DataAcq SDK
User’s Manual
Table of Contents

About this Manual .................................................. 9
Intended Audience .................................................. 9
What You Should Learn from this Manual ....................... 9
Organization of this Manual ....................................... 10
Conventions Used in this Manual ................................. 10
Related Information ................................................. 11
Where to Get Help ................................................... 11

Chapter 1: Getting Started ............................... 13
What is the DataAcq SDK? ....................................... 14
Quick Start .......................................................... 15
 What You Need ..................................................... 15
 Installing the Software .......................................... 16
Using the DataAcq SDK Online Help ....................... 16
About the Example Programs .................................. 17
About the Library Function Calling Conventions .......... 19

Chapter 2: Function Summary ....................... 21
Data Acquisition Functions .................................... 22
 Information Functions ............................................ 22
 Subsystem Capability Queries ................................. 24
 Initialization and Termination Functions .................... 35
 Configuration Functions ........................................ 36
 Operation Functions ............................................. 43
 Data Conversion Functions .................................. 47
 Data Management Functions ................................ 48
 Buffer Management Functions .............................. 48
 Buffer List Management Functions ....................... 50
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuous Operations</td>
<td>75</td>
</tr>
<tr>
<td>Continuous Post-Trigger Mode</td>
<td>76</td>
</tr>
<tr>
<td>Continuous Pre-Trigger Mode</td>
<td>78</td>
</tr>
<tr>
<td>Continuous About-Trigger Mode</td>
<td>79</td>
</tr>
<tr>
<td>Triggered Scan Mode</td>
<td>81</td>
</tr>
<tr>
<td>Scan-Per-Trigger Mode</td>
<td>82</td>
</tr>
<tr>
<td>Internal Retrigger Mode</td>
<td>83</td>
</tr>
<tr>
<td>Retrigger Extra Mode</td>
<td>84</td>
</tr>
<tr>
<td>Clock Sources</td>
<td>85</td>
</tr>
<tr>
<td>Internal Clock Source</td>
<td>85</td>
</tr>
<tr>
<td>External Clock Source</td>
<td>86</td>
</tr>
<tr>
<td>Extra Clock Source</td>
<td>87</td>
</tr>
<tr>
<td>Trigger Sources</td>
<td>88</td>
</tr>
<tr>
<td>Software (Internal) Trigger Source</td>
<td>88</td>
</tr>
<tr>
<td>External Digital (TTL) Trigger Source</td>
<td>89</td>
</tr>
<tr>
<td>External Analog Threshold (Positive) Trigger Source</td>
<td>89</td>
</tr>
<tr>
<td>External Analog Threshold (Negative) Trigger Source</td>
<td>90</td>
</tr>
<tr>
<td>Analog Event Trigger Source</td>
<td>90</td>
</tr>
<tr>
<td>Digital Event Trigger Source</td>
<td>90</td>
</tr>
<tr>
<td>Timer Event Trigger Source</td>
<td>91</td>
</tr>
<tr>
<td>Extra Trigger Source</td>
<td>91</td>
</tr>
<tr>
<td>Coupling Type</td>
<td>91</td>
</tr>
<tr>
<td>Excitation Current Sources and Values</td>
<td>92</td>
</tr>
<tr>
<td>Thermocouples</td>
<td>92</td>
</tr>
<tr>
<td>Thermocouple Input Types</td>
<td>93</td>
</tr>
<tr>
<td>Thermocouple Correction and Linearization</td>
<td>94</td>
</tr>
<tr>
<td>Buffers</td>
<td>95</td>
</tr>
<tr>
<td>Ready Queue</td>
<td>96</td>
</tr>
<tr>
<td>Inprocess Queue</td>
<td>97</td>
</tr>
<tr>
<td>Done Queue</td>
<td>99</td>
</tr>
</tbody>
</table>
Contents

Buffer and Queue Management .......................... 101
Buffer Wrap Modes ........................................ 102
DMA and Interrupt Resources .......................... 104
Counter/Timer Operations ............................... 106
User Counter/Timers ...................................... 106
Counter/Timer Operation Mode ......................... 107
  Event Counting .......................................... 108
  Up/Down Counting ...................................... 109
  Frequency Measurement ............................... 111
    Using the Windows Timer ............................ 111
    Using a Pulse of a Known Duration ................. 112
Edge-to-Edge Measurement ............................. 115
Continuous Edge-to-Edge Measurement ................. 117
Rate Generation ......................................... 120
One-Shot ................................................ 123
Repetitive One-Shot .................................... 126
C/T Clock Sources ....................................... 128
  Internal C/T Clock .................................... 129
  External C/T Clock .................................... 129
  Internally Cascaded Clock ............................ 130
  Extra C/T Clock Source .............................. 131
Gate Types ............................................. 131
  Software Gate Type .................................. 132
  High-Level Gate Type ................................ 132
  Low-Level Gate Type ................................ 132
  Low-Edge Gate Type ................................ 133
  High-Edge Gate Type ................................ 133
  Any Level Gate Type ................................ 133
  High-Level, Debounced Gate Type .................... 134
  Low-Level, Debounced Gate Type ..................... 134
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release the Subsystem and Terminate the Session.</td>
<td>192</td>
</tr>
<tr>
<td>Handle Errors</td>
<td>192</td>
</tr>
<tr>
<td>Continuous Analog Input</td>
<td>193</td>
</tr>
<tr>
<td>Declare Variables and User Functions</td>
<td>193</td>
</tr>
<tr>
<td>Initialize the Driver</td>
<td>195</td>
</tr>
<tr>
<td>Get a Handle to the Subsystem</td>
<td>196</td>
</tr>
<tr>
<td>Set the DataFlow to Continuous</td>
<td>196</td>
</tr>
<tr>
<td>Specify the Channel List and Channel Parameters</td>
<td>196</td>
</tr>
<tr>
<td>Specify the Clocks</td>
<td>197</td>
</tr>
<tr>
<td>Specify DMA Usage</td>
<td>198</td>
</tr>
<tr>
<td>Set Up Window Handle and Buffering</td>
<td>198</td>
</tr>
<tr>
<td>Configure the Subsystem</td>
<td>199</td>
</tr>
<tr>
<td>Start the Continuous Analog Input Operation</td>
<td>199</td>
</tr>
<tr>
<td>Deal with Messages and Buffers</td>
<td>199</td>
</tr>
<tr>
<td>Convert Values to Voltage</td>
<td>202</td>
</tr>
<tr>
<td>Clean Up</td>
<td>203</td>
</tr>
<tr>
<td>Handle Errors</td>
<td>203</td>
</tr>
</tbody>
</table>

**Index** ........................................................................... 205
About this Manual

This manual describes how to get started using the DataAcq SDK™ (Software Development Kit) to develop application programs for data acquisition devices that conform to the DT-Open Layers™ standard.

Intended Audience

This document is intended for engineers, scientists, technicians, OEMs, system integrators, or others responsible for developing application programs using Microsoft® Developer’s Studio (version 6.0 and higher) to perform data acquisition operations.

It is assumed that you are a proficient programmer, that you are experienced programming in the Microsoft® Windows® 2000, Windows XP, or Windows Vista™ operating environment on the PC or compatible computer platform, and that you have familiarity with data acquisition principles and the requirements of your application.

What You Should Learn from this Manual

This manual summarizes the functions provided by the DataAcq SDK, and describes how to use the functions to develop a data acquisition program. Using this manual, you should be able to successfully install the DataAcq SDK and get started writing an application program for data acquisition.

This manual is intended to be used with the online help for the DataAcq SDK, which you can find in the same program group as the DataAcq SDK software. The online help for the DataAcq SDK contains all of the specific reference information for each of the functions, error codes, and Windows messages.
Organization of this Manual

This manual is organized as follows:

- **Chapter 1, “Getting Started,”** tells how to install the DataAcq SDK.

- **Chapter 2, “Function Summary,”** summarizes the functions provided in the DataAcq SDK.

- **Chapter 3, “Using the DataAcq SDK,”** describes the operations that you can perform using the DataAcq SDK.

- **Chapter 4, “Programming Flowcharts,”** provides programming flowcharts for using the functions provided in the DataAcq SDK.

- **Chapter 5, “Product Support,”** describes how to get help if you have trouble using the DataAcq SDK.

- **Appendix A, “Sample Code,”** provides code fragments that illustrate the use of the functions in the DataAcq SDK.

- An index completes this manual.

Conventions Used in this Manual

The following conventions are used in this manual:

- Notes provide useful information that requires special emphasis, cautions provide information to help you avoid losing data or damaging your equipment, and warnings provide information to help you avoid catastrophic damage to yourself or your equipment.

- Items that you select or type are shown in **bold**. Function names also appear in bold.

- Code fragments are shown in **courier font**.
Related Information

Refer to the following documentation for more information on using the DataAcq SDK:

- DataAcq SDK Online Help. This Windows help file is located in the same program group as the DataAcq SDK software and contains all of the specific reference information for each of the functions, error codes, and Windows messages provided by the DataAcq SDK. Refer to page 16 for information on how to open this help file.

- Device-specific manuals that describe how to get started using your device, the features of the device, and the capabilities supported by the driver for the device. These manuals are on your Data Acquisition OMNI CD™ CD.


Where to Get Help

Should you run into problems installing or using the DataAcq SDK, our Technical Support Department is available to provide prompt, technical assistance. Refer to Chapter 5 for more information. If you are outside the U.S. or Canada, call your local distributor; see our web site (www.datatranslation.com) for the name and telephone number of your nearest distributor.
Getting Started

What is the DataAcq SDK? ........................................... 14
Quick Start .......................................................... 15
Using the DataAcq SDK Online Help ............................ 16
About the Example Programs ................................. 17
About the Library Function Calling Conventions .......... 19
What is the DataAcq SDK?

The DataAcq SDK is a programmer’s DLL (Dynamic Linked Library) that supports Data Translation’s most popular data acquisition devices under Microsoft Windows 2000, Windows XP, and Windows Vista.

The DataAcq SDK functions are fully compatible with DT-Open Layers™, which is a set of standards for developing integrated, modular application programs under Windows.

Because DT-Open Layers is modular and uses Windows DLLs, you can add support for a new data acquisition device at any time. Just add the new DT-Open Layers device driver, modify your code to incorporate the features of the new device, and then recompile the code. All calls to DataAcq SDK functions currently in your application program can remain untouched.

The list of supported data acquisition devices is constantly expanding. For the most up-to-date information, refer to the Data Translation web site (www.datatranslation.com).
Quick Start

The following is an overview of the tasks required to install and use the DataAcq SDK:

1. Make sure that your system meets the requirements for installing the DataAcq SDK. For more information, refer to the next section.

2. Install the DataAcq SDK. For more information, refer to page 16.

3. Create your application program. For more information, refer to the Microsoft Developer’s Studio documentation and to the sample code provided in Appendix A starting on page 187.

If you have problems installing or using the DataAcq SDK, refer to page 16 for information on opening the online help, or refer to Chapter 5 starting on page 185 for information on contacting the Data Translation Technical Support Department.

What You Need

To use the DataAcq SDK, you need the following:

- Pentium-based computer. For Windows 2000, 133 MHz or higher, with Service Pack 4. For Windows XP, 233 MHz or higher with Service Pack 2.
- Windows 2000, Windows XP, or Windows Vista
- Minimum RAM requirements depend on the operating system you are using; consult your operating system documentation for details
- CD-ROM drive
- One or more of the supported Data Translation data acquisition devices
- Any language that has the ability to call DLLs and receive callbacks
Installing the Software

The DataAcq SDK is installed automatically when you install the device driver for the module. Refer to your board manual for more information.

Using the DataAcq SDK Online Help

The *DataAcq SDK User’s Manual* is intended to be used with the online help for the DataAcq SDK. The online help contains all of the specific reference information for each of the functions, error codes, and Windows messages not included in this manual.

To open the online help file, select the following from the Windows Task Bar: Start | Programs | Data Translation, Inc | DT-Open Layers for Win32 | SDK | Data Acquisition SDK Help from the Windows Start menu.
About the Example Programs

To help you understand more about using the functions of the DataAcq SDK in an actual program, the DataAcq SDK provides a C example program (CEXMPL32.EXE). This example program allows you to configure any of the subsystems on the data acquisition device. The source code is located in the following directory: C:\Program Files\Data Translation\Win32\SDK\Examples\CExample. Resource files are also provided.

Additionally, the DataAcq SDK provides the following simple example programs. These programs are designed to use minimum Windows user interface code, while demonstrating the functions of the DataAcq SDK. Source code and resource files are provided for each of these programs:

- **ContAdc** – Opens the first available DT-Open Layers device, opens and configures an A/D subsystem, and performs continuous operations. Displays results in a dialog box.

- **ContDac** – Opens the first available DT-Open Layers device, opens and configures a D/A subsystem, and performs continuous operations outputting a square wave. Displays results in a dialog box.

- **DtConsole** – Opens the first available DT-Open Layers device, opens and configures an A/D subsystem, and performs a continuous A/D operation on a console screen.

- **IepeContAdc** – Opens the first available DT-Open Layers device, opens and configures an A/D subsystem, configures a channel to use AC coupling and an internal excitation current source for an IEPE input, and performs a continuous A/D operation. Displays results in a dialog box.

- **GenerateFreq** – Opens the first available DT-Open Layers device, opens and configures a C/T subsystem, and continuously outputs a pulse. Displays results in a dialog box.
• **MeasureFreq** – Opens the first available DT-Open Layers device, opens and configures a C/T subsystem, and continuously measures a pulse. Displays results in a dialog box.

• **SvAdc** – Opens the first available DT-Open Layers device, opens and configures an A/D subsystem, and performs a single-value operation. Displays results in a message box.

• **SvDac** – Opens the first available DT-Open Layers device, opens and configures a D/A subsystem, and performs a single-value operation with maximum positive and maximum negative range. Displays results in a message box.

• **SvDin** – Opens the first available DT-Open Layers device, opens and configures a DIN subsystem, and performs a single-value operation. Displays results in a message box.

• **SvDout** – Opens the first available DT-Open Layers device, opens and configures a DOUT subsystem, and performs a single-value operation. Displays results in a message box.

• **ThermoAdc** – Opens the first available DT-Open Layers device, opens and configures an A/D subsystem, and performs a thermocouple measurement. Displays results in a dialog box.

Each example program provided in the DataAcq SDK comes with the MSVC.DSW (workspace) and MSVC.DSP (project) files for use in the integrated development environment provided by the Microsoft Developer’s Studio. No special switches are necessary beyond instructing the IDE to create a Windows EXE or DLL.

**Note:** The DataAcq SDK installation program automatically includes an environment variable (DA_SDK). All the example programs use this environment variable; therefore, you can build the example programs without adding any include or library files to your projects.
About the Library Function Calling Conventions

The DataAcq SDK functions adhere to the Microsoft Pascal calling conventions. You can find prototypes for these functions in the include files OLDAAPI.H and OLMEM.H. We recommend that you follow these calling conventions for proper operation.

DataAcq SDK functions return an ECODE value, which is an unsigned long value indicating the status of the requested function. Check the return status value for an error condition using the symbolic constants defined in the include files. This practice is illustrated in the C example program (CEXMP32.EXE).

Note: For detailed information on the error codes, refer to the DataAcq SDK online help.
2

Function Summary

Data Acquisition Functions ........................................ 22
Data Management Functions ........................................ 48
Data Acquisition Functions

The following groups of data acquisition functions are available:

- Information functions
- Initialization and termination functions
- Configuration functions
- Operation functions
- Data conversion functions

These functions are briefly described in the following subsections.

Note: For specific information about each of these functions, refer to the DataAcq SDK online help. See page 16 for information on opening the online help file.

Information Functions

To determine the capabilities of your installed devices, subsystems on each device, and software, use the Information functions listed in Table 1.

Table 1: Information Functions

<table>
<thead>
<tr>
<th>Query about</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Devices</td>
<td>olDaEnumBoards</td>
<td>Lists all currently installed DT-Open Layers data acquisition devices, drivers, and driver parameters.</td>
</tr>
<tr>
<td></td>
<td>olDaEnumBoardsEx</td>
<td>Lists all currently-installed DT-Open Layers DataAcq devices and returns registry information for each device.</td>
</tr>
</tbody>
</table>
Table 1: Information Functions (cont.)

<table>
<thead>
<tr>
<th>Query about</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Devices (cont.)</td>
<td>olGetBoardInfo</td>
<td>Gets the driver name, model name, and instance number of the specified board, based on its board name.</td>
</tr>
<tr>
<td></td>
<td>olDaGetDeviceName</td>
<td>Gets the full name of the specified device (this name is set by the driver as part of the installation procedure).</td>
</tr>
<tr>
<td>Subsystems</td>
<td>olDaEnumSubSystems</td>
<td>Lists the names, types, and element number for each subsystem supported by the specified device.</td>
</tr>
<tr>
<td></td>
<td>olDaGetDevCaps</td>
<td>Returns the number of elements available for the specified subsystem on the specified device.</td>
</tr>
<tr>
<td></td>
<td>olDaGetSSCaps</td>
<td>Returns information about whether the specified subsystem capability is supported and/or the number of capabilities supported. Refer to Table 2 for a list of possible capabilities and return values.</td>
</tr>
<tr>
<td></td>
<td>olDaGetSSCapsEx</td>
<td>Returns information about extended subsystem capabilities. Refer to Table 3 for a list of possible capabilities and return values.</td>
</tr>
<tr>
<td></td>
<td>olDaEnumSSCaps</td>
<td>Lists the possible settings for the specified subsystem capabilities, including filters, ranges, gains, and resolution.</td>
</tr>
<tr>
<td></td>
<td>olDaGetDASSInfo</td>
<td>Returns the subsystem type and element number of the specified subsystem with the specified device handle.</td>
</tr>
</tbody>
</table>
### Table 1: Information Functions (cont.)

<table>
<thead>
<tr>
<th>Query about</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subsystems (cont.)</td>
<td>olDaIsRunning</td>
<td>This function returns a Boolean value indicating whether the specified subsystem is currently running.</td>
</tr>
<tr>
<td></td>
<td>olDaGetQueueSize</td>
<td>Returns the size of the specified queue (ready, done or inprocess) for the specified subsystem. The size indicates the number of buffers on the specified queue.</td>
</tr>
<tr>
<td></td>
<td>olDaEnumSSLList</td>
<td>Lists all subsystems on the simultaneous start list.</td>
</tr>
<tr>
<td>Software</td>
<td>olDaGetDriverVersion</td>
<td>Returns the device driver version number.</td>
</tr>
<tr>
<td></td>
<td>olDaGetVersion</td>
<td>Returns the software version of the DataAcq SDK.</td>
</tr>
<tr>
<td></td>
<td>olDaGetErrorString</td>
<td>Returns the string that corresponds to a device error code value.</td>
</tr>
</tbody>
</table>

### Subsystem Capability Queries

Table 2 lists the subsystem capabilities that you can query using the `olDaGetSSCaps` function; this function returns values as integers.

Table 3 lists the subsystem capabilities that you can query using the `olDaGetSSCapsEx` function; this function returns values as floating-point numbers.

Note that capabilities may be added as new devices are developed; for the most recent set of capabilities, refer to the DataAcq SDK online help.
Table 2: Capabilities to Query with olDaGetSSCaps

<table>
<thead>
<tr>
<th>Query about</th>
<th>Capability</th>
<th>Function Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Flow Mode</td>
<td>OLSSC_SUP_SINGLEVALUE</td>
<td>Nonzero if subsystem supports single-value operations.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_CONTINUOUS</td>
<td>Nonzero if subsystem supports continuous post-trigger operations.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_CONTINUOUS_PRETRIG</td>
<td>Nonzero if subsystem supports continuous pre-trigger operations.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_CONTINUOUS_ABOUTTRIG</td>
<td>Nonzero if subsystem supports continuous about-trigger (both pre- and post-trigger) operations.</td>
</tr>
<tr>
<td>Simultaneous Operations</td>
<td>OLSSC_SUP_SIMULTANEOUS_START</td>
<td>Nonzero if subsystem can be started simultaneously with another subsystem on the device.</td>
</tr>
<tr>
<td>Pausing Operations</td>
<td>OLSSC_SUP_PAUSE</td>
<td>Nonzero if subsystem supports pausing during continuous operation.</td>
</tr>
<tr>
<td>Windows Messaging</td>
<td>OLSSC_SUP_POSTMESSAGE</td>
<td>Nonzero if subsystem supports asynchronous operations.</td>
</tr>
<tr>
<td>Buffering</td>
<td>OLSSC_SUP_BUFFERING</td>
<td>Nonzero if subsystem supports buffering.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_WRPSINGLE</td>
<td>Nonzero if subsystem supports single-buffer wrap mode.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_WRPMULTIPLE</td>
<td>Nonzero if subsystem supports multiple-buffer wrap mode.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_INPROCESSFLUSH</td>
<td>Nonzero if subsystem supports transferring data from a buffer on a subsystem's inprocess queue.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_WAVEFORM_MODE</td>
<td>Nonzero if subsystem supports waveform generation.</td>
</tr>
</tbody>
</table>
### Table 2: Capabilities to Query with olDaGetSSCaps (cont.)

<table>
<thead>
<tr>
<th>Query about</th>
<th>Capability</th>
<th>Function Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffering (cont.)</td>
<td>OLSSC_SUP_WRPWAVEFORM_ONLY</td>
<td>Nonzero if the subsystem supports waveform-based operations using the onboard FIFO only. If this capability is nonzero, the buffer wrap mode must be set to single. In addition, the buffer size must be less than or equal to the FIFO size.</td>
</tr>
<tr>
<td>DMA</td>
<td>OLSSC_NUMDMACHANS</td>
<td>Number of DMA channels supported.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_GAPFREE_NODMA</td>
<td>Nonzero if subsystem supports gap-free continuous operation with no DMA.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_GAPFREE_SINGLEDMA</td>
<td>Nonzero if subsystem supports gap-free continuous operation with a single DMA channel.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_GAPFREE_DUALDMA</td>
<td>Nonzero if subsystem supports gap-free continuous operation with two DMA channels.</td>
</tr>
<tr>
<td>Triggered Scan Mode</td>
<td>OLSSC_SUP_TRIGSCAN</td>
<td>Nonzero if subsystem supports triggered scans.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_MAXMULTISCAN</td>
<td>Maximum number of scans per trigger or retrigger supported by the subsystem.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_RETRIGGER_SCAN_PER_TRIGGER</td>
<td>Nonzero if subsystem supports scan-per-trigger triggered scan mode (retrigger is the same as the initial trigger source).</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_RETRIGGER_INTERNAL</td>
<td>Nonzero if subsystem supports internal retriggered scan mode. (retrigger source is on the device; initial trigger is any available trigger source).</td>
</tr>
</tbody>
</table>
# Table 2: Capabilities to Query with olDaGetSSCaps (cont.)

<table>
<thead>
<tr>
<th>Query about</th>
<th>Capability</th>
<th>Function Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triggered Scan Mode (cont.)</td>
<td>OLSSC_SUP_RETRIGGER_EXTRA</td>
<td>Nonzero if subsystem supports retrigger-extra triggered scan mode (retrigger can be any supported trigger source; initial trigger is any available trigger source).</td>
</tr>
<tr>
<td>Channel-Gain List</td>
<td>OLSSC_CGLDEPTH</td>
<td>Number of entries in channel-gain list.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_RANDOM_CGL</td>
<td>Nonzero if subsystem supports random channel-gain list setup.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_SEQUENTIAL_CGL</td>
<td>Nonzero if subsystem supports sequential channel-gain list setup.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_ZEROSEQUENTIAL_CGL</td>
<td>Nonzero if subsystem supports sequential channel-gain list setup starting with channel zero.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_SIMULTANEOUS_SH</td>
<td>Nonzero if subsystem supports simultaneous operations.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_CHANNELLIST_INHIBIT</td>
<td>Nonzero if subsystem supports channel-gain list entry inhibition.</td>
</tr>
<tr>
<td>Gain</td>
<td>OLSSC_SUP_PROGRAMGAIN</td>
<td>Nonzero if subsystem supports programmable gain.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_NUMGAINS</td>
<td>Number of gain selections.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_NONCONTIGUOUS_CHANNELNUM</td>
<td>Number of random-order entries allowed in the channel-gain list.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_SINGLEVALUE_AUTORANGE</td>
<td>Nonzero if subsystem supports autoranging operations.</td>
</tr>
<tr>
<td>Synchronous Digital I/O</td>
<td>OLSSC_SUP_SYNCHRONOUS_DIGITALIO</td>
<td>Nonzero if subsystem supports synchronous digital output operations.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_MAXDIGITALIOLIST_VALUE</td>
<td>Maximum value for synchronous digital output channel list entry.</td>
</tr>
</tbody>
</table>
Table 2: Capabilities to Query with olDaGetSSCaps (cont.)

<table>
<thead>
<tr>
<th>Query about</th>
<th>Capability</th>
<th>Function Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O Channels</td>
<td>OLSSC_NUMCHANNELS</td>
<td>Number of I/O channels.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_EXP2896</td>
<td>Nonzero if subsystem supports channel expansion with DT2896.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_EXP727</td>
<td>Nonzero if subsystem supports channel expansion with DT727.</td>
</tr>
<tr>
<td>Channel Type</td>
<td>OLSSC_SUP_SINGEENDED</td>
<td>Nonzero if subsystem supports single-ended inputs.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_MAXSECHANS</td>
<td>Number of single-ended channels.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_DIFFERENTIAL</td>
<td>Nonzero if subsystem supports differential inputs.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_MAXDICHANS</td>
<td>Number of differential channels.</td>
</tr>
<tr>
<td>Filters</td>
<td>OLSSC_SUP_FILTERPERCHAN</td>
<td>Nonzero if subsystem supports filtering per channel.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_NUMFILTERS</td>
<td>Number of filter selections.</td>
</tr>
<tr>
<td>Ranges</td>
<td>OLSSC_NUMRANGES</td>
<td>Number of range selections.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_RANGEPERCHANNEL</td>
<td>Nonzero if subsystem supports different range settings for each channel.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_CURRENT_OUTPUTS</td>
<td>Non-zero if subsystem supports current outputs.</td>
</tr>
<tr>
<td>Resolution</td>
<td>OLSSC_SUP_SWRESOLUTION</td>
<td>Nonzero if subsystem supports software-programmable resolution.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_NUMRESOLUTIONS</td>
<td>Number of different resolutions that you can program for the subsystem.</td>
</tr>
</tbody>
</table>
### Table 2: Capabilities to Query with olDaGetSSCaps (cont.)

<table>
<thead>
<tr>
<th>Query about</th>
<th>Capability</th>
<th>Function Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Encoding</td>
<td>OLSSC_SUP_BINARY</td>
<td>Nonzero if subsystem supports binary encoding.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_2SCOMP</td>
<td>Nonzero if subsystem supports two's complement encoding.</td>
</tr>
<tr>
<td>Triggers</td>
<td>OLSSC_SUP_SOFTTRIG</td>
<td>Nonzero if subsystem supports an internal software trigger.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_EXTERNTRIG</td>
<td>Nonzero if subsystem supports an external digital (TTL) trigger for a continuous operation.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_SV_POS_EXTERN_TTLTRIG</td>
<td>Nonzero if subsystem supports a positive, external digital (TTL) trigger for a single-value operation.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_SV_NEG_EXTERN_TTLTRIG</td>
<td>Nonzero if subsystem supports a negative, external digital (TTL) trigger for a single-value operation.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_THRESHTRIGPOS</td>
<td>Nonzero if subsystem supports a positive analog threshold trigger.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_THRESHTRIGNEG</td>
<td>Nonzero if subsystem supports a negative analog threshold trigger.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_ANALOGEVENTTRIG</td>
<td>Nonzero if subsystem supports analog event trigger.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_DIGITALEVENTTRIG</td>
<td>Nonzero if subsystem supports digital event trigger.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_TIMEREVENTTRIG</td>
<td>Nonzero if subsystem supports timer event trigger.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_NUMEXTRATRIGGERS</td>
<td>Number of extra trigger sources supported.</td>
</tr>
</tbody>
</table>
Table 2: Capabilities to Query with olDaGetSSCaps (cont.)

<table>
<thead>
<tr>
<th>Query about</th>
<th>Capability</th>
<th>Function Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clocks</td>
<td>OLSSC_SUP_INTCLOCK</td>
<td>Nonzero if subsystem supports internal clock.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_EXTCLOCK</td>
<td>Nonzero if subsystem supports external clock.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_NUMEXTRACLOCKS</td>
<td>Number of extra clock sources.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_SIMULTANEOUS_CLOCKING</td>
<td>Non-zero if subsystem supports simultaneous clocking of all channels.</td>
</tr>
<tr>
<td>Coupling</td>
<td>OLSSC_SUP_AC_COUPLING</td>
<td>Non-zero if subsystem supports programmable AC coupling for an IEPE input.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_DC_COUPLING</td>
<td>Non-zero if subsystem supports programmable DC coupling for an IEPE input.</td>
</tr>
<tr>
<td>Excitation Current</td>
<td>OLSSC_SUP_INTERNAL_EXCITATION_CURRENT_SRC</td>
<td>Non-zero if subsystem supports a programmable internal excitation current source for an IEPE input.</td>
</tr>
<tr>
<td>Source</td>
<td>OLSSC_SUP_EXTERNAL_EXCITATION_CURRENT_SRC</td>
<td>Non-zero if subsystem supports a programmable external excitation current source for an IEPE input.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_NUM_EXCITATION_CURRENT_VALUES</td>
<td>Number of different values for the internal excitation current source that can be programmed for the subsystem.</td>
</tr>
<tr>
<td>Query about</td>
<td>Capability</td>
<td>Function Returns</td>
</tr>
<tr>
<td>-------------</td>
<td>------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Thermocouples</td>
<td>OLSSC_SUP_THERMOCOUPLES</td>
<td>Non-zero if subsystem supports thermocouple inputs.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_TEMPERATURE_DATA_IN_STREAM</td>
<td>(Has meaning only if OLSSC_SUP_THERMOCOUPLES is non-zero.) Non-zero if the device performs CJC correction and linearization and returns temperature values in the data stream. If zero, the device returns only raw counts and the application must perform the CJC correction and linearization.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_INTERLEAVED_CJC_IN_STREAM</td>
<td>(Has meaning only if OLSSC_SUP_TEMPERATURE_DATA_IN_STREAM is non-zero.) Non-zero if the device can optionally interleave CJC data with A/D data in the data stream.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_CJC_SOURCE_CHANNEL</td>
<td>(Has meaning only if OLSSC_SUP_TEMPERATURE_DATA_IN_STREAM is 0.) Non-zero if one of the analog input channels on the device is used as the CJC input.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_CJC_SOURCE_INTERNAL</td>
<td>(Has meaning only if OLSSC_SUP_TEMPERATURE_DATA_IN_STREAM is non-zero.) Non-zero if the CJC is measured internally on the device rather than using one of the analog input channels as the CJC input.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_RETURNS_FLOATS</td>
<td>Non-zero if the subsystem returns floating-point rather than integer values.</td>
</tr>
</tbody>
</table>
### Table 2: Capabilities to Query with olDaGetSSCaps (cont.)

<table>
<thead>
<tr>
<th>Query about</th>
<th>Capability</th>
<th>Function Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter/Timer Modes</td>
<td>OLSSC_SUP_CASCADING</td>
<td>Nonzero if subsystem supports cascading.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_CTMODE_COUNT</td>
<td>Nonzero if subsystem supports event counting mode.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_CTMODE_RATE</td>
<td>Nonzero if subsystem supports rate generation (continuous pulse output) mode.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_CTMODE_ONESHOT</td>
<td>Nonzero if subsystem supports (single) one-shot mode.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_CTMODE_ONESHOT_RPT</td>
<td>Nonzero if subsystem supports repetitive one-shot mode.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_CTMODE_UP_DOWN</td>
<td>Nonzero if subsystem supports up/down counting mode.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_CTMODE_MEASURE</td>
<td>Returns a value indicating how edge-to-edge measurement mode is supported (see page 115 for more information).</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_CTMODE_CONT_MEASURE</td>
<td>Returns a value indicating how continuous edge-to-edge measurement mode is supported (see page 115 for more information).</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_QUADRATURE_DECODER</td>
<td>Nonzero if subsystem supports taking input from quadrature encoders.</td>
</tr>
<tr>
<td>Counter/Timer Pulse Output Types</td>
<td>OLSSC_FIXED_PULSE_WIDTH</td>
<td>Nonzero if subsystem supports fixed output pulse widths.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_PLS_HIGH2LOW</td>
<td>Nonzero if subsystem supports high-to-low output pulses.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_PLS_LOW2HIGH</td>
<td>Nonzero if subsystem supports low-to-high output pulses.</td>
</tr>
</tbody>
</table>
### Table 2: Capabilities to Query with olDaGetSSCaps (cont.)

<table>
<thead>
<tr>
<th>Query about</th>
<th>Capability</th>
<th>Function Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter/Timer</td>
<td>OLSSC_SUP_GATE_NONE</td>
<td>Nonzero if subsystem supports an internal (software) gate type.</td>
</tr>
<tr>
<td>Gates</td>
<td>OLSSC_SUP_GATE_HIGH_LEVEL</td>
<td>Nonzero if subsystem supports high-level gate type.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_GATE_LOW_LEVEL</td>
<td>Nonzero if subsystem supports low-level gate type.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_GATE_HIGH_EDGE</td>
<td>Nonzero if subsystem supports high-edge gate type.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_GATE_LOW_EDGE</td>
<td>Nonzero if subsystem supports low-edge gate type.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_GATE_LEVEL</td>
<td>Nonzero if subsystem supports level change gate type.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_GATE_HIGH_LEVEL_DEBOUNCE</td>
<td>Nonzero if subsystem supports high-level gate type with input debounce.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_GATE_LOW_LEVEL_DEBOUNCE</td>
<td>Nonzero if subsystem supports low-level gate type with input debounce.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_GATE_HIGH_EDGE_DEBOUNCE</td>
<td>Nonzero if subsystem supports high-edge gate type with input debounce.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_GATE_LOW_EDGE_DEBOUNCE</td>
<td>Nonzero if subsystem supports low-edge gate type with input debounce.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_GATE_LEVEL_DEBOUNCE</td>
<td>Nonzero if subsystem supports level change gate type with input debounce.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_SUP_INTERRUPT</td>
<td>Nonzero if subsystem supports interrupt-driven I/O.</td>
</tr>
</tbody>
</table>
### Table 2: Capabilities to Query with `olDaGetSSCaps` (cont.)

<table>
<thead>
<tr>
<th>Query about</th>
<th>Capability</th>
<th>Function Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIFOs</td>
<td>OLSSC_SUP_FIFO</td>
<td>Nonzero if subsystem has a FIFO in the data path.</td>
</tr>
<tr>
<td></td>
<td>OLSSC_FIFO_SIZE_IN_K</td>
<td>Size of the output FIFO, in kilobytes.</td>
</tr>
<tr>
<td>Processors</td>
<td>OLSSC_SUP_PROCESSOR</td>
<td>Nonzero if subsystem has a processor on device.</td>
</tr>
<tr>
<td>Software Calibration</td>
<td>OLSSC_SUP_SWCAL</td>
<td>Nonzero if subsystem supports software calibration.</td>
</tr>
</tbody>
</table>

### Table 3: Capabilities to Query with `olDaGetSSCapsEx`

<table>
<thead>
<tr>
<th>Query about</th>
<th>Capability</th>
<th>Function Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triggered Scan Mode</td>
<td>OLSSCE_MAXRETRIGGER</td>
<td>Maximum retrigger frequency supported by the subsystem.</td>
</tr>
<tr>
<td></td>
<td>OLSSCE_MINRETRIGGER</td>
<td>Minimum retrigger frequency supported by the subsystem.</td>
</tr>
<tr>
<td>Clocks</td>
<td>OLSSCE_BASECLOCK</td>
<td>Base clock frequency supported by the subsystem.</td>
</tr>
<tr>
<td></td>
<td>OLSSCE_MAXCLOCKDIVIDER</td>
<td>Maximum external clock divider supported by the subsystem.</td>
</tr>
<tr>
<td></td>
<td>OLSSCE_MINCLOCKDIVIDER</td>
<td>Minimum external clock divider supported by the subsystem.</td>
</tr>
<tr>
<td></td>
<td>OLSSCE_MAXTHROUGHPUT</td>
<td>Maximum throughput supported by the subsystem.</td>
</tr>
<tr>
<td></td>
<td>OLSSCE_MINTHROUGHPUT</td>
<td>Minimum throughput supported by the subsystem.</td>
</tr>
</tbody>
</table>
Initialization and Termination Functions

Once you have identified the available devices, use the Initialization functions described in Table 4.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>olDaInitialize</td>
<td>Provides the means for the software to associate specific requests with a particular device; it must be called before any other function. This function loads a specified device’s software support and provides a “device handle” value. This value is used to identify the device, and must be supplied as an argument in all subsequent function calls that reference the device.</td>
</tr>
<tr>
<td>olDaGetDASS</td>
<td>Allocates a subsystem for use by returning a handle to the subsystem.</td>
</tr>
</tbody>
</table>

When you are finished with your program, use the Termination functions listed in Table 5.

### Table 3: Capabilities to Query with olDaGetSScapsEx (cont.)

<table>
<thead>
<tr>
<th>Query about</th>
<th>Capability</th>
<th>Function Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thermo-couples</td>
<td>OLSSCE_CJC_MILLIVOLTS_PER_DEGREE_C</td>
<td>(Has meaning only if OLSSC_SUP_THERMOCOUPLES is non-zero.) Number of millivolts per degree C for the CJC input.</td>
</tr>
</tbody>
</table>
Configuration Functions

Once you have initialized a subsystem and determined what its capabilities are, set or get the value of the subsystem’s parameters by calling the Configuration functions listed in Table 6.

Table 5: Termination Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>olDaReleaseDASS</td>
<td>Releases the specified subsystem and relinquishes all resources associated with it.</td>
</tr>
<tr>
<td>olDaTerminate</td>
<td>Ends a session between your application and the specified device. The device is returned to an inactive state and all resources are returned to the system.</td>
</tr>
</tbody>
</table>

Table 6: Configuration Functions

<table>
<thead>
<tr>
<th>Feature</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Flow Mode</td>
<td>olDaSetDataFlow</td>
<td>Sets the data flow mode.</td>
</tr>
<tr>
<td></td>
<td>olDaGetDataFlow</td>
<td>Gets the data flow mode.</td>
</tr>
<tr>
<td>Windows Messaging</td>
<td>olDaSetNotificationProcedure</td>
<td>Specifies the notification procedure to call for information messages from the subsystem.</td>
</tr>
<tr>
<td></td>
<td>olDaGetNotificationProcedure</td>
<td>Gets the address of the notification procedure.</td>
</tr>
<tr>
<td></td>
<td>olDaSetWndHandle</td>
<td>Sets the window to which information messages are sent.</td>
</tr>
<tr>
<td></td>
<td>olDaGetWndHandle</td>
<td>Gets the window handle.</td>
</tr>
</tbody>
</table>
Table 6: Configuration Functions (cont.)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer Wrap Mode</td>
<td>olDaSetWrapMode</td>
<td>Sets the buffer processing wrap mode.</td>
</tr>
<tr>
<td></td>
<td>olDaGetWrapMode</td>
<td>Gets the buffer processing wrap mode.</td>
</tr>
<tr>
<td>DMA</td>
<td>olDaSetDmaUsage</td>
<td>Sets the number of DMA channels to be used.</td>
</tr>
<tr>
<td></td>
<td>olDaGetDmaUsage</td>
<td>Gets the number of DMA channels to be used.</td>
</tr>
<tr>
<td>Triggered Scans</td>
<td>olDaSetTriggeredScanUsage</td>
<td>Enables or disables triggered scan mode.</td>
</tr>
<tr>
<td></td>
<td>olDaGetTriggeredScanUsage</td>
<td>Gets the triggered scan mode setting.</td>
</tr>
<tr>
<td></td>
<td>olDaSetMultiscanCount</td>
<td>Sets the number of times to scan per trigger/retrigger.</td>
</tr>
<tr>
<td></td>
<td>olDaGetMultiscanCount</td>
<td>Gets the number of times to scan per trigger/retrigger.</td>
</tr>
<tr>
<td></td>
<td>olDaSetRetriggerMode</td>
<td>Sets the retrigger mode.</td>
</tr>
<tr>
<td></td>
<td>olDaGetRetriggerMode</td>
<td>Gets the retrigger mode.</td>
</tr>
<tr>
<td></td>
<td>olDaSetRetriggerFrequency</td>
<td>Sets the frequency of the internal retrigger when using internal retrigger mode.</td>
</tr>
<tr>
<td></td>
<td>olDaGetRetriggerFrequency</td>
<td>Gets the frequency of the internal retrigger when using internal retrigger mode.</td>
</tr>
</tbody>
</table>
Table 6: Configuration Functions (cont.)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel-Gain List</td>
<td><strong>olDaSetChannelListSize</strong></td>
<td>Sets the size of the channel-gain list.</td>
</tr>
<tr>
<td></td>
<td><strong>olDaGetChannelListSize</strong></td>
<td>Gets the size of the channel-gain list.</td>
</tr>
<tr>
<td></td>
<td><strong>olDaSetChannelListEntry</strong></td>
<td>Sets the channel number of a channel-gain list entry.</td>
</tr>
<tr>
<td></td>
<td><strong>olDaGetChannelListEntry</strong></td>
<td>Gets the channel number of a channel-gain list entry.</td>
</tr>
<tr>
<td></td>
<td><strong>olDaSetGainListEntry</strong></td>
<td>Sets a gain value for a channel-gain list entry.</td>
</tr>
<tr>
<td></td>
<td><strong>olDaGetGainListEntry</strong></td>
<td>Gets the gain value of a channel-gain list entry.</td>
</tr>
<tr>
<td></td>
<td><strong>olDaSetChannelListEntryInhibit</strong></td>
<td>Enables/disables channel entry inhibition for a channel-gain list entry.</td>
</tr>
<tr>
<td></td>
<td><strong>olDaGetChannelListEntryInhibit</strong></td>
<td>Gets the channel entry inhibition setting of a channel-gain list entry.</td>
</tr>
<tr>
<td></td>
<td><strong>olDaSetDigitalIOListEntry</strong></td>
<td>Sets the digital value to output for the channel-gain list entry.</td>
</tr>
<tr>
<td></td>
<td><strong>olDaGetDigitalIOListEntry</strong></td>
<td>Gets the digital value to output for the channel-gain list entry.</td>
</tr>
<tr>
<td>Synchronous Digital I/O</td>
<td><strong>olDaSetSynchronousDigitalIOUsage</strong></td>
<td>Enables or disables synchronous digital I/O operations.</td>
</tr>
<tr>
<td></td>
<td><strong>olDaGetSynchronousDigitalIOUsage</strong></td>
<td>Gets the synchronous digital I/O setting.</td>
</tr>
<tr>
<td>Feature</td>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Channel Type</td>
<td><strong>olDaSetChannelType</strong></td>
<td>Sets the channel configuration type of a channel.</td>
</tr>
<tr>
<td></td>
<td><strong>olDaGetChannelType</strong></td>
<td>Gets the channel configuration type of a channel.</td>
</tr>
<tr>
<td>Filters</td>
<td><strong>olDaSetChannelFilter</strong></td>
<td>Sets the filter cut-off frequency for a channel.</td>
</tr>
<tr>
<td></td>
<td><strong>olDaGetChannelFilter</strong></td>
<td>Gets the filter cut-off frequency for a channel.</td>
</tr>
<tr>
<td>Ranges</td>
<td><strong>olDaSetRange</strong></td>
<td>Sets the voltage range for a subsystem.</td>
</tr>
<tr>
<td></td>
<td><strong>olDaGetRange</strong></td>
<td>Gets the voltage range for a subsystem.</td>
</tr>
<tr>
<td></td>
<td><strong>olDaSetChannelRange</strong></td>
<td>Sets the voltage range for a channel.</td>
</tr>
<tr>
<td></td>
<td><strong>olDaGetChannelRange</strong></td>
<td>Gets the voltage range for a channel.</td>
</tr>
<tr>
<td>Resolution</td>
<td><strong>olDaSetResolution</strong></td>
<td>Sets the number of bits of resolution.</td>
</tr>
<tr>
<td></td>
<td><strong>olDaGetResolution</strong></td>
<td>Gets the number of bits of resolution.</td>
</tr>
<tr>
<td>Data Encoding</td>
<td><strong>olDaSetEncoding</strong></td>
<td>Sets the data encoding type.</td>
</tr>
<tr>
<td></td>
<td><strong>olDaGetEncoding</strong></td>
<td>Gets the data encoding type.</td>
</tr>
</tbody>
</table>

Table 6: Configuration Functions (cont.)
### Triggers

<table>
<thead>
<tr>
<th>Feature</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>olDaSetTrigger</code></td>
<td>Sets the post-trigger source.</td>
</tr>
<tr>
<td></td>
<td><code>olDaGetTrigger</code></td>
<td>Gets the post-trigger source.</td>
</tr>
<tr>
<td></td>
<td><code>olDaSetPretriggerSource</code></td>
<td>Sets the pre-trigger source.</td>
</tr>
<tr>
<td></td>
<td><code>olDaGetPretriggerSource</code></td>
<td>Gets the pre-trigger source.</td>
</tr>
<tr>
<td></td>
<td><code>olDaSetRetrigger</code></td>
<td>Sets the retrigger source for retrigger-extra retrigger mode.</td>
</tr>
<tr>
<td></td>
<td><code>olDaGetRetrigger</code></td>
<td>Gets the retrigger source for retrigger-extra retrigger mode.</td>
</tr>
</tbody>
</table>

### Clocks

<table>
<thead>
<tr>
<th>Feature</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>olDaSetClockSource</code></td>
<td>Sets the clock source.</td>
</tr>
<tr>
<td></td>
<td><code>olDaGetClockSource</code></td>
<td>Gets the clock source.</td>
</tr>
<tr>
<td></td>
<td><code>olDaSetClockFrequency</code></td>
<td>Sets the frequency of the internal clock or a counter/timer’s output frequency.</td>
</tr>
<tr>
<td></td>
<td><code>olDaGetClockFrequency</code></td>
<td>Gets the frequency of the internal clock or a counter/timer’s output frequency.</td>
</tr>
<tr>
<td></td>
<td><code>olDaSetExternalClockDivider</code></td>
<td>Sets the input divider value of the external clock.</td>
</tr>
<tr>
<td></td>
<td><code>olDaGetExternalClockDivider</code></td>
<td>Gets the input divider value of the external clock.</td>
</tr>
</tbody>
</table>
Table 6: Configuration Functions (cont.)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEPE Support</td>
<td><code>olDaSetCouplingType</code></td>
<td>Sets the coupling type to use or the specified channel.</td>
</tr>
<tr>
<td></td>
<td><code>olDaGetCouplingType</code></td>
<td>Gets the coupling type that is used by the specified channel.</td>
</tr>
<tr>
<td></td>
<td><code>olDaSetExcitationCurrentSource</code></td>
<td>Sets the excitation current source for the specified channel.</td>
</tr>
<tr>
<td></td>
<td><code>olDaGetExcitationCurrentSource</code></td>
<td>Gets the excitation current source that is used by the specified channel.</td>
</tr>
<tr>
<td></td>
<td><code>olDaSetExcitationCurrentValue</code></td>
<td>Sets the value of the internal excitation current source to use for the specified channel.</td>
</tr>
<tr>
<td></td>
<td><code>olDaGetExcitationCurrentValue</code></td>
<td>Gets the value of the internal excitation current source that is used by the specified channel.</td>
</tr>
<tr>
<td>Thermo-couple Support</td>
<td><code>olDaSetThermocoupleType</code></td>
<td>Sets the type of thermocouple to use for the input channel of the specified subsystem.</td>
</tr>
<tr>
<td></td>
<td><code>olDGetThermocoupleType</code></td>
<td>Gets the currently configured thermocouple type for the input channel of the specified subsystem.</td>
</tr>
<tr>
<td></td>
<td><code>olDaSetReturnCjcTemperatureInStream</code></td>
<td>Enables or disables the specified subsystem from returning CJC values in the data stream.</td>
</tr>
</tbody>
</table>
### Table 6: Configuration Functions (cont.)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thermo-couple Support</td>
<td>olDaGetReturnCjcTemperatureInStream</td>
<td>Returns whether the specified subsystem has been enabled or disabled from returning CJC values in the data stream.</td>
</tr>
<tr>
<td>(cont.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Counter/Timers</td>
<td>olDaSetCTMode</td>
<td>Sets the counter/timer mode.</td>
</tr>
<tr>
<td></td>
<td>olDaGetCTMode</td>
<td>Gets the counter/timer mode.</td>
</tr>
<tr>
<td></td>
<td>olDaSetCascadeMode</td>
<td>Sets the counter/timer cascade mode.</td>
</tr>
<tr>
<td></td>
<td>olDaGetCascadeMode</td>
<td>Gets the counter/timer cascade mode.</td>
</tr>
<tr>
<td></td>
<td>olDaSetGateType</td>
<td>Sets the gate type for the counter/timer mode.</td>
</tr>
<tr>
<td></td>
<td>olDaGetGateType</td>
<td>Gets the gate type for the counter/timer mode.</td>
</tr>
<tr>
<td></td>
<td>olDaSetPulseType</td>
<td>Sets the pulse type for the counter/timer mode.</td>
</tr>
<tr>
<td></td>
<td>olDaGetPulseType</td>
<td>Gets the pulse type for the counter/timer mode.</td>
</tr>
<tr>
<td></td>
<td>olDaSetPulseWidth</td>
<td>Sets the pulse output width for the counter/timer mode.</td>
</tr>
<tr>
<td></td>
<td>olDaGetPulseWidth</td>
<td>Gets the pulse width for the counter/timer mode.</td>
</tr>
<tr>
<td></td>
<td>olDaSetMeasureStartEdge</td>
<td>Sets the start edge for edge-to-edge measurement operations.</td>
</tr>
<tr>
<td></td>
<td>olDaGetMeasureStartEdge</td>
<td>Gets the start edge for edge-to-edge measurement operations.</td>
</tr>
</tbody>
</table>
Table 6: Configuration Functions (cont.)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter/Timers (cont.)</td>
<td>olDaSetMeasureStopEdge</td>
<td>Sets the stop edge for edge-to-edge measurement operations.</td>
</tr>
<tr>
<td></td>
<td>olDaGetMeasureStopEdge</td>
<td>Gets the stop edge for edge-to-edge measurement operations.</td>
</tr>
<tr>
<td></td>
<td>olDaSetQuadDecoder</td>
<td>Sets various aspects of a quadrature decoder.</td>
</tr>
</tbody>
</table>

Operation Functions

Once you have set the parameters of a subsystem, use the Operation functions listed in Table 7.

Table 7: Operation Functions

<table>
<thead>
<tr>
<th>Operation</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-Value Operations</td>
<td>olDaGetSingleValue</td>
<td>Reads a single input value from the specified channel of a subsystem (using a specified gain), and returns the data as a 32-bit integer (long).</td>
</tr>
<tr>
<td></td>
<td>olDaGetSingleValues</td>
<td>For subsystems that support simultaneous operations, reads a single input value from all of the input channels of the specified subsystem (using a specified gain), and returns the data as 32-bit integers (longs).</td>
</tr>
<tr>
<td>Operation</td>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Single-Value Operations</td>
<td><code>olDaGetSingleFloat</code></td>
<td>Reads a single input value from the specified channel of a subsystem (using a specified gain) and returns the data as a floating-point value.</td>
</tr>
<tr>
<td>(cont.)</td>
<td><code>olDaGetSingleFloats</code></td>
<td>For subsystems that support simultaneous operations, reads a single input value from all of the input channels of the specified subsystem (using the specified gain), and returns the data as floating-point values.</td>
</tr>
<tr>
<td></td>
<td><code>olDaGetCjcTemperature</code></td>
<td>Reads the CJC temperature of an input channel on the specified subsystem, and returns the temperature, in degrees C, as a floating-point value.</td>
</tr>
<tr>
<td></td>
<td><code>olDaGetCjcTemperatures</code></td>
<td>For subsystems that support simultaneous operations, this function reads the CJC temperature of each input channel on the specified subsystem, and returns the temperatures, in degrees C, as floating-point values.</td>
</tr>
<tr>
<td></td>
<td><code>olDaGetSingleValueEx</code></td>
<td>Determines the appropriate gain for the range (called autoranging), if desired, reads a single input value from the specified subsystem channel, and returns the value as both a code and a voltage.</td>
</tr>
<tr>
<td></td>
<td><code>olDaPutSingleValue</code></td>
<td>Writes a single output value to the specified channel of a subsystem.</td>
</tr>
</tbody>
</table>
### Table 7: Operation Functions (cont.)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-Value Operations</td>
<td><code>olDaPutSingleValues</code></td>
<td>For subsystems that support simultaneous operations, writes a single value to each output channel of the specified subsystem.</td>
</tr>
<tr>
<td>Configure Operation</td>
<td><code>olDaConfig</code></td>
<td>After setting up a specified subsystem using the configuration functions, configures the subsystem with new parameter values.</td>
</tr>
<tr>
<td>Start/Stop Operations</td>
<td><code>olDaStart</code></td>
<td>Starts the operation for which the subsystem has been configured.</td>
</tr>
<tr>
<td></td>
<td><code>olDaPause</code></td>
<td>Pauses a continuous operation on the subsystem.</td>
</tr>
<tr>
<td></td>
<td><code>olDaContinue</code></td>
<td>Continues the previously paused operation on the subsystem.</td>
</tr>
<tr>
<td></td>
<td><code>olDaStop</code></td>
<td>Stops the operation and returns the subsystem to the ready state.</td>
</tr>
<tr>
<td></td>
<td><code>olDaAbort</code></td>
<td>Stops the subsystem’s operation immediately.</td>
</tr>
<tr>
<td></td>
<td><code>olDaReset</code></td>
<td>Causes the operation to terminate immediately, and re-initializes the subsystem.</td>
</tr>
<tr>
<td>Buffer Operations</td>
<td><code>olDaGetBuffer</code></td>
<td>Gets a completed buffer from the done queue of the specified subsystem.</td>
</tr>
<tr>
<td></td>
<td><code>olDaPutBuffer</code></td>
<td>Assigns a data buffer for the subsystem to the ready queue.</td>
</tr>
<tr>
<td></td>
<td><code>olDaFlushBuffers</code></td>
<td>Transfers all data buffers held by the subsystem to the done queue.</td>
</tr>
</tbody>
</table>
### Table 7: Operation Functions (cont.)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer Operations (cont.)</td>
<td><code>olDaFlushFromBufferInprocess</code></td>
<td>Copies all valid samples, up to a given number of samples, from the inprocess buffer to a specified buffer. It also sets the logical size of the buffer with flushed data to the number of samples copied and places the inprocess buffer on the done queue when it has been filled with the remaining samples.</td>
</tr>
<tr>
<td>Counter/Timer Operations</td>
<td><code>olDaReadEvents</code></td>
<td>Gets the number of events that have been counted since the subsystem was started with <code>olDaStart</code>.</td>
</tr>
<tr>
<td></td>
<td><code>olDaMeasureFrequency</code></td>
<td>Measures the frequency of the input clock source for the selected counter/timer.</td>
</tr>
<tr>
<td>Simultaneous Operations</td>
<td><code>olDaGetSSLList</code></td>
<td>Gets a handle to a simultaneous start list.</td>
</tr>
<tr>
<td></td>
<td><code>olDaPutDassToSSLList</code></td>
<td>Puts the specified subsystem on the simultaneous start list.</td>
</tr>
<tr>
<td></td>
<td><code>olDaSimultaneousPreStart</code></td>
<td>Simultaneously prestarts (performs setup operations on) all subsystems on the specified simultaneous start list.</td>
</tr>
<tr>
<td></td>
<td><code>olDaSimultaneousStart</code></td>
<td>Simultaneously starts all subsystems on the specified simultaneous start list.</td>
</tr>
<tr>
<td></td>
<td><code>olDaReleaseSSLList</code></td>
<td>Releases the specified simultaneous start list and relinquishes all resources associated with it.</td>
</tr>
</tbody>
</table>
Data Conversion Functions

Once you have acquired data, you can use the functions listed in Table 8 to convert the data, if desired.

Table 8: Data Conversion Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>olDaCodeToVolts</td>
<td>Converts a code value to voltage value, using the range, gain, resolution, and encoding you specify.</td>
</tr>
<tr>
<td>olDaVoltsToCode</td>
<td>Converts a voltage value to code value, using the range, gain, resolution, and encoding you specify.</td>
</tr>
</tbody>
</table>
Data Management Functions

Data management functions link the various layers of the DT-Open Layers architecture together. The fundamental data object in the DataAcq SDK is a buffer. All functions that create, manipulate, and delete buffers are encapsulated in the data management portion of the DataAcq SDK.

The following groups of data management functions are available:

- Buffer management functions
- List management functions

The following subsections summarize these functions.

---

**Note:** For specific information about each of these functions, refer to the DataAcq SDK online help. See page 16 for information on launching the online help file.

---

Buffer Management Functions

The Buffer Management functions, listed in Table 9, are a set of object-oriented tools intended for both application and system programmers. When a buffer object is created, a buffer handle (HBUF) is returned. This handle is used in all subsequent buffer manipulation.
### Table 9: Buffer Management Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>olDmAllocBuffer</td>
<td>Creates a buffer object of a specified number of samples, where each sample is 2 bytes.</td>
</tr>
<tr>
<td>olDmCallocBuffer</td>
<td>Creates a buffer object of a specified number of samples of a specified size.</td>
</tr>
<tr>
<td>olDmCopyBuffer</td>
<td>Copies data from the buffer to the specified array.</td>
</tr>
<tr>
<td>olDmCopyFromBuffer</td>
<td>Copies data from the buffer to the specified array.</td>
</tr>
<tr>
<td>olDmCopyToBuffer</td>
<td>Copies data from the specified array to the buffer.</td>
</tr>
<tr>
<td>olDmFreeBuffer</td>
<td>Deletes a buffer object.</td>
</tr>
<tr>
<td>olDmGetBufferPtr</td>
<td>Gets a pointer to the buffer data.</td>
</tr>
<tr>
<td>olDmGetBufferSize</td>
<td>Gets the physical buffer size (in bytes).</td>
</tr>
<tr>
<td>olDmGetDataBits</td>
<td>Gets the number of valid data bits.</td>
</tr>
<tr>
<td>olDmSetDataBits</td>
<td>Sets the number of valid data bits.</td>
</tr>
<tr>
<td>olDmSetDataWidth</td>
<td>Sets the width of each data sample.</td>
</tr>
<tr>
<td>olDmGetDataWidth</td>
<td>Gets the width of each data sample.</td>
</tr>
<tr>
<td>olDmGetErrorString</td>
<td>Gets the string corresponding to a data management error code value.</td>
</tr>
<tr>
<td>olDmGetMaxSamples</td>
<td>Gets the physical size of the buffer (in samples).</td>
</tr>
<tr>
<td>olDmGetTimeDateStamp</td>
<td>Gets the time and date of the buffer's data.</td>
</tr>
<tr>
<td>olDmSetValidSamples</td>
<td>Sets the number of valid samples in the buffer.</td>
</tr>
<tr>
<td>olDmGetValidSamples</td>
<td>Gets the number of valid samples.</td>
</tr>
<tr>
<td>olDmGetVersion</td>
<td>Gets the version of the data management library.</td>
</tr>
<tr>
<td>olDmMallocBuffer</td>
<td>Creates a buffer object of a specified number of bytes.</td>
</tr>
<tr>
<td>olDmReAllocBuffer</td>
<td>Reallocates a buffer object (alloc() interface).</td>
</tr>
</tbody>
</table>
Buffer List Management Functions

Buffer List Management functions, described in Table 10, provide a straightforward mechanism for handling buffer lists, called queues, that the software creates internally as well as other lists that you might want to create. You are not required to use these functions; however, you may find them helpful in your application.

Buffer List Management functions are particularly useful when dealing with a device that acquires or outputs continuous data. Refer to Chapter 5 for more information on queues and other lists.

Table 10: Buffer List Management Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>olDmCreateList</td>
<td>Creates a user-defined list object.</td>
</tr>
<tr>
<td>olDmEnumBuffers</td>
<td>Enumerates all buffers on a queue or on a list you created.</td>
</tr>
<tr>
<td>olDmEnumLists</td>
<td>Enumerates all queues or lists.</td>
</tr>
<tr>
<td>olDmFreