web server and standard ASP web pages.

4.3.2 Communicating with the DataQ DI-158U Using a Dataflex Win32 Application

In this section we look at communication with the DataQ DI-158U using a standard Dataflex Win32 application. To begin this process we open the Visual Dataflex Studio and create a new project workspace. [Fig 4.5.1]
At this point we have a basic Dataflex application, as seen in [Fig 4.5.1] the a Dataflex application resides in a default Dataflex desktop space, much like any other Windows application. The first thing required is to create a view and add controls to this view for the user to access. In this view is where our actual program resides. [Fig 4.5.2]

![Fig 4.5.2 View created with controls.](image)

In this sample program four text boxes and a button have been added. Additionally the DataQ ActiveX object, already imported as a Dataflex class has been added to the view. When the button is pressed four calls to the DataQ ActiveX object are made and he results are returned into the corresponding text boxes in the view.
5. Users Guide

This section outlines the software and steps required to implement the required hardware and software so that work on this project can continue. The overall view of the system architecture is presented in Fig. 5.1.

![Fig 5.1 System Overview Diagram]
5.1 Component Listing

The following components are needed to make the sensor network fully operational.

5.1.1 Hardware Components

1. Digi Xbee Pro Development Kit (contains)
   3 Xbee Pro modules

2. Xbee modules
   1 USB Xbee interface master unit board
   4 RS-232 Xbee interface remote unit boards
   4 9vdc external power supplies.

Wireless RF Networking Modules and API

http://www.digi.com/products/wireless/zigbee-mesh/xbee-zb-module.jsp

2. Maxim 1-Wire

1-Wire sensors, network adapter and API

API available from:

http://www.maxim-ic.com/products/1-wire/

Network adapter, sensors and cables available from:

T3-R1-A: Temperature sensor

CHT-R1-A: Temperature sensor housing

DS9097U-A: 1-Wire Serial Adaptor

CABLE-A-P: 1 meter RJ-11 to RJ-45 1-Wire Interface Cable

http://www.hobbyboards.com
5.1.2 Software Components

1. SerialIO Serialport

   Commercial serial port driver for Java is available, although RXTX may be substituted:


   http://users.frii.com/jarvi/rxtx/

2. Faircom Corporation

   Database engine and JDBC API is available from:

   http://www.faircom.com
5.2 Installation Overview

5.2.1 Hardware Installation:

This section summarizes the installation of the hardware components defined in section 5.1. For detailed information see [2].

Step 1. Connect USB Xbee interface master unit board to the control computer.

- Connect USB cable to PC.
- Connect USB cable to USB Xbee interface master unit board.

Step 2. Connect 1-Wire network devices to RS-232 Xbee interface remote unit boards.

- Connect 1-Wire DS9097U-A to RS-232 Xbee interface remote unit.
- Connect CABLE-A-P to 1-Wire DS9097U-A
- Connect CABLE-A-P to T3-R1-A
- Connect 9vdc power adapter to RS-232 Xbee interface remote unit.
5.2.2 Software Installation

This section summarizes the installation of the components defined in section 5.1 for additional detailed information see [2]

**Step 1.** Install the Xbee device drivers, configuration utility and API included on the Xbee Professional Development Kit CD included with the hardware.

**Step 2.** Once the Xbee device drivers are installed, confirm communications between Xbee RF modules using the vendor provided loopback adapter and range test software.

**Step 3.** Install the Maxim 1-Wire API, attach a 1-Wire network adapter and 1-Wire sensor to the development PC and confirm communications between the 1-Wire network adapter and 1-Wire sensor.

**Step 4.** Install the SerialIO Serial Java serial port driver or the RXTX serial port driver on the local PC.

Note 1. At the time of this writing the 1-Wire API included a modified version of the RXTX serial port driver which could not be accessed outside the 1-Wire API and communications with the 1-Wire devices.

Note 2. Since the inception of this project, Digi, the vendor, has released an API to handle communications and control of the Xbee Wireless RF modules without having to use AT commands over the serial port. Using this API instead of the SerialIO or RXTX serial port drivers would be recommended.
Step 5. Install the Faircom Ctree SQL Server and Ctree API. At the inception of this project the current version of the Faircom Server and API was V8.27. The current version of this product is V9 Ctree Ace and a freely downloadable version for development purposes is available at the Faircom Website. This project is not dependant upon the use of the Faricom server or API. Other freely available database engine products such as MySQL should satisfy the project requirements. However the Faircom product was chosen by the author’s desire to work with a commercially developed and maintained product.

5.3 Code development overview.

This section summarizes the program development for the Xbee Wireless Sensor Array project. Complete details and sample source code may be found in [2].

Step 1. Communication between Xbee modules.

- Determine communication method: Broadcast or Point-to-Point.
- Set communication channels on Xbee master and remote units.
- Develop logic to switch between units using Point to Point or build proper formatted payload to send receiving unit information plus data when using Broadcast mode.

Step 2. Communication with 1-Wire devices.

- Load 1-Wire driver
- Determine communications speed
- Initialize 1-Wire network adapter
- Query 1-Wire network for devices or desired 1-Wire device
- Load driver for desired 1-Wire device.
- Return value from 1-Wire device and translate based upon formulas provided in API. For example the 1-Wire temperature sensor returns an integer, this integer must be converted by the developer to Celsius or Fahrenheit based on the project requirements.

**Step 3.** Initialize communication with the Faircom SQL database.

- Install the JDBC driver on the development workstation.
- Build the JDBC connection string for communication with the SQL server.

```java
private static void initialize ()
{
    System.out.println("Initialize Connection to Faircom SQL Server");
    try {
        // Load the driver class
        Class.forName("ctree.jdbc.ctreeDriver");

        conn = DriverManager.getConnection
        ("jdbc:ctree:6597@localhost:myDatabase", "ADMIN", "ADMIN");

        // Create a SQL statement
        stmt = conn.createStatement();
    }
    catch (Exception e) {
        System.out.println("Exception: " + e.getMessage());
    }
}
```
• Create the desired SQL tables.

```java
private static void define ()
{
    System.out.println("Define Sensor Data Database");
    String tblCreate = "CREATE TABLE SENSORDATA (" +
            "sensortimestamp VARCHAR(40), " +
            "sensordatetime VARCHAR(40), " +
            "sensornumber VARCHAR(5), " +
            "sensortype VARCHAR(15), " +
            "sensorreading VARCHAR(10)) ");

    try {
        stmt.executeUpdate(tblCreate);
    } catch (Exception e) {
        System.out.println("Exception: " + e.getMessage());
    }
}
```

• Populate the SQL tables as desired with the data returned from the 1-Wire sensors.

```java
private static void Add_Records (String sensorNumber, String sensorType, String tempsForSensor)
{
    Timestamp dateTime = new Timestamp(System.currentTimeMillis());
    long ts = System.currentTimeMillis();

    String field1 = sensorNumber;
    String field2 = sensorType;
    String field3 = tempsForSensor;

    System.out.println("Adding Records to Faircom Database");
    String sqlInsert = "INSERT INTO sensordata VALUES 
            ( "'" + ts + "," + dateTime + "," + field1 + "," + field2 + "," + field3 + ");
```
//String RecordData = { "(" + field1 + "," + field2 + "," + field3 + ")" };  
System.out.println(" timestamp " + ts);

try {

    // System.out.println(sqlInsert + RecordData);
    System.out.println(sqlInsert + RecordData);
    stmt.executeUpdate(sqlInsert + RecordData);
}

} catch (Exception e) {

    System.out.println("Insert Record Exception: " + e.getMessage());

}
References


5. Faircom Corporation, [http://www.faircom.com](http://www.faircom.com)


7. Digi Corporation, [http://www.digi.com](http://www.digi.com)
Appendix A: Sample Java Code to implement connection to the Faircom SQL database via the Internet

```java
import java.lang.*;
import java.applet.*;
import java.awt.Graphics;
import java.io.*;
import java.net.URL;
import java.sql.*;
import java.awt.*;

public class sensorWeb extends Applet {
    static Connection conn;
    static Statement stmt;
    public static String tempsForSensor = "";
    String tempReading = "";
    String sensorType = "";
    String sensorNumber = "";
    String output1 = "";
    String output2 = "";

    // init
    public void init() {
        add(new Button("Connect"));
        add(new Button("Query/Draw"));
    }

    public boolean action(Event e, Object arg) {
        if (((Button)e.target).getLabel() == "Connect") {
            System.out.println("Initialize Connection to Faircom SQL Server");
            try {
                // Load the driver class
                Class.forName("ctree.jdbc.ctreeDriver");
                conn = DriverManager.getConnection("jdbc:ctree:6597@localhost:myDatabase", "ADMIN", "ADMIN");

                // Create a SQL statement
                stmt = conn.createStatement();
            } catch (Exception e1) {
                System.out.println("Exception 1: " + e1.getMessage());
            }
        } else if (((Button)e.target).getLabel() == "Query/Draw") {
            try {
                // Execute a query
                ResultSet rs = stmt.executeQuery("select * from sensordata");
                output2 = "executed query";
                String clear = "                  ";
                while (rs.next()) {
                    output1 = rs.getString(1)+ rs.getString(2)+ rs.getString(3);
                    System.out.println(output1);
                }
            } catch (Exception e1) {
                System.out.println("Exception 1: " + e1.getMessage());
            }
        }
    }
}
```
catch (Exception e2) {
    System.out.println("Exception 2: " + e2.getMessage());
}

repaint();

return true;


public void start() {

}

// end start

public void paint(Graphics g) {
    g.drawString(output1, 10, 50);
    g.drawString(output2, 10, 100);

}

// end paint
Appendix B: Sample HTML page to load reporting applet and connect to the Faircom SQL database.

```html
<html>
<head>
  <title>Test</title>
</head>
<body>
  <p>Faircom ctreeSQL and Sensor Web Connect and Query test applet.</p>
  <center>
    <applet code="sensorWeb.class"
            archive="ctreeJDBC.jar"
            width=500 height=400>
    </applet>
  </center>
</body>
</html>
```
Appendix C: DataQ DI-158U Connection Diagram

DI-158 Series Connections

Connecting to Ground
The ground connection on the I/O terminal strip (labelled AGnd) should always be connected to a signal ground, if one is not inherently made through analog or digital signal connections. Without this connection, the DI-158 does not have a return path which could lead to inaccurate measurements. Under no circumstances should the DI-158 be used in a measurement situation where it will experience common mode voltages in excess of ±30 VDC or peak AC (the device’s full-scale range without damage).

Port Designations
Refer to the following for connections.

AGnd: Analog Ground
DAC0: Digital to Analog converter
DAC1: Digital to Analog converter
D3: Digital I/O bit 3
D2: Digital I/O bit 2
D1: Digital I/O bit 1—Remote Event Marker*
D0: Digital I/O bit 0—Remote Start/Stop*
DGnd: Digital Ground

Analog Channels: 4 Differential Analog Channels—“+” is the positive terminal; “-” is the negative terminal
Appendix D: DataQ ActiveX Object Wrapped in a Dataflex Class

// Visual DataFlex COM proxy classes generated from C:\WINDOWS\system32\DataqSdk.ocx
Use FlexCom20.pkg

Define OLEenumTrigSlope for Integer
  // SlopeNegative
  Define OLESlopeNegative for 1
  // SlopePositive
  Define OLESlopePositive for 0

Define OLEenumTrigMode for Integer
  // None
  Define OLENone for 0
  // AnalogSlopeTrigger
  Define OLEAnalogSlopeTrigger for 1
  // DigitalTrigger
  Define OLEDigitalTrigger for 2
  // SoftwareTrigger
  Define OLESoftwareTrigger for 5
  // AnalogLevelTrigger
  Define OLEAnalogLevelTrigger for 6

Define OLEenumHystIdx for Integer
  // Hysteresis0
  Define OLEHysteresis0 for 0
  // Hysteresis1
  Define OLEHysteresis1 for 1
  // Hysteresis2
  Define OLEHysteresis2 for 2
  // Hysteresis3
  Define OLEHysteresis3 for 3
  // Hysteresis4
  Define OLEHysteresis4 for 4
  // Hysteresis6
  Define OLEHysteresis6 for 5
  // Hysteresis9
  Define OLEHysteresis9 for 6
  // Hysteresis13
  Define OLEHysteresis13 for 7
  // Hysteresis19
  Define OLEHysteresis19 for 8
  // Hysteresis28
  Define OLEHysteresis28 for 9
  // Hysteresis40
  Define OLEHysteresis40 for 10
  // Hysteresis58
  Define OLEHysteresis58 for 11
  // Hysteresis84
  Define OLEHysteresis84 for 12
  // Hysteresis122
  Define OLEHysteresis122 for 13
  // Hysteresis176
  Define OLEHysteresis176 for 14
  // Hysteresis255
  Define OLEHysteresis255 for 15

// CLSID: {F88080E3-9B4B-11D3-ABFF-0040055DE809}
// Dispatch interface for DataqSdk Control
Class cCom_DDataqSdk is a Mixin

// Sets or returns the device driver which defines the data acquisition device. You can use the DATAQ Device Driver Wizard shipped with the
// ActiveX package to set up the driver from outside of the program
{ MethodType=Property DesignTime=False }
Function ComDeviceDriver Returns String
  String retVal
  Get ComProperty of (phDispatchDriver(Self)) 1 OLE_VT_BSTR to retVal
  Function_Return retVal
End_Function

// Sets or returns the device driver which defines the data acquisition device. You can use the DATAQ Device Driver Wizard shipped with the ActiveX package to set up the driver from outside of the program
{ MethodType=Property DesignTime=False }
Procedure SetComDeviceDriver String value
    Set ComProperty of (phDispatchDriver(Self)) 1 OLE_VT_BSTR to value
End_Procedure

// Sets the device ID if USB or Ethernet option is used, or if the device is used under Windows NT/2000
{ MethodType=Property DesignTime=False }
Function ComDeviceID Returns String
    String retVal
    Get ComProperty of (phDispatchDriver(Self)) 2 OLE_VT_BSTR to retVal
    Function_Return retVal
End_Function

// Sets the device ID if USB or Ethernet option is used, or if the device is used under Windows NT/2000
{ MethodType=Property DesignTime=False }
Procedure SetComDeviceID String value
    Set ComProperty of (phDispatchDriver(Self)) 2 OLE_VT_BSTR to value
End_Procedure

// Sets or returns the number of channels to be enabled. Channel allocation can be changed via ADChannelList method
{ MethodType=Property DesignTime=False }
Function ComADChannelCount Returns Short
    Short retVal
    Get ComProperty of (phDispatchDriver(Self)) 3 OLE_VT_I2 to retVal
    Function_Return retVal
End_Function

// Sets or returns the number of channels to be enabled. Channel allocation can be changed via ADChannelList method
{ MethodType=Property DesignTime=False }
Procedure SetComADChannelCount Short value
    Set ComProperty of (phDispatchDriver(Self)) 3 OLE_VT_I2 to value
End_Procedure

// Sets or returns the per-channel sample rate
{ MethodType=Property DesignTime=False }
Function ComSampleRate Returns Real
    Real retVal
    Get ComProperty of (phDispatchDriver(Self)) 4 OLE_VT_R8 to retVal
    Function_Return retVal
End_Function

// Sets or returns the per-channel sample rate
{ MethodType=Property DesignTime=False }
Procedure SetComSampleRate Real value
    Set ComProperty of (phDispatchDriver(Self)) 4 OLE_VT_R8 to value
End_Procedure

// Sets or returns the maximum burst sample rate from which the sample rate is derived. The property can only be used when the program is running
{ MethodType=Property DesignTime=False }
Function ComMaxBurstRate Returns Real
    Real retVal
    Get ComProperty of (phDispatchDriver(Self)) 5 OLE_VT_R8 to retVal
    Function_Return retVal
End_Function

// Sets or returns the maximum burst sample rate from which the sample rate is derived. The property can only be used when the program is running
{ MethodType=Property DesignTime=False }
Procedure SetComMaxBurstRate Real value
    Set ComProperty of (phDispatchDriver(Self)) 5 OLE_VT_R8 to value
End_Procedure

// Sets or returns the EventPoint to determine how often NewData event will be fired
{ MethodType=Property DesignTime=False }
Function ComEventPoint Returns Integer
// Sets or returns the EventPoint to determine how often NewData event will be fired
{ MethodType=Property DesignTime=False }
Procedure Set ComEventPoint Integer value
    Set ComProperty of (phDispatchDriver(Self)) 6 OLE_VT_I4 to value
End_Procedure

// Sets or returns the burst counter to fine tune sample rate. Refer to the SDK manual for more information
{ MethodType=Property DesignTime=False }
Function ComBurstCounter Returns Integer
    Integer retVal
    Get ComProperty of (phDispatchDriver(Self)) 7 OLE_VT_I4 to retVal
Function_Return retVal
End_Function

// Sets or returns the burst counter to fine tune sample rate. Refer to the SDK manual for more information
{ MethodType=Property DesignTime=False }
Procedure Set ComBurstCounter Integer value
    Set ComProperty of (phDispatchDriver(Self)) 7 OLE_VT_I4 to value
End_Procedure

// Sets or returns Trigger Method. Refer to the SDK manual for more information
{ MethodType=Property DesignTime=False }
Function ComTrigMode Returns OLEenumTrigMode
    OLEenumTrigMode retVal
    Get ComProperty of (phDispatchDriver(Self)) 8 OLE_VT_I4 to retVal
Function_Return retVal
End_Function

// Sets or returns Trigger Method. Refer to the SDK manual for more information
{ MethodType=Property DesignTime=False }
Procedure Set ComTrigMode OLEenumTrigMode value
    Set ComProperty of (phDispatchDriver(Self)) 8 OLE_VT_I4 to value
End_Procedure

// Sets or returns Trigger level in binary format
{ MethodType=Property DesignTime=False }
Function ComTrigLevel Returns Short
    Short retVal
    Get ComProperty of (phDispatchDriver(Self)) 9 OLE_VT_I2 to retVal
Function_Return retVal
End_Function

// Sets or returns Trigger level in binary format
{ MethodType=Property DesignTime=False }
Procedure Set ComTrigLevel Short value
    Set ComProperty of (phDispatchDriver(Self)) 9 OLE_VT_I2 to value
End_Procedure

// Sets or returns Trigger slope
{ MethodType=Property DesignTime=False }
Function ComTrigSlope Returns OLEenumTrigSlope
    OLEenumTrigSlope retVal
    Get ComProperty of (phDispatchDriver(Self)) 10 OLE_VT_I4 to retVal
Function_Return retVal
End_Function

// Sets or returns Trigger slope
{ MethodType=Property DesignTime=False }
Procedure Set ComTrigSlope OLEenumTrigSlope value
    Set ComProperty of (phDispatchDriver(Self)) 10 OLE_VT_I4 to value
End_Procedure

// Sets or returns Trigger channel index reference to the channel allocation
{ MethodType=Property DesignTime=False }
Function ComTrigScnChnIdx Returns Short

Short retVal
Get ComProperty of (phDispatchDriver(Self)) 11 OLE_VT_I2 to retVal
Function_Return retVal
End_Function

// Sets or returns Triger channel index reference to the channel allocation
{ MethodType=Property DesignTime=False }
Procedure Set ComTrigScnChnIdx Short value
Set ComProperty of (phDispatchDriver(Self)) 11 OLE_VT_I2 to value
End_Procedure

// Sets or returns the number of data points to be acquired before the trigger point
{ MethodType=Property DesignTime=False }
Function ComTrigPreLength Returns Short
Short retVal
Get ComProperty of (phDispatchDriver(Self)) 12 OLE_VT_I2 to retVal
Function_Return retVal
End_Function

// Sets or returns the number of data points to be acquired after the trigger point
{ MethodType=Property DesignTime=False }
Procedure Set ComTrigPostLength Short value
Set ComProperty of (phDispatchDriver(Self)) 13 OLE_VT_I2 to value
End_Procedure

// Sets or returns the trigger hysteresis index
{ MethodType=Property DesignTime=False }
Function ComTrigHysteresisIdx Returns OLEenumHystIdx
OLEenumHystIdx retVal
Get ComProperty of (phDispatchDriver(Self)) 14 OLE_VT_I4 to retVal
Function_Return retVal
End_Function

// Sets or returns the trigger hysteresis index
{ MethodType=Property DesignTime=False }
Procedure Set ComTrigHysteresisIdx OLEenumHystIdx value
Set ComProperty of (phDispatchDriver(Self)) 14 OLE_VT_I4 to value
End_Procedure

// Sets the digital output port state, if available. The property can only be used when the program is running
{ MethodType=Property DesignTime=False }
Function ComDigitalOutput Returns Short
Short retVal
Get ComProperty of (phDispatchDriver(Self)) 15 OLE_VT_I2 to retVal
Function_Return retVal
End_Function

// Sets the digital output port state, if available. The property can only be used when the program is running
{ MethodType=Property DesignTime=False }
Procedure Set ComDigitalOutput Short value
Set ComProperty of (phDispatchDriver(Self)) 15 OLE_VT_I2 to value
End_Procedure

// Returns the number of new samples that are available since the last call to the GetData or GetDataEx method
{ MethodType=Property DesignTime=False }
Function ComAvailableData Returns Integer
Integer retVal
Get ComProperty of (phDispatchDriver(Self)) 16 OLE_VT_I4 to retVal
Function_Return retVal
End_Function

// Returns the number of new samples that are available since the last call to the GetData or GetDataEx method
{ MethodType=Property DesignTime=False }
Procedure Set ComAvailableData Integer value
Set ComProperty of (phDispatchDriver(Self)) 16 OLE_VT_I4 to value
End_Procedure

// Returns the digital input port state, if available. The property can only be used when the program is running
{ MethodType=Property DesignTime=False }
Function ComDigitalInput Returns Short
  Short retVal
  Get ComProperty of (phDispatchDriver(Self)) 17 OLE_VT_I2 to retVal
  Function_Return retVal
End_Function

// Returns the digital input port state, if available. The property can only be used when the program is running
{ MethodType=Property DesignTime=False }
Procedure Set ComDigitalInput Short value
Set ComProperty of (phDispatchDriver(Self)) 17 OLE_VT_I2 to value
End_Procedure

// Returns the device model number. The property can only be used when the program is running
{ MethodType=Property DesignTime=False }
Function ComInfoBoardID Returns Short
  Short retVal
  Get ComProperty of (phDispatchDriver(Self)) 18 OLE_VT_I2 to retVal
  Function_Return retVal
End_Function

// Returns the device model number. The property can only be used when the program is running
{ MethodType=Property DesignTime=False }
Procedure Set ComInfoBoardID Short value
Set ComProperty of (phDispatchDriver(Self)) 18 OLE_VT_I2 to value
End_Procedure

// Returns the device serial number. The property can only be used when the program is running
{ MethodType=Property DesignTime=False }
Function ComInfoSerial Returns String
  String retVal
  Get ComProperty of (phDispatchDriver(Self)) 19 OLE_VT_BSTR to retVal
  Function_Return retVal
End_Function

// Returns the device serial number. The property can only be used when the program is running
{ MethodType=Property DesignTime=False }
Procedure Set ComInfoSerial String value
Set ComProperty of (phDispatchDriver(Self)) 19 OLE_VT_BSTR to value
End_Procedure

// Returns the gain configuration of the device. The property can only be used when the program is running
{ MethodType=Property DesignTime=False }
Function ComInfoPGL Returns Boolean
  Boolean retVal
  Get ComProperty of (phDispatchDriver(Self)) 20 OLE_VT_BOOL to retVal
  Function_Return retVal
End_Function

// Returns the gain configuration of the device. The property can only be used when the program is running
{ MethodType=Property DesignTime=False }
Procedure Set ComInfoPGL Boolean value
Set ComProperty of (phDispatchDriver(Self)) 20 OLE_VT_BOOL to value
End_Procedure

// Returns the last error code
{ MethodType=Property DesignTime=False }
Function ComLastErrorCode Returns Integer
Integer retVal
Get ComProperty of (phDispatchDriver(Self)) 21 OLE_VT_I4 to retVal
Function_Return retVal
End_Function

// Returns the last error code
{ MethodType=Property DesignTime=False } 
Procedure Set ComLastErrorCode Integer value 
  Set ComProperty of (phDispatchDriver(Self)) 21 OLE_VT_I4 to value 
End_Procedure

// Returns the rev number of the hardware
{ MethodType=Property DesignTime=False } 
Function ComInfoRev Returns Short 
  Short retVal 
  Get ComProperty of (phDispatchDriver(Self)) 22 OLE_VT_I2 to retVal 
Function_Return retVal 
End_Function

// Returns the rev number of the hardware
{ MethodType=Property DesignTime=False } 
Procedure Set ComInfoRev Short value 
  Set ComProperty of (phDispatchDriver(Self)) 22 OLE_VT_I2 to value 
End_Procedure

// Sets or returns the counter used in all the channels in the scan list to fine tune sample rate. Refer to the SDK manual for more information
{ MethodType=Property DesignTime=False } 
Function ComADCounter Returns Integer 
  Integer retVal 
  Get ComProperty of (phDispatchDriver(Self)) 23 OLE_VT_I4 to retVal 
Function_Return retVal 
End_Function

// Sets or returns the counter used in all the channels in the scan list to fine tune sample rate. Refer to the SDK manual for more information
{ MethodType=Property DesignTime=False } 
Procedure Set ComADCounter Integer value 
  Set ComProperty of (phDispatchDriver(Self)) 23 OLE_VT_I4 to value 
End_Procedure

// Acquires all the available data from all enabled channels and return the result in a Variant
Function ComGetData Returns Variant 
  Handle hDispatchDriver 
  Variant retVal 
  Get phDispatchDriver to hDispatchDriver 
  Get InvokeComMethod of hDispatchDriver 24 OLE_VT_VARIANT to retVal 
Function_Return retVal 
End_Function

// Defines the allocation of channels. The default is sequential, incremented
Procedure ComADChannelList Variant llChannel 
  Handle hDispatchDriver 
  Get phDispatchDriver to hDispatchDriver 
  Get InvokeComMethod of hDispatchDriver 25 OLE_VT_VARIANT to llChannel 
Function_Return llChannel 
End_Procedure

// Defines the counter setting. Refer to SDK manual for more info. If you change SampleRate property, this value will be over written by the driver
Procedure ComADCounterList Variant llCounter 
  Handle hDispatchDriver 
  Get phDispatchDriver to hDispatchDriver 
  Get InvokeComMethod of hDispatchDriver 26 OLE_VT_VOID to llCounter 
Function_Return llCounter 
End_Procedure

// Defines the front-end configuration as single-ended or differential. The default is single-ended
Procedure ComADDiffList Variant llDiff 
  Handle hDispatchDriver 

Get phDispatchDriver to hDispatchDriver
Send PrepareParams to hDispatchDriver 1
Send DefineParam to hDispatchDriver OLE_VT_VARIANT llDiff
Send InvokeComMethod to hDispatchDriver 27 OLE_VT_VOID
End_Procedure

// Defines the channel gain-index setting. The default index is 0, or a gain of 1, for all channels
Procedure ComADGainList Variant llGain
Handle hDispatchDriver
Get phDispatchDriver to hDispatchDriver
Send PrepareParams to hDispatchDriver 1
Send DefineParam to hDispatchDriver OLE_VT_VARIANT llGain
Send InvokeComMethod to hDispatchDriver 28 OLE_VT_VOID
End_Procedure

// Defines the acquisition method for intelligent oversampling. The default is Average
Procedure ComADMethodList Variant llMethod
Handle hDispatchDriver
Get phDispatchDriver to hDispatchDriver
Send PrepareParams to hDispatchDriver 1
Send DefineParam to hDispatchDriver OLE_VT_VARIANT llMethod
Send InvokeComMethod to hDispatchDriver 29 OLE_VT_VOID
End_Procedure

// Acquires specified amount of data and return it via the array
Function ComGetDataEx Short ByRef llBuffer Short llCount Returns Short
Handle hDispatchDriver
Short retVal
Get phDispatchDriver to hDispatchDriver
Send PrepareParams to hDispatchDriver 2
Send DefineParam to hDispatchDriver (OLE_VT_BYREF ior OLE_VT_I2) llBuffer
Send DefineParam to hDispatchDriver OLE_VT_I2 llCount
Get InvokeComMethod of hDispatchDriver 30 OLE_VT_I2 to retVal
Function_Return retVal
End_Function

// Outputs analog data via DAC port
Procedure ComDAOutput Short llData Short llPort
Handle hDispatchDriver
Get phDispatchDriver to hDispatchDriver
Send PrepareParams to hDispatchDriver 2
Send DefineParam to hDispatchDriver (OLE_VT_BYREF ior OLE_VT_I2) llData
Send DefineParam to hDispatchDriver OLE_VT_I2 llPort
Send InvokeComMethod to hDispatchDriver 31 OLE_VT_VOID
End_Procedure

// Starts ADC scanning
Procedure ComStart
Handle hDispatchDriver
Get phDispatchDriver to hDispatchDriver
Send InvokeComMethod to hDispatchDriver 32 OLE_VT_VOID
End_Procedure

// Stops ADC scanning
Procedure ComStop
Handle hDispatchDriver
Get phDispatchDriver to hDispatchDriver
Send InvokeComMethod to hDispatchDriver 33 OLE_VT_VOID
End_Procedure

// Grabs the latest specified amount (<32K) of data to the specified array
Function ComGetDataFrameEx Short ByRef llBuffer Short llCount Returns Short
Handle hDispatchDriver
Short retVal
Get phDispatchDriver to hDispatchDriver
Send PrepareParams to hDispatchDriver 2
Send DefineParam to hDispatchDriver (OLE_VT_BYREF ior OLE_VT_I2) llBuffer
Send DefineParam to hDispatchDriver OLE_VT_I2 llCount
Get InvokeComMethod of hDispatchDriver 34 OLE_VT_I2 to retVal
Function_Return retVal
End_Function

// Grabs the latest specified amount (<32K) of data and returns in a Variant
Function ComGetDataFrame Short llCount Returns Variant
    Handle hDispatchDriver
    Variant retval
    Get phDispatchDriver to hDispatchDriver
    Send prepareParams to hDispatchDriver 1
    Send DefineParam to hDispatchDriver OLE_VT_I2 llCount
    Get InvokeComMethod of hDispatchDriver 35 OLE_VT_VARIANT to retval
Function_Return retval
End_Function

Procedure ComAboutBox
    Handle hDispatchDriver
    Get phDispatchDriver to hDispatchDriver
    Send InvokeComMethod to hDispatchDriver -552 OLE_VT_VOID
End_Procedure

End_Class

// CLSID: {F88080E4-9B4B-11D3-ABFF-0040055DE809}
// Event interface for DataqSdk Control
Class cCom_DDataqSdkEvents is a Mixin

    // Fires when the number of available new data points exceeds the number specified in the EventPoint property
    { MethodType=Event }
    Procedure OnComNewData Short llCount
End_Procedure

    // Fires when the input buffer (around 60,000 samples) is overrun
    { MethodType=Event }
    Procedure OnComOverRun
End_Procedure

    // Error in the ActiveX or Device driver
    { MethodType=Event }
    Procedure OnComControlError Integer llCode
End_Procedure

    { Visibility=Private }
    Procedure RegisterComEvents
        Send RegisterComEvent 1 msg_OnComNewData
        Send RegisterComEvent 2 msg_OnComOverRun
        Send RegisterComEvent 3 msg_OnComControlError
    End_Procedure
End_Class

// CoClass
// ProgID: DATAQSDK.DataqSdkCtrl.1
// CLSID: {F88080E5-9B4B-11D3-ABFF-0040055DE809}
// DataqSdk Control
{ CLSID="{F88080E5-9B4B-11D3-ABFF-0040055DE809}" }
Class cComDataqSdk is a cComActiveXControl
    Import_Class_Protocol cCom_DDataqSdk
    Import_Class_Protocol cCom_DDataqSdkEvents

Procedure Construct_Object
    Forward Send Construct_Object
    Set psProgId to "{F88080E5-9B4B-11D3-ABFF-0040055DE809}"*
    Set psEventId to "{F88080E5-9B4B-11D3-ABFF-0040055DE809}"*
    Set peAutoCreate to acAutoCreate
End_Procedure
End_Class
Appendix E: Dataflex Source Code for Communication with the DataQ DI-158U

Use DFAllEnt.pkg
Use cCJStandardCommandBarSystem.pkg

Object oHtmlHelp is a cHtmlHelp
End_Object

Object oApplication is a cApplication
Set peHelpType to htHtmlHelp
End_Object

Use oEditContextMenu.pkg
Use oDEOEditionContextMenu.pkg

Object oMain is a Panel
Set Label To "My Project"
Set Location to 4 3
Set Size to 300 450

Object oCommandBarSystem is a cCJCommandBarSystem
Set pbTimerUpdate to True

Procedure OnCreateCommandBars
Handle hoOptions
Get OptionsObject to hoOptions
Forward Send OnCreateCommandBars
End_Procedure

Object oMenuBar is a cCJMenuBar
Object oFileMenu is a cCJMenuItem
Set peControlType to xtpControlPopup
Set psCaption to "&File"
Set psDescription to "Find, Save, Delete, Clear information or quit this application."
Set psCategory to "Menus"

Object oClearMenuItem is a cCJClearMenuItem
Set pbAddToDesignerMenu to True
End_Object

Object oClearAllMenu is a cCJClearAllMenuItem
Set pbAddToDesignerMenu to True
End_Object

Object oPromptMenuItem is a cCJPromptMenuItem
Set pbAddToDesignerMenu to True
Set pbControlBeginGroup to True
End_Object

Object oFindMenuItem is a cCJFindMenuItem
Set pbAddToDesignerMenu to True
Set pbControlBeginGroup to True
End_Object

Object oFindNextMenu is a cCJFindNextMenuItem
Set pbAddToDesignerMenu to True
End_Object

Object oFindPreviousMenu is a cCJFindPreviousMenuItem
Set pbAddToDesignerMenu to True
End_Object

Object oFindFirstMenu is a cCJFindFirstMenuItem
Set pbAddToDesignerMenu to True
End_Object

Object oFindLastMenu is a cCJFindLastMenuItem
Set pbAddToDesignerMenu to True
End_Object
End_Object

Object oSaveMenuItem is a cCJSaveMenuItem
  Set pbAddToDesignerMenu to True
  Set pbControlBeginGroup to True
End_Object

Object oDeleteMenuItem is a cCJDeleteMenuItem
  Set pbAddToDesignerMenu to True
End_Object

Object oExitMenu is a cCJExitMenuItem
  Set pbControlBeginGroup to True
End_Object

End_Object

Object oViewMenu is a cCJMenuItem
  Set peControlType to xtpControlPopup
  Set psCaption to "&View"
  Set psTooltip to "View"
  Set psDescription to "Available Views"

Object oodataQTestMenuItem is a cCJMenuItem
  Set psCaption to "odataQTest"
  Set psTooltip to "odataQTest"

Procedure OnExecute Variant vCommandBarControl
  Handle hoClient
  Get Client_Id to hoClient
  Send Activate_odataQTest of hoClient
End_Procedure
End_Object

End_Object

Object oReportMenu is a cCJMenuItem
  Set peControlType to xtpControlPopup
  Set psCaption to "&Report"
  Set psTooltip to "Report"
  Set psDescription to "Available Reports"
End_Object

Object oNavigateMenu is a cCJMenuItem
  Set peControlType to xtpControlPopup
  Set psCaption to "&Navigate"
  Set psTooltip to "Navigate"
  Set psDescription to "Move to different areas of the application"

Object oNextAreaMenu is a cCJNextAreaMenu
End_Object

Object oPriorAreaMenu is a cCJPriorAreaMenu
End_Object

Object oNextViewMenu is a cCJNextViewMenu
End_Object

Object oPriorViewMenu is a cCJPriorViewMenu
End_Object

Object oPromptMenu is a cCJPromptMenu
  Set pbControlBeginGroup to True
End_Object

Object oZoomMenu is a cCJZoomMenu
End_Object

End_Object

Object oWindowMenu is a cCJMDIWindowsMenu
Set peControlType to xtpControlPopup
Set psCaption to "&Window"
Set psToolTip to "Window"
Set psDescription to "Display Current Views and set other display options."

// These are the static windows items. More will be created in onInitPopup
Object oDisplayOptionsMenu is a cCJMenuItem
  Set peControlType to xtpControlPopup
  Set psCaption to "&Display Options"
  Set psToolTip to "Display Options"
  Set psDescription to "Set display options"
Object oStatusbarMenu is a cCJStatusbarMenu
  End_Object
Object oAutoArrangeIconsMenu is a cCJAutoArrangeIconsMenu
  End_Object
Object oRestoreMenusMenu is a cCJRestoreMenusMenu
  Set pbControlBeginGroup to True
  End_Object
  End_Object
Object oCascadeMenu is a cCJCascadeMenu
  Set pbControlBeginGroup to True
  End_Object
Object oHorizTile is a cCJTileHorizontally
  End_Object
Object oVertTile is a cCJTileVertically
  End_Object
Object oMinimizeMenuItem is a cCJMinimizeWindowsMenuItem
  Set pbControlBeginGroup to True
  End_Object
Object oRestoreMenuItem is a cCJRestoreWindowsMenuItem
  End_Object
Object oArrangeIconsMenuItem is a cCJAutoArrangeIconsMenuItem
  Set pbControlBeginGroup to True
  End_Object
  End_Object
Object oHelpMenu is a cCJMenuItem
  Set peControlType to xtpControlPopup
  Set psCaption to "&Help"
  Set psToolTip to "Help"
  Set psDescription to "Access Information for learning and using this DataFlex application."
  Set psToolBar to "Help"
Object oHelpMenuItem is a cCJHelpMenuItem
  End_Object
Object oAboutMenuItem is a cCJAboutMenuItem
  End_Object
  End_Object
Object oFindToolBar is a cCJToolbar
  Set psTitle to "Finding Toolbar"
  End_Object
Object oFindFirstTool is a cCJFindFirstMenuItem
  End_Object
Object oFindPreviousTool is a cCJFindPreviousMenuItem
  End_Object
Object oFindMenuTool is a cCJFindMenuItem
End_Object

Object oFindNextTool is a cCJFindNextMenuItem
End_Object

Object oFindLastTool is a cCJFindLastMenuItem
End_Object

Object oPromptToolItem is a cCJPromptMenuItem
Set pbControlBeginGroup to True
End_Object

End_Object

Object oFileToolBar is a cCJToolbar
Set psTitle to "Data Entry Toolbar"

Object oClearToolBarItem is a cCJClearMenuItem
Set peControlStyle to xtpButtonIconAndCaption
End_Object

Object oClearAllToolBarItem2 is a cCJClearAllMenuItem
Set peControlStyle to xtpButtonIconAndCaption
End_Object

Object oSaveToolBarItem is a cCJSaveMenuItem
Set peControlStyle to xtpButtonIconAndCaption
Set pbControlBeginGroup to True
End_Object

Object oDeleteToolBarItem is a cCJDeleteMenuItem
Set pbControlBeginGroup to True
End_Object

End_Object

Object oEditToolBar is a cCJToolbar
Set psTitle to "Edit Toolbar"

Object oCutToolBarItem is a cCJCutMenuItem
End_Object

Object oCopyToolBarItem is a cCJCopyMenuItem
End_Object

Object oPasteToolBarItem is a cCJPasteMenuItem
End_Object

Object oDeleteEditToolBarItem is a cCJDeleteEditMenuItem
Set pbControlBeginGroup to True
End_Object

End_Object

Object oStatusBar is a cCJStatusBar

Object oStatusPane1 is a cCJStatusBarPane
Set piID to sbpIDIdlePane
Set pbStyleStretch to True
End_Object

Object oStatusPane2 is a cCJStatusBarPane
Set phoViewPane to Self
Set pbStyleStretch to True
End_Object

End_Object
Object oClientArea is a ClientArea
   Use StdAbout.pkg
   Use dataQTest.vw

   Procedure Activate_About
      Send DoAbout "" "" "" ""
   End_Procedure

End_Object

End_Object

Start_UI
   Use Windows.pkg
   Use DFClient.pkg
   Use cComDataqSdk.pkg

   Deferred_View Activate_odataQTest for ;
   Object odataQTest is a dbView
      Set Border_Style to Border_Thick
      Set Size to 303 246
      Set Location to 2 2

   Object oButton1 is a Button
      Set Location to 236 139
      Set Label to "Refresh"

      // fires when the button is clicked
      Procedure OnClick
         Variant myVariant
         String hold
         Integer [] iaResult
         Pointer iaResultPointer
         Move 99 to iaResult[0]
         Move (AddressOf(iaResult)) to iaResultPointer
         Broadcast Send myRefresh to odataQTest
         Showln "Sending ComStart"
         Send ComStart to oDataQ
         Sleep 1
         Showln "Available Data" (ComAvailableData(oDataQ(Self)))
         // Showln "Sending ComGetData"
         Get ComGetDataFrame of oDataQ 1 to iaResult
         // Get ComGetDataFrameEx of oDataQ iaResultPointer 4 to iaResult
         // Showln "iaResult: " iaResult[0] " " iaResult[1] " " iaResult[2]
         Showln "myVariantLength:" (Length( myVariant))
         Showln "myVariant:" myVariant
         Showln "Sending ComStop"
         Send ComStop to oDataQ
      End_Procedure

End_Object

Object oDataQ is a cComDataqSdk
   Set Size to 35 89
   Set Location to 266 3

   Procedure OnCreate
      Forward Send OnCreate
      Showln "OnCreate:"
      // ToDo: Set the ActiveX properties here...
   End_Procedure
Procedure OnComNewData Short llCount
    Showln "LL COUNT RETURNED: " llCount
End_Procedure

Procedure OnComControlError Integer llCode
    Showln "ComControllerError:" llCode
End_Procedure

Embed_ActiveX_Resource
    AAABAL8NAADKBQAAAAAAAEAAAAAABAJ0AAAAA0AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA.
End_Embed_ActiveX_Resource

End_Object

Object oBoardID is a Form
    Set Size to 13 100
    Set Location to 148 88
    Set Label to "Board ID:

Procedure myRefresh
    Set Value to (ComInfoBoardID(oDataQ(Self)))
End_Procedure

    //OnChange is called on every changed character

    //Procedure OnChange
    //    String sValue
    //    //    Get Value to sValue
    //End_Procedure

End_Object

Object oBoardRevisionNo is a Form
    Set Size to 13 100
    Set Location to 165 90
    Set Label to "Board Rev#:

Procedure myRefresh
    Set Value to (ComInfoRev(oDataQ(Self)))
End_Procedure

    //OnChange is called on every changed character

    //Procedure OnChange
    //    String sValue
    //    //    Get Value to sValue
    //End_Procedure

End_Object

Object oChannelCount is a Form
    Set Size to 13 100
    Set Location to 183 90
    Set Label to "Channel Count:

Procedure myRefresh
    Set Value to (ComADChannelCount(oDataQ(Self)))
End_Procedure

    //OnChange is called on every changed character

    //Procedure OnChange
    //    String sValue
    //    //    Get Value to sValue
    //End_Procedure
Object oGetData is a Form
    Set Size to 13 100
    Set Location to 199 90
    Set Label to "Serial #"

Procedure myRefresh
    Set Value to (ComInfoSerial(oDataQ(Self)))
End_Procedure

//OnChange is called on every changed character
//Procedure OnChange
//    String sValue
//    // Get Value to sValue
//End_Procedure

End_Object

Object oDeviceDriver is a Form
    Set Size to 13 100
    Set Location to 10 98
    Set Label to "Device Driver"

Procedure Activating
    Forward Send Activating
    Set Value to (ComDeviceDriver(oDataQ(Self)))
End_Procedure

//OnChange is called on every changed character
//Procedure OnChange
//    String sValue
//    // Get Value to sValue
//End_Procedure

End_Object

Object oDeviceID is a Form
    Set Size to 13 100
    Set Location to 31 99
    Set Label to "Device ID"

Procedure Activating
    Forward Send Activating
    Set Value to (ComDeviceID(oDataQ(Self)))
End_Procedure

//OnChange is called on every changed character
//Procedure OnChange
//    String sValue
//    // Get Value to sValue
//End_Procedure

End_Object

Object oChannelCount is a Form
    Set Size to 13 100
    Set Location to 54 99
    Set Label to "Channel Count"

Procedure myRefresh
    Set Value to (ComADChannelCount(oDataQ(Self)))
End_Procedure
Procedure OnChange
  // String sValue
  Set ComADChannelCount of oDataQ to (Integer(value(Self)))
  // Get Value to sValue
End_Procedure

End_Object

Object oEventPoint is a Form
  Set Size to 13 100
  Set Location to 76 99
  Set Label to "Event Point"
  Set Value to 20

Procedure OnChange
  Set ComEventPoint of oDataQ to (Integer(value(Self)))
  // String sValue
  //
  // Get Value to sValue
End_Procedure

End_Object

Object oDeviceDriver is a Form
  Set Size to 13 100
  Set Location to 10 98
  Set Label to "Device Driver"

Procedure Activating
  Forward Send Activating
  Set Value to (ComDeviceDriver(oDataQ(Self)))
End_Procedure

//OnChange is called on every changed character

//Procedure OnChange
//  String sValue
//
//  // Get Value to sValue
//End_Procedure

End_Object

Cd_End_Object

// Visual DataFlex COM proxy classes generated from C:\WINDOWS\system32\DataqSdk.ocx
Use FlexCom20.pkg

Define OLEenumTrigSlope for Integer
  // SlopeNegative
  Define OLESlopeNegative for 1
  // SlopePositive
  Define OLESlopePositive for 0

Define OLEenumTrigMode for Integer
  // None
  Define OLENone for 0
  // AnalogSlopeTrigger
  Define OLEAnalogSlopeTrigger for 1
  // DigitalTrigger
  Define OLEDigitalTrigger for 2
  // SoftwareTrigger
  Define OLESSoftwareTrigger for 3
  // AnalogLevelTrigger
  Define OLEAnalogLevelTrigger for 6
Define OLEenumHystIdx for Integer

// Hysteresis0
Define OLEHysteresis0 for 0
// Hysteresis1
Define OLEHysteresis1 for 1
// Hysteresis2
Define OLEHysteresis2 for 2
// Hysteresis3
Define OLEHysteresis3 for 3
// Hysteresis4
Define OLEHysteresis4 for 4
// Hysteresis6
Define OLEHysteresis6 for 5
// Hysteresis9
Define OLEHysteresis9 for 6
// Hysteresis13
Define OLEHysteresis13 for 7
// Hysteresis19
Define OLEHysteresis19 for 8
// Hysteresis28
Define OLEHysteresis28 for 9
// Hysteresis40
Define OLEHysteresis40 for 10
// Hysteresis58
Define OLEHysteresis58 for 11
// Hysteresis84
Define OLEHysteresis84 for 12
// Hysteresis122
Define OLEHysteresis122 for 13
// Hysteresis176
Define OLEHysteresis176 for 14
// Hysteresis255
Define OLEHysteresis255 for 15

// CLSID: {F88080E3-9B4B-11D3-ABFF-0040055DE809}
// Dispatch interface for DataqSdk Control
Class cCom_DDataqSdk is a Mixin

    // Sets or returns the device driver which defines the data acquisition device. You can use the DATAQ Device Driver Wizard shipped with the ActiveX package to set up the driver from outside of the program
    { MethodType=Property DesignTime=False }
    Function ComDeviceDriver Returns String
        String retVal
        Get ComProperty of (phDispatchDriver(Self)) 1 OLE_VT_BSTR to retVal
        Function_Return retVal
    End_Function

    // Sets or returns the device driver which defines the data acquisition device. You can use the DATAQ Device Driver Wizard shipped with the ActiveX package to set up the driver from outside of the program
    { MethodType=Property DesignTime=False }
    Procedure Set ComDeviceDriver String value
        Set ComProperty of (phDispatchDriver(Self)) 1 OLE_VT_BSTR to value
    End_Procedure

    // Sets the device ID if USB or Ethernet option is used, or if the device is used under Windows NT/2000
    { MethodType=Property DesignTime=False }
    Function ComDeviceID Returns String
        String retVal
        Get ComProperty of (phDispatchDriver(Self)) 2 OLE_VT_BSTR to retVal
        Function_Return retVal
    End_Function

    // Sets the device ID if USB or Ethernet option is used, or if the device is used under Windows NT/2000
    { MethodType=Property DesignTime=False }
    Procedure Set ComDeviceID String value
        Set ComProperty of (phDispatchDriver(Self)) 2 OLE_VT_BSTR to value
    End_Procedure

    // Sets or returns the number of channels to be enabled. Channel allocation can be changed via ADChannelList method
    { MethodType=Property DesignTime=False }
Function ComADChannelCount Returns Short
    Short retVal
    Get ComProperty of (phDispatchDriver(Self)) 3 OLE_VT_I2 to retVal
    Function_Return retVal
End_Function

// Sets or returns the number of channels to be enabled. Channel allocation can be changed via ADChannelList method
    { MethodType=Property DesignTime=False }
Procedure Set ComADChannelCount Short value
    Set ComProperty of (phDispatchDriver(Self)) 3 OLE_VT_I2 to value
End_Procedure

// Sets or returns the per-channel sample rate
    { MethodType=Property DesignTime=False }
Function ComSampleRate Returns Real
    Real retVal
    Get ComProperty of (phDispatchDriver(Self)) 4 OLE_VT_R8 to retVal
    Function_Return retVal
End_Function

// Sets or returns the per-channel sample rate
    { MethodType=Property DesignTime=False }
Procedure Set ComSampleRate Real value
    Set ComProperty of (phDispatchDriver(Self)) 4 OLE_VT_R8 to value
End_Procedure

// Sets or returns the maximum burst sample rate from which the sample rate is derived. The property can only be used when the program is running
    { MethodType=Property DesignTime=False }
Function ComMaxBurstRate Returns Real
    Real retVal
    Get ComProperty of (phDispatchDriver(Self)) 5 OLE_VT_R8 to retVal
    Function_Return retVal
End_Function

// Sets or returns the maximum burst sample rate from which the sample rate is derived. The property can only be used when the program is running
    { MethodType=Property DesignTime=False }
Procedure Set ComMaxBurstRate Real value
    Set ComProperty of (phDispatchDriver(Self)) 5 OLE_VT_R8 to value
End_Procedure

// Sets or returns the EventPoint to determine how often NewData event will be fired
    { MethodType=Property DesignTime=False }
Function ComEventPoint Returns Integer
    Integer retVal
    Get ComProperty of (phDispatchDriver(Self)) 6 OLE_VT_I4 to retVal
    Function_Return retVal
End_Function

// Sets or returns the EventPoint to determine how often NewData event will be fired
    { MethodType=Property DesignTime=False }
Procedure Set ComEventPoint Integer value
    Set ComProperty of (phDispatchDriver(Self)) 6 OLE_VT_I4 to value
End_Procedure

// Sets or returns the burst counter to fine tune sample rate. Refer to the SDK manual for more information
    { MethodType=Property DesignTime=False }
Function ComBurstCounter Returns Integer
    Integer retVal
    Get ComProperty of (phDispatchDriver(Self)) 7 OLE_VT_I4 to retVal
    Function_Return retVal
End_Function

// Sets or returns the burst counter to fine tune sample rate. Refer to the SDK manual for more information
    { MethodType=Property DesignTime=False }
Procedure Set ComBurstCounter Integer value
    Set ComProperty of (phDispatchDriver(Self)) 7 OLE_VT_I4 to value
End_Procedure
// Sets or returns Trigger Method. Refer to the SDK manual for more information
{ MethodType=Property DesignTime=False }
Function ComTrigMode Returns OLEenumTrigMode
  OLEenumTrigMode retVal
  Get ComProperty of (phDispatchDriver(Self)) 8 OLE_VT_I4 to retVal
  Function_Return retVal
End_Function

// Sets or returns Trigger Method. Refer to the SDK manual for more information
{ MethodType=Property DesignTime=False }
Procedure Set ComTrigMode OLEenumTrigMode value
  Set ComProperty of (phDispatchDriver(Self)) 8 OLE_VT_I4 to value
End_Procedure

// Sets or returns Trigger level in binary format
{ MethodType=Property DesignTime=False }
Function ComTrigLevel Returns Short
  Short retVal
  Get ComProperty of (phDispatchDriver(Self)) 9 OLE_VT_I2 to retVal
  Function_Return retVal
End_Function

// Sets or returns Trigger level in binary format
{ MethodType=Property DesignTime=False }
Procedure Set ComTrigLevel Short value
  Set ComProperty of (phDispatchDriver(Self)) 9 OLE_VT_I2 to value
End_Procedure

// Sets or returns Trigger slope
{ MethodType=Property DesignTime=False }
Function ComTrigSlope Returns OLEenumTrigSlope
  OLEenumTrigSlope retVal
  Get ComProperty of (phDispatchDriver(Self)) 10 OLE_VT_I4 to retVal
  Function_Return retVal
End_Function

// Sets or returns Trigger slope
{ MethodType=Property DesignTime=False }
Procedure Set ComTrigSlope OLEenumTrigSlope value
  Set ComProperty of (phDispatchDriver(Self)) 10 OLE_VT_I4 to value
End_Procedure

// Sets or returns Triger channel index reference to the channel allocation
{ MethodType=Property DesignTime=False }
Function ComTrigScnChnIdx Returns Short
  Short retVal
  Get ComProperty of (phDispatchDriver(Self)) 11 OLE_VT_I2 to retVal
  Function_Return retVal
End_Function

// Sets or returns Triger channel index reference to the channel allocation
{ MethodType=Property DesignTime=False }
Procedure Set ComTrigScnChnIdx Short value
  Set ComProperty of (phDispatchDriver(Self)) 11 OLE_VT_I2 to value
End_Procedure

// Sets or returns the number of data points to be acquired before the trigger point
{ MethodType=Property DesignTime=False }
Function ComTrigPreLength Returns Short
  Short retVal
  Get ComProperty of (phDispatchDriver(Self)) 12 OLE_VT_I2 to retVal
  Function_Return retVal
End_Function

// Sets or returns the number of data points to be acquired before the trigger point
{ MethodType=Property DesignTime=False }
Procedure Set ComTrigPreLength Short value
  Set ComProperty of (phDispatchDriver(Self)) 12 OLE_VT_I2 to value
End_Procedure
// Sets or returns the number of data points to be acquired after the trigger point
{ MethodType=Property DesignTime=False }
Function ComTrigPostLength Returns Short
    Short retVal
    Get ComProperty of (phDispatchDriver(Self)) 13 OLE_VT_I2 to retVal
    Function_Return retVal
End_Function

// Sets or returns the number of data points to be acquired after the trigger point
{ MethodType=Property DesignTime=False }
Procedure Set ComTrigPostLength Short value
    Set ComProperty of (phDispatchDriver(Self)) 13 OLE_VT_I2 to value
End_Procedure

// Sets or returns the trigger hysteresis index
{ MethodType=Property DesignTime=False }
Function ComTrigHysteresisIdx Returns OLEenumHystIdx
    OLEenumHystIdx retVal
    Get ComProperty of (phDispatchDriver(Self)) 14 OLE_VT_I4 to retVal
    Function_Return retVal
End_Function

// Sets or returns the trigger hysteresis index
{ MethodType=Property DesignTime=False }
Procedure Set ComTrigHysteresisIdx OLEenumHystIdx value
    Set ComProperty of (phDispatchDriver(Self)) 14 OLE_VT_I4 to value
End_Procedure

// Sets the digital output port state, if available. The property can only be used when the program is running
{ MethodType=Property DesignTime=False }
Function ComDigitalOutput Returns Short
    Short retVal
    Get ComProperty of (phDispatchDriver(Self)) 15 OLE_VT_I2 to retVal
    Function_Return retVal
End_Function

// Sets the digital output port state, if available. The property can only be used when the program is running
{ MethodType=Property DesignTime=False }
Procedure Set ComDigitalOutput Short value
    Set ComProperty of (phDispatchDriver(Self)) 15 OLE_VT_I2 to value
End_Procedure

// Returns the number of new samples that are available since the last call to the GetData or GetDataEx method
{ MethodType=Property DesignTime=False }
Function ComAvailableData Returns Integer
    Integer retVal
    Get ComProperty of (phDispatchDriver(Self)) 16 OLE_VT_I4 to retVal
    Function_Return retVal
End_Function

// Returns the number of new samples that are available since the last call to the GetData or GetDataEx method
{ MethodType=Property DesignTime=False }
Procedure Set ComAvailableData Integer value
    Set ComProperty of (phDispatchDriver(Self)) 16 OLE_VT_I4 to value
End_Procedure

// Returns the digital input port state, if available. The property can only be used when the program is running
{ MethodType=Property DesignTime=False }
Function ComDigitalInput Returns Short
    Short retVal
    Get ComProperty of (phDispatchDriver(Self)) 17 OLE_VT_I2 to retVal
    Function_Return retVal
End_Function

// Returns the digital input port state, if available. The property can only be used when the program is running
{ MethodType=Property DesignTime=False }
Procedure Set ComDigitalInput Short value
    Set ComProperty of (phDispatchDriver(Self)) 17 OLE_VT_I2 to value
End_Procedure
// Returns the device model number. The property can only be used when the program is running
{ MethodType=Property DesignTime=False }
Function ComInfoBoardID Returns Short

  Short retVal
  Get ComProperty of (phDispatchDriver(Self)) 18 OLE_VT_I2 to retVal
  Function_Return retVal
End_Function

// Returns the device model number. The property can only be used when the program is running
{ MethodType=Property DesignTime=False }
Procedure Set ComInfoBoardID Short value
  Set ComProperty of (phDispatchDriver(Self)) 18 OLE_VT_I2 to value
End_Procedure

// Returns the device serial number. The property can only be used when the program is running
{ MethodType=Property DesignTime=False }
Function ComInfoSerial Returns String

  String retVal
  Get ComProperty of (phDispatchDriver(Self)) 19 OLE_VT_BSTR to retVal
  Function_Return retVal
End_Function

// Returns the device serial number. The property can only be used when the program is running
{ MethodType=Property DesignTime=False }
Procedure Set ComInfoSerial String value
  Set ComProperty of (phDispatchDriver(Self)) 19 OLE_VT_BSTR to value
End_Procedure

// Returns the gain configuration of the device. The property can only be used when the program is running
{ MethodType=Property DesignTime=False }
Function ComInfoPGL Returns Boolean

  Boolean retVal
  Get ComProperty of (phDispatchDriver(Self)) 20 OLE_VT_BOOL to retVal
  Function_Return retVal
End_Function

// Returns the gain configuration of the device. The property can only be used when the program is running
{ MethodType=Property DesignTime=False }
Procedure Set ComInfoPGL Boolean value
  Set ComProperty of (phDispatchDriver(Self)) 20 OLE_VT_BOOL to value
End_Procedure

// Returns the last error code
{ MethodType=Property DesignTime=False }
Function ComLastErrorCode Returns Integer

  Integer retVal
  Get ComProperty of (phDispatchDriver(Self)) 21 OLE_VT_I4 to retVal
  Function_Return retVal
End_Function

// Returns the last error code
{ MethodType=Property DesignTime=False }
Procedure Set ComLastErrorCode Integer value
  Set ComProperty of (phDispatchDriver(Self)) 21 OLE_VT_I4 to value
End_Procedure

// Returns the rev number of the hardware
{ MethodType=Property DesignTime=False }
Function ComInfoRev Returns Short

  Short retVal
  Get ComProperty of (phDispatchDriver(Self)) 22 OLE_VT_I2 to retVal
  Function_Return retVal
End_Function

// Returns the rev number of the hardware
{ MethodType=Property DesignTime=False }
Procedure Set ComInfoRev Short value
  Set ComProperty of (phDispatchDriver(Self)) 22 OLE_VT_I2 to value
End_Procedure
// Sets or returns the counter used in all the channels in the scan list to fine tune sample rate. Refer to the SDK manual for more information
{ MethodType=Property DesignTime=False }
Function ComADCounter Returns Integer

Integer retVal
Get ComProperty of (phDispatchDriver(Self)) 23 OLE_VT_I4 to retVal
Function_Return retVal
End_Function

// Sets or returns the counter used in all the channels in the scan list to fine tune sample rate. Refer to the SDK manual for more information
{ MethodType=Property DesignTime=False }
Procedure Set ComADCounter Integer value
Set ComProperty of (phDispatchDriver(Self)) 23 OLE_VT_I4 to value
End_Procedure

// Acquires all the available data from all enabled channels and return the result in a Variant
Function ComGetData Returns Variant
Handle hDispatchDriver
Variant retVal
Get phDispatchDriver to hDispatchDriver
Get InvokeComMethod of hDispatchDriver 24 OLE_VT_VARIANT to retVal
Function_Return retVal
End_Function

// Defines the allocation of channels. The default is sequential, incremented
Procedure ComADChannelList Variant llChannel
Handle hDispatchDriver
Get phDispatchDriver to hDispatchDriver
Send PrepareParams to hDispatchDriver 1
Send DefineParam to hDispatchDriver OLE_VT_VARIANT llChannel
Send InvokeComMethod to hDispatchDriver 25 OLE_VT_VOID
End_Procedure

// Defines the counter setting. Refer to SDK manual for more info. If you change SampleRate property, this value will be over written by the driver
Procedure ComADCounterList Variant llCounter
Handle hDispatchDriver
Get phDispatchDriver to hDispatchDriver
Send PrepareParams to hDispatchDriver 1
Send DefineParam to hDispatchDriver OLE_VT_VARIANT llCounter
Send InvokeComMethod to hDispatchDriver 26 OLE_VT_VOID
End_Procedure

// Defines the front-end configuration as single-ended or differential. The default is single-ended
Procedure ComADDiffList Variant llDiff
Handle hDispatchDriver
Get phDispatchDriver to hDispatchDriver
Send PrepareParams to hDispatchDriver 1
Send DefineParam to hDispatchDriver OLE_VT_VARIANT llDiff
Send InvokeComMethod to hDispatchDriver 27 OLE_VT_VOID
End_Procedure

// Defines the channel gain-index setting. The default index is 0, or a gain of 1, for all channels
Procedure ComADGainList Variant llGain
Handle hDispatchDriver
Get phDispatchDriver to hDispatchDriver
Send PrepareParams to hDispatchDriver 1
Send DefineParam to hDispatchDriver OLE_VT_VARIANT llGain
Send InvokeComMethod to hDispatchDriver 28 OLE_VT_VOID
End_Procedure

// Defines the acquisition method for intelligent oversampling. The default is Average
Procedure ComADMethodList Variant llMethod
Handle hDispatchDriver
Get phDispatchDriver to hDispatchDriver
Send PrepareParams to hDispatchDriver 1
Send DefineParam to hDispatchDriver OLE_VT_VARIANT llMethod
Send InvokeComMethod to hDispatchDriver 29 OLE_VT_VOID
End_Procedure

// Acquires specified amount of data and return it via the array
Function ComGetDataEx Short ByRef llBuffer Short llCount Returns Short  
Handle hDispatchDriver  
Short retVal  
Get phDispatchDriver to hDispatchDriver  
Send PrepareParams to hDispatchDriver 2  
Send DefineParam to hDispatchDriver (OLE_VT_BYREF ior OLE_VT_I2) llBuffer  
Send DefineParam to hDispatchDriver OLE_VT_I2 llCount  
Get InvokeComMethod of hDispatchDriver 30 OLE_VT_I2 to retVal  
Function_Return retVal  
End_Function  

// Outputs analog data via DAC port  
Procedure ComDAOutput Short llData Short llPort  
Handle hDispatchDriver  
Get phDispatchDriver to hDispatchDriver  
Send PrepareParams to hDispatchDriver 2  
Send DefineParam to hDispatchDriver OLE_VT_I2 llData  
Send DefineParam to hDispatchDriver OLE_VT_I2 llPort  
Send InvokeComMethod to hDispatchDriver 31 OLE_VT_VOID  
End_Procedure  

// Starts ADC scanning  
Procedure ComStart  
Handle hDispatchDriver  
Get phDispatchDriver to hDispatchDriver  
Send InvokeComMethod to hDispatchDriver 32 OLE_VT_VOID  
End_Procedure  

// Stops ADC scanning  
Procedure ComStop  
Handle hDispatchDriver  
Get phDispatchDriver to hDispatchDriver  
Send InvokeComMethod to hDispatchDriver 33 OLE_VT_VOID  
End_Procedure  

// Grabs the latest specified amount (<32K) of data to the specified array  
Function ComGetDataFrameEx Short ByRef llBuffer Short llCount Returns Short  
Handle hDispatchDriver  
Short retVal  
Get phDispatchDriver to hDispatchDriver  
Send PrepareParams to hDispatchDriver 2  
Send DefineParam to hDispatchDriver (OLE_VT_BYREF ior OLE_VT_I2) llBuffer  
Send DefineParam to hDispatchDriver OLE_VT_I2 llCount  
Get InvokeComMethod of hDispatchDriver 34 OLE_VT_I2 to retVal  
Function_Return retVal  
End_Function  

// Grabs the latest specified amount (<32K) of data and returns in a Variant  
Function ComGetDataFrame Short llCount Returns Variant  
Handle hDispatchDriver  
Variant retVal  
Get phDispatchDriver to hDispatchDriver  
Send PrepareParams to hDispatchDriver 1  
Send DefineParam to hDispatchDriver OLE_VT_I2 llCount  
Get InvokeComMethod of hDispatchDriver 35 OLE_VT_VARIANT to retVal  
Function_Return retVal  
End_Function  

Procedure ComAboutBox  
Handle hDispatchDriver  
Get phDispatchDriver to hDispatchDriver  
Send InvokeComMethod to hDispatchDriver -552 OLE_VT_VOID  
End_Procedure  

End_Class  

// CLSID: {F88080E4-9B4B-11D3-ABFF-0040055DE809}  
// Event interface for DataqSdk Control  
Class cCom_DDataqSdkEvents is a Mixin  

// Fires when the number of available new data points exceeds the number specified in the EventPoint property
MethodType=Event
Procedure OnComNewData Short llCount
End_Procedure

// Fires when the input buffer (around 60,000 samples) is overrun
MethodType=Event
Procedure OnComOverRun
End_Procedure

// Error in the ActiveX or Device driver
MethodType=Event
Procedure OnComControlError Integer llCode
End_Procedure

Visibility=Private
Procedure RegisterComEvents
Send RegisterComEvent 1 msg_OnComNewData
Send RegisterComEvent 2 msg_OnComOverRun
Send RegisterComEvent 3 msg_OnComControlError
End_Procedure
End_Class

// CoClass
// ProgID: DATAQSDK.DataqSdkCtrl.1
// CLSID: {F88080E5-9B4B-11D3-ABFF-0040055DE809}
// DataqSdk Control
Class cComDataqSdk is a cComActiveXControl
Import_Class_Protocol cCom_DDataqSdk
Import_Class_Protocol cCom_DDataqSdkEvents

Procedure Construct_Object
Forward Send Construct_Object
Set psProgID to "{F88080E5-9B4B-11D3-ABFF-0040055DE809}"
Set psEventId to "{F88080E4-9B4B-11D3-ABFF-0040055DE809}"
Set peAutoCreate to acAutoCreate
End_Procedure
End_Class

/**************************************************************************
// Copyright (c) 1997 Data Access Corporation, Miami Florida,
// All rights reserved.
// DataFlex is a registered trademark of Data Access Corporation.
/*
// Module Name:
/*   DFERROR.PKG
**************************************************************************/

// 08/24/94 JJT made DFPanel a tool panel
// 08/30/94 JJT Find past end and beginning now just beep. Change
// in error_report. Support new property to disable this
// Bell on Find Error State (dflts to true)
// 03/01/95 JJT Changed Panel look. Added help support.
// 07/23/96 JJT - New Class names
// 12/10/96 JJT - Added Error_Processing_State so we can know if
// we are within an error. Error_report will check this and
// ignore recursive errors. This is a public message which
// other objects may inquire (NOT set) to see if we are
// within an error condition. Since error uses message
// box it will send notify_focus_change to the focus object
// creating a possibility of error recursion.
//
// 12/10/96 JJT - Removed code which has been commented out.
// 8/5/97 JJT - Created ability to get extra error information from the
// object generating the error. If the global variable
// ghoErrorSource is non-zero then that is the object that
// generated the error. It is expected that this object will
// understand the message: Get Extended Error Message and return
// a string of additional information. This string can be multi
// line with "n" separators. This will be used by DD objects to
// provide additional error file (field and file). Error_report
// is modified to support this.
// 8/19/97 JIT - Zero ghoErrorSource during error_report. We do this because the
// procedure in charge of zeroing this may not get returned after
// and error
// 10/2/98 MG - Changed error_report with additional parameter
//
// Error Class and object
//
// This package describes the Error_Type class and an instance
// Error_Info_Object. The error class allows for an interactive interface
// for all errors generated in a program. The object displays important
// information about the error, and allows the user to get help about the
// error condition.
//
// The programmer may define the error numbers that will be trapped by
// the error object through the use of the Trap_Error, Ignore_Error,
// Trap_All and Ignore_All messages.
//
// The Help buttons has been defined to generate a special help name which
// will allow the user to get help on the specific error condition. The
// help name is made up of the constant ERROR: and the error number. The
// help data files should contain help subjects for any system errors or
// program defined conditions.
//
Use LanguageText.pkg

#CHKSUB 1 1 // Verify the UI subsystem.

use Windows.pkg
use msgbox.pkg
Use GlobalFunctionsProcedures.pkg

//integer ghoErrorSource
//Move 0 to ghoErrorSource
//ghoErrorSource object is expected to support this message
//Register Function Extended_Error_Message returns string

// Include or define all useful symbols.
#include ERRORNUM.INC
#REPLACE MAX_ERROR_NUMBER 32766

// This array stores the set of trapped errors as toggled ranges starting
// with the errors that are trapped. The array should always contain 0 and
// MAX_ERROR_NUMBER + 1, which are the limits. If an array contained the
// following items...
//
// { 0, 5, 10, MAX_ERROR_NUMBER + 1 }.
//
// This would mean that errors 1 - 4 are trapped, 5 - 9 are ignored, and
// 10 through the rest are trapped.
//
// { Visibility=Private }
class Trapped_Errors_Array is an array

// Find largest error LE targetError. Assumes array is sorted.
function findErrorLE integer targetError returns integer
Integer lowIndex hiIndex midIndex currError

// If error is outside of boudary conditions, use
// value of closest valid error# instead.
if targetError le 0;
  move 1 to targetError
else if targetError ge MAX_ERROR_NUMBER ;
  move ( MAX_ERROR_NUMBER - 1 ) to targetError
//
move 0 to lowIndex
move ( item_count( self ) - 1 ) to hiIndex
// midIndex will contain the closest error LE to target upon exit.
repeat
  move ( ( lowIndex + hiIndex ) / 2 ) to midIndex
  move ( integer_value( self, midIndex ) ) to currError

  // midIndex is targetIndex if a match occurs
  if currError eq targetError ;
    function_return midIndex
  end

  // We are either on it or just below it.
  if ( lowIndex eq midIndex ) begin
    if ( integer_value( self, hiIndex ) le targetError ) ;
      move hiIndex to midIndex
    function_return midIndex
  end

  // No match, so move the boundaries.
  if currError gt targetError ;
    move ( midIndex - 1 ) to hiIndex
  else ;
    move midIndex to lowIndex
  end

until lowIndex gt hiIndex

function_return midIndex
end_function

// Boundaries of the table are assumed to hold error limits.
procedure initArray
  send delete_data
  set array_value item 0 to 0
  set array_value item 1 to ( MAX_ERROR_NUMBER + 1 )
end_procedure

// Return 1 if Error is trapped, 0 otherwise.
function isTrapped integer Error# returns integer
  function_return ( NOT ( mod( findErrorLE( self, Error# ), 2 ) ) )
end_function

// Add the error as long as it doesn't violate boundary conditions.
// This routine leaves the array unsorted.
procedure addError integer Error#
  if ( ( Error# lt MAX_ERROR_NUMBER ) AND ( Error# gt 0 ) ) ;
    set array_value item ( item_count( self ) ) to ( INTEGER( Error# ) )
  end_procedure

// Set error to flagged state.
procedure handleError integer Error# integer trapFlag
  integer prevErrIndex prevErrFlag prevErrValue nextErrValue
  if ( ( Error# gt MAX_ERROR_NUMBER ) OR ( Error# lt 0 ) ) begin
    error DFERR_ERROR_NUMBER_OUT_OF_RANGE
    procedure_return
  end

  get findErrorLE Error# to prevErrIndex
  get isTrapped   Error# to prevErrFlag

  // if eq, Error already handled in some range.
  if prevErrFlag NE trapFlag Begin
    // This is kind of complicated. If we are adding an error,
    // we have to account for the error already being in the
// array as well as rejoining ranges that have been previously
// split and splitting ranges when adding a new flag.

get integer_value item ( prevErrIndex + 1 ) to nextErrValue
get integer_value item prevErrIndex to prevErrValue

// Do this first so prevErrIndex stays valid.
if nextErrValue eq ( Error# + 1 ) ;
   send delete_item ( prevErrIndex + 1 )
else ;
   send addError ( Error# + 1 )

if ( prevErrValue lt Error# ) ;
   send addError Error#
else ;
   send delete_item prevErrIndex
End
send sort_items UPWARD_DIRECTION

end_procedure

//*** Flag error as trappable
procedure Trap_Error integer Error#
   send handleError Error# 1
end_procedure

//*** Flag error as non-trappable
procedure Ignore_Error integer Error#
   send handleError Error# 0
end_procedure

//*** Flag all errors as trappable
procedure Trap_All
   send initArray
end_procedure

//*** Flag all errors as non-trappable
procedure Ignore_All
   send delete_data
   set array_value item 0 to 0
   set array_value item 1 to 1
   set array_value item 2 to ( MAX_ERROR_NUMBER + 1 )
end_procedure

end_class

{ HelpTopic=ErrorSystem }
class ErrorSystem is a cObject

procedure construct_object
   forward send construct_object

   set delegation_mode to no_delegate_or_error

   { Category="Error Handling" }
   Property Integer Verbose_State True
   { DesignTime=False }
   Property Integer Current_Error_Number 0
   { DesignTime=False }
   Property Integer Error_Line_Number 0
   { DesignTime=False }

   // Flag which is sent when error is being processed. This
   // stops error recursion.
   { Visibility=Private }
   Property Integer Error_Processing_State False

   // This allows us to skip find errors (GT & LT) and to only
   // ring a bell when these occur.
   //
Property Integer Bell_on_Find_Error_State TRUE

object trappedErrors is a Trapped_Errors_Array
    send initArray
end_object

send Trap_All

move self to Error_Object_Id
end_procedure

Function Help_Context Integer Context_Type Returns String
    Function_Return (Current_Error_Number(self))
End_Function

//*** Catch and display error Error#.
procedure Trap_Error integer Error#
    send Trap_Error to ( trappedErrors( self ) ) Error#
end_procedure

//*** Pass error Error# on to the regular DataFlex error handler.
procedure Ignore_Error integer Error#
    send Ignore_Error to ( trappedErrors( self ) ) Error#
end_procedure

//*** Catch and display all errors.
procedure Trap_All
    send Trap_All to ( trappedErrors( self ) )
end_procedure

//*** Forward all error to regular DataFlex error handler.
procedure Ignore_All
    send Ignore_All to ( trappedErrors( self ) )
end_procedure

//*** Catch and display all errors.
procedure Trap_Error integer Error#
    send Trap_Error to ( trappedErrors( self ) ) Error#
end_procedure

//*** Forward all error to regular DataFlex error handler.
procedure Ignore_All
    send Ignore_All to ( trappedErrors( self ) )
end_procedure

//*** Build complete error description from Flexerrs and user error message.
{ Visibility=Private }
function Error_Description integer Error# string ErrMsg returns string
    string Full_Error_Text
    trim ErrMsg to ErrMsg
    move (trim(error_text(DESKTOP,Error#))) to Full_Error_Text
    if (ErrMsg<>"") begin
        if ( ( Full_Error_Text<>"" ) AND ;
            error_text_available( DESKTOP, Error# ) ) Begin
            // Make sure last character of error text is a separating symbol.
            // if not, add a "." So we have format of "error-text. error-detail"
            If ( pos(right(Full_Error_Text,1),",;:;")=0 ) ;
            Move (Full_Error_Text - ".") to Full_Error_Text
            Move (Full_Error_Text * ErrMsg) to Full_Error_Text
        end
        else ;
        move ErrMsg to Full_Error_Text
    end
    function_return Full_Error_Text
end_function

//*** return true if an error number is critical
{ Visibility=Private }
function Is_Critical integer Error# returns integer
    function_return ("3.10.18.19.20.21.22.43.70.72.74.75.78.80.97.",
        contains(","+string(Error#)+"."))
end_function
/*** Handle error event, displaying error info to user.  
{ MethodType=Event }  
procedure Error_Report integer ErrNum integer Err_Line stringErrMsg  
integer Text_Obj# rval icon# critical Foc oldco  
string Temp errorText sMess  
string sSource  
Integer iSrcPos iSrc iTxtLen  
If (Error_processing_State(self)) ; // don't allow error  
Procedure_return                           // recursion  
Set Error_Processing_State to TRUE // we are now in an error reporting state  
Set Current_Error_Number to ErrNum  
Set Error_Line_Number to Err_Line  
//  
// Changes made by JJT so find errors don't report - just beep  
//  
#REPLACE FIND_PAST_END 42  
#REPLACE FIND_PAST_BEGIN 41  
If ( Bell_On_find_Error_State(self) AND ;  
ErrNum=FIND_PAST_END or ErrNum=FIND_PAST_BEGIN) Begin  
Send Bell  
End  
Else if NOT ( isTrapped( TrappedErrors( self ), ErrNum )) begin  
// if trapped do nothing  
// We used to forward send. Since this is based on array, it does not understand this message, the forward was  
// not understood. Since arrays don't delegate or error, nothing happened.  
// An easier way to do nothing, is to do nothing, hence this line if removed  
//forward send Error_Report ErrNum Err_LineErrMsg  
end  
Else Begin  
// See if source information is provided (Source = module.function). If so remove  
// as detail. Must find last instance of this in string  
Move (pos(C_ErrorContextSourceText,ErrMsg)) to iSrc  
If iSrc Begin  
Move (iSrc-1) to iSrcPos  
Move (length(C_ErrorContextSourceText)) to iTxtLen  
Move ErrMsg to sSource  
Repeat // this makes sure we find last instance of this  
Move (remove(sSource, 1, iSrc-1 + iTxtLen )) to sSource // right part of string  
Move (pos(C_ErrorContextSourceText,sSource)) to iSrc // see if it was the last  
If iSrc = ; // if not, track length  
Move (iSrcPos + iTxtLen + iSrc-1) to iSrcPos  
Until (iSrc=0)  
Move (trim(left(ErrMsg,iSrcPos))) toErrMsg  
If (right(ErrMsg,1)=');  
Move (left(ErrMsg,length(ErrMsg)-1)) toErrMsg  
end  
Get Error_Description ErrNumErrMsg to errorText  
// if the error source is identified we can get extended error  
// text for our error message  
if ghoErrorSource Begin  
get extended_error_Message of ghoErrorSource to sMess  
if sMess ne " ;  
Move (ErrorText + "\n\n" + sMess ) to ErrorText  
End  
if (Verbose_State(self)) begin  
Move ( ErrorText + "\n\n" + SFormat(C_$TechnicalDetails, ErrNum, Err_Line) ) to ErrorText  
If (sSource<>"\n";  
Move (ErrorText + "\n" + C_$ErrorSource +"\n" * sSource) to ErrorText  
end  
Get Is_Critical errmsg to critical  
Move (if(critical,MB_IconHand,MB_IconExclamation)) to icon#
Move self to oldco
Get Focus of desktop to foc
If Foc gt desktop Move foc to self
Get Message_Box Errortext C\_ERROR MB\_Ok icon# to rval
Move Oldco to self
// abort on critical errors
if Critical abort
End
Move 0 to ghoErrorSource
Set Error\_Processing\_State to FALSE // no longer reporting an error
end_procedure

// JJT- Note if you are using the WINDAF windows help system
// the following functions are not used.

// The functions below are used to construct a general help
// name for errors that are generated by the system. If processing
// comes here, then there was no module specific help found. These
// functions will provide a more general help name that appears in
// the form of SYSTEM..ERROR:#. All global errors should be
// places in the help file under this application and module name.

/*** Returns "ERROR:erronum" to supply error help.***/
{ MethodType=Property Visibility=Private Obsolete=True }
function Help\_Name returns string
  function_return (Append("ERROR: ",lastErr))
end_function

{ MethodType=Property Visibility=Private Obsolete=True }
function Application\_Name returns string
  function_return 'SYSTEM'
end_function

{ MethodType=Property Visibility=Private }
function Module\_Name returns string
  function_return "
end_function
end_class

object Error\_Info\_Object is a ErrorSystem
end_object