Communication With External devices & Applications

DIO Function:

Using Digital I/O

The Spider-81 has an 8-output and 8-input Digital I/O design. The Spider-81B also has this same design but with reduced numbers of inputs and outputs. The 12V power supply pin is also removed for the Spider-81B. The Spider-80X has an interchangeable 4-output and 4-input Digital I/O design. This allows an input to flip to an output, or vice versa.

If a system is composed of multiple Spider-81 modules, only the master module can use the Digital I/O features. Each Spider-81 has 8 isolated digital inputs and 8 digital outputs, corresponding to the pins on the Digital I/O connector. A digital input is detected when a low-high-low voltage change occurs, which triggers the event actions set in the **Event Action Rules** section of the Test Configuration window. Any device that can output square voltage pulses can communicate with the data acquisition/controller system.

Event Actions Rules in DIO

A digital output is sent when a digital output action is triggered by an event in **Event Action Rules**. The output signal can be set as a pulse or a step in the negative or positive direction. Any device that can read such a voltage signal can then be controlled by the data acquisition/controller system.

To associate a system event or a user event to the **Send Digital Output Signals** action, refer to **Event Action Rules** chapter for more details.

Sending Digital Output Before Run / After Run

This function is designed for the customers who want to turn on certain hardware devices, such as power amplifier of the shaker, before the controller enters its test mode. It can be found under **Test parameters** > **Advanced Settings**, **Digital I/O** tab

In order to streamline the shaker test process, the controller can send a digital output before a test is running, even before pre-test starts. The feature gives a possibility to turn on or get ready a necessary external device before pre-test starts.

The configuration of **Send Digital Output Before Run** resides in **Test parameters** > **Advanced Settings** > **DIO**.

Send Digital Output Before Run: enable this feature.

Delay time: wait for the given time duration after the digital output signal is sent.

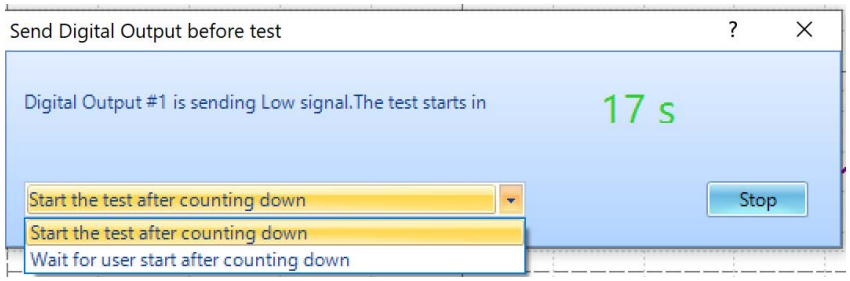
Digital Output channel: sends the signal through the digital output channel.

Signal pattern: assigns the pattern of the digital output signal. It can be High, Low, High-Low-High, or Low-High-Low.

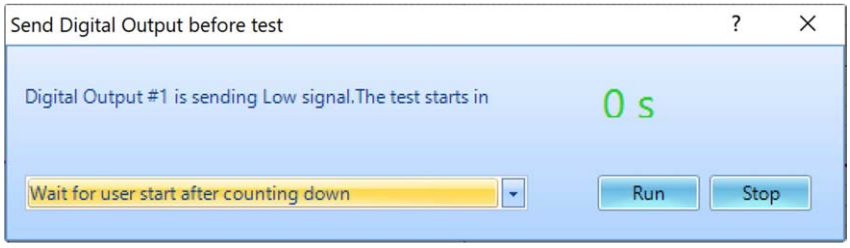
Pulse duration: for signal patterns High-Low-High and Low-High-Low only. Specifies the length of the state in the middle.

After pressing the Start button on the Spider hardware or on the EDM control panel, the controller initializes the test, send digital output, and pops up the following dialog box to count down.

When “Start the test after counting down” is selected, pre-test starts immediately at 0s.



If you need more time before starting pre-test, select “**Wait for user start after counting down**” before 0s. The software will wait for the user’s command to run or stop the test.



Receiving Digital Input to Start Test

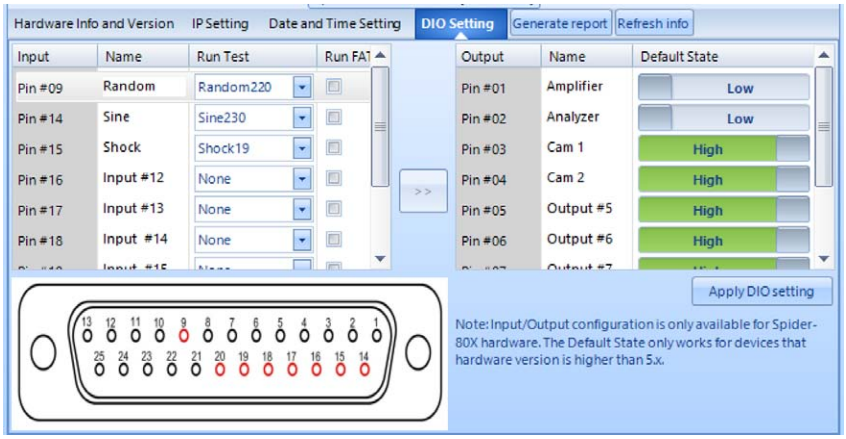
The Spider controller can receive a digital input signal to set off a selected test. The digital input signal given to the controller can be from a third-party equipment automatically or manually.

Before doing so, go to **Tools > Spider Configuration** and select the Spider system to begin configuration.

Configurations of digital input channels are on the left. Find a digital input pin, go to the **Run Test** column, select a test from a dropdown menu to assign the test to the digital input to start.

Click "**Apply DIO setting**" after all changes are done.

In the following screenshot, the digital input pin #09, #14, #15 are configured to start tests Random230, Sine230, and Shock19.



Regardless of the current test, connect EDM to the controller and Control Panel shows Online. Send the signal to the digital input channel and the corresponding test starts.

CAN Bus:

CAN Bus Interface in EDM

The CAN bus (Controller Area Network bus) protocol is a peer-to-peer communication standard for automotive devices without relying on a complicated central computer. It is configured with two wires (CAN high and CAN low) which vary in voltage to communicate a bit series of 1's and 0's. CAN bus was initially designed in 1990, with the ISO standard released in 1993.

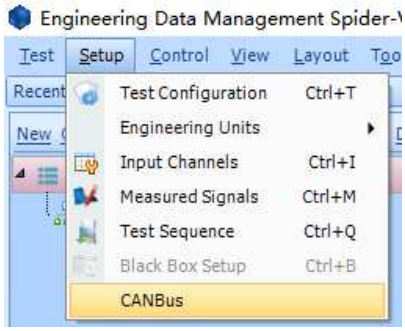
Using CAN bus, an electronic car component (ex: car battery, engine control unit, etc.) can communicate any arbitrary data such as its temperature or working status. A DBC file is used to encode and decode the 1's and 0's into meaningful data, which can be a different mapping for each customer.

EDM supports integration with CAN bus signals for monitoring and Alarm / Abort purposes. Given a customer's DBC file and custom-built Crystal Instruments USB CAN adapter, CAN bus alarm and abort rules can be configured during a vibration test. For example, a user running vibration tests on an EV battery can now configure EDM to monitor the battery's temperature and stop or pause the test when the temperature matches or exceeds a particular value.

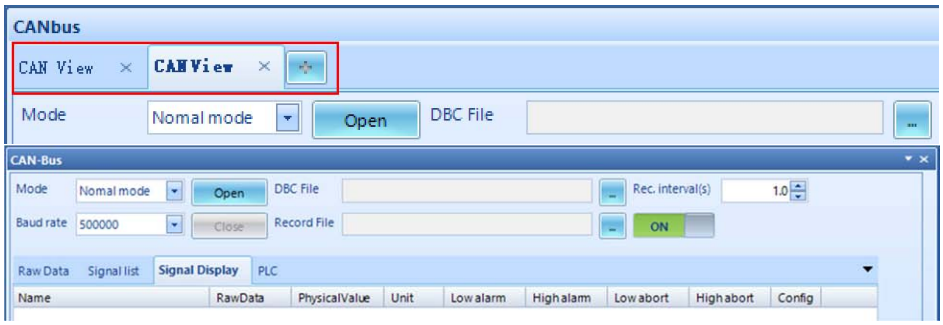
In terms of USB hardware, EDM supports the custom-built Crystal Instruments USB CAN adapter as well as the PCAN-USB adapter developed by Peak Systems.

Configuring CAN Bus on EDM

Click on **Setup > CANBus** to open the **CAN-Bus** window, which contains all settings and functionality related to CAN bus.



A single CAN bus adapter may contain multiple CAN bus channels. The "+" button on the top creates a CAN bus setting page for each CAN bus channel.



There are several modes that can be used for the CAN-Bus window

Normal mode: allows for both sending and receiving CAN bus signals. Used when connecting the Crystal Instruments CAN-Bus USB adapter to the PC running EDM. If the USB adapter is installed, you should be able to click **Open** to locate the DBC file in the filesystem

Listen only mode: a similar mode as "Normal mode" except only CAN-Bus signals can be listened to (no sending allowed).

Self-test mode: used when connecting the Crystal Instruments CAN-Bus USB adapter without a CAN-bus node on the other end. Provides a convenient way to "self-test" certain CAN-Bus signals arriving over the wire.

PCAN-USB: used when connecting the Peak Systems PCAN-USB adapter to the PC running EDM. If the USB adapter is installed, you should be able to click **Open** to locate the DBC file in the filesystem.

Other settings:

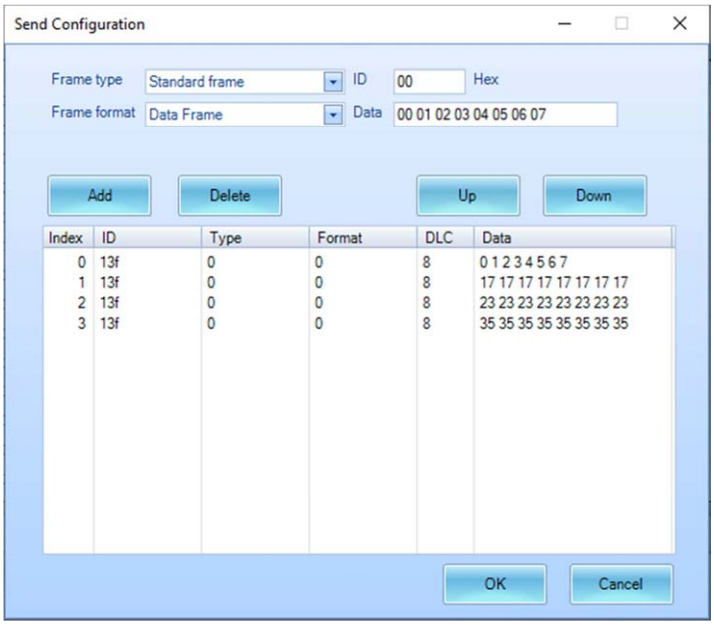
Baud rate: the rate at which data is transmitted over the network. Must match the Baud rate of the external CAN bus device.

DBC File: the .DBC file that describes the data transmitted over CAN bus. Necessary to decode the bits into meaningful information.

Record File: specifies the file location for storing recorded CAN bus data (can be turned ON or OFF)

Sending Data in CAN Bus

Below the **Raw Data** tab is a configuration for sending CAN-Bus data. Click on **Config** to open the "Send Configuration" window



Frames of CAN bus data can be configured with the desired ID (in Hexadecimal) and Data body.

Frame type: supports “Standard frame” and “Extended frame”

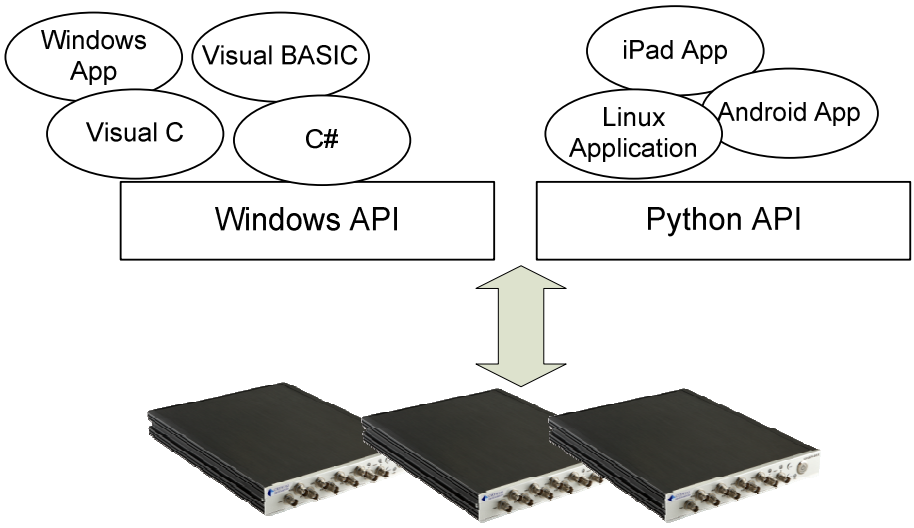
Frame format: supports “Data Frame” and “Remote frame”

API

Crystal Instruments Spider Application Programming Interface (API) is a collection of Windows Dynamic-Linked Libraries (DLL) or Python API providing external applications an easy interface to access and control the Spider-80/80X hardware.

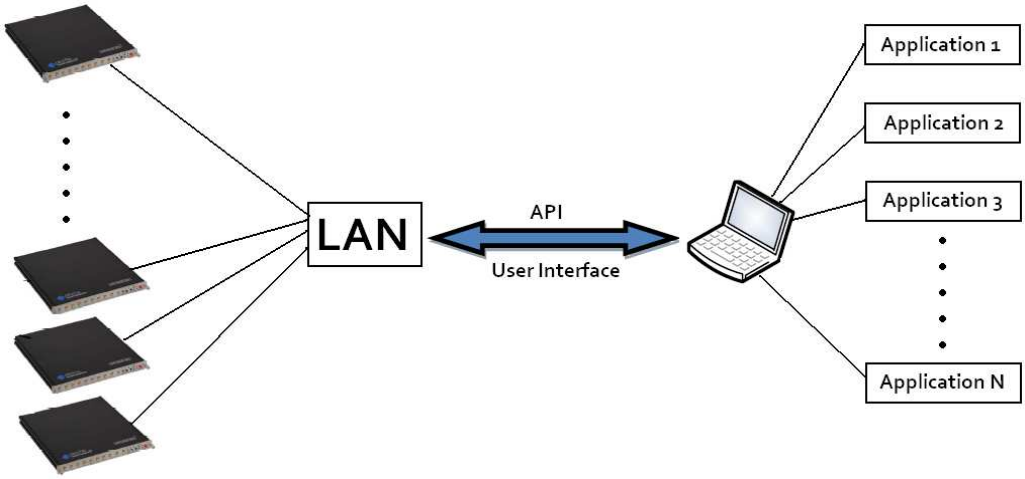
If Windows OS is used, the user can develop their own application in languages of Windows App, VC, VB or C#; if Linux, iOS or Android are used, the Python API can be used as interface.

The Spider API defines a set of command systems based on character strings. This implementation is widely compatible with various connection tools such as APIs, scripts, socket messages and handheld devices, facilitating future technical support.



For over 15 years Crystal Instruments has been dedicated to researching, developing, and producing numerous high quality hardware solutions utilizing advanced DSP technologies. This offers superior real-time data processing. The Spider hardware is a modular, truly distributed, scalable dynamic measurement system, making it ideal for a wide range of industries including machine condition monitoring, automotive, aviation, aerospace, electronics and military use demanding easy, quick and accurate data recording, real-time signal processing and vibration control.

Spider API assists users in developing their own customized and industry-specific applications. Crystal Instruments offers advanced hardware and API, allowing users to focus on their own user interface. While the Spider system is running, the user can access the signal data in real-time using the API. The signals include both time and frequency domain data. The user can also initiate long time history recording. After the test, the time recording can be downloaded to PC using API function calls.



The Spider Windows API was developed using Microsoft Visual Studio technologies as well as Python. It provides a high level, user-friendly interface to the most common development tools within the Windows environment, such as Microsoft Visual C++/C#/Basic. As long as the programming languages support the call of dynamic-linked library (DLL) or Python, users can utilize Spider API to develop their own applications. DLL gives users easy function calls to send commands to the Spider hardware to set up the front end, control the acquisition of data, check the status of the processor, and access DSP processed time and frequency data.

Socket Messages

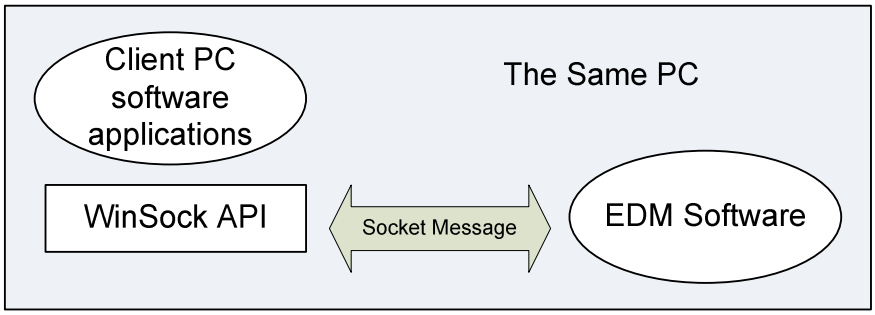
Communicate with EDM Using Socket Messages

EDM, Crystal Instruments' desktop software, monitors, and controls Spider systems through a network connection. EDM can establish and receive socket connections from other application processes as well. These applications can control the operation of tests run through EDM, and can also receive messages and be controlled from EDM. For example, EDM can interface with temperature chamber software or shaker amplifier software running on the same computer or on another device across the network.

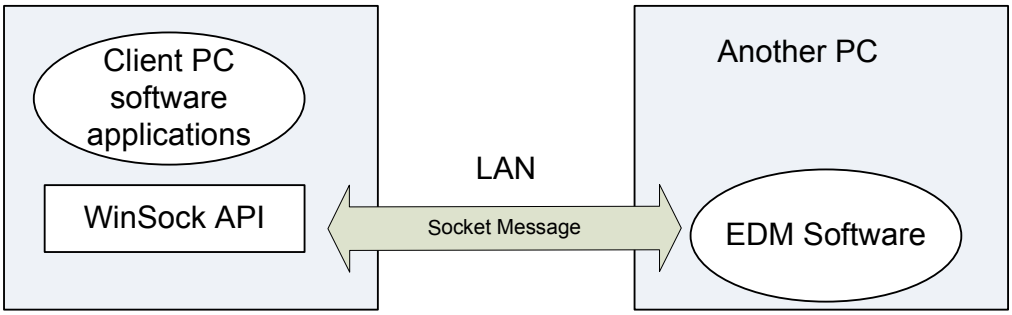
Using the Event Action Rules in EDM, it is easy to customize when EDM sends messages and how EDM responds to incoming messages. Any message can be sent in response to test events, or at a set time in the Run Schedule. EDM also has a set of standard messages it will respond to that can be used by third party software to read EDM information and to manipulate EDM to control Spider systems.

Socket messages are implemented as data structures. EDM acts as a socket server and listens for incoming commands. In addition, EDM can establish socket connections to client programs and send out user-defined commands at instances defined in the run schedule.

The client programs running on a PC communicate with the Ethernet socket layer through Winsock APIs. Most programming platforms, including C#, VC++, and VB, provide direct function calls to Winsock API.
For example, an amplifier controller software running on a PC makes function calls to Winsock. These Winsock functions are included in the built-in libraries in the Windows operating systems.



The Client software can run on either the same PC as the EDM server or on a different PC as long as they can establish socket connections.



EDM Sends Socket Messages

Besides responding to socket messages from a client program, EDM can actively sends socket messages to a client program by utilizing certain Event Action Rules.
In the event action rules configuration of a test on EDM, the action “Send Socket Message” should be added to a system event or a user defined event.
When the event occurs and triggers a socket message action, EDM will send the corresponding character string to the socket port. The client application receives the message and can be programmed to respond to that message.
System events are commonly generated by the DSP program on the Spider. While running a vibration control test, EDM and DSP have direct and constant communication. If a signal exceeds the specified limits, the DSP aborts the test and notifies EDM. EDM will then respond to the event and send the socket messages to the client program.

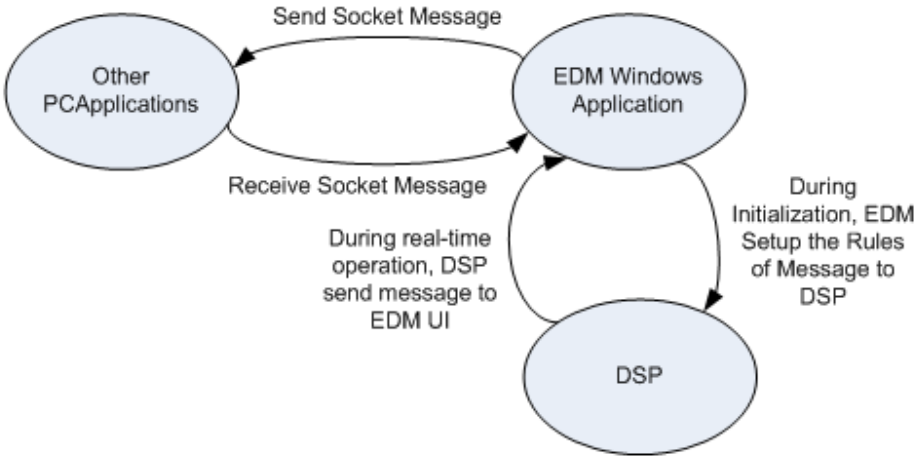


Figure 6. Status Machine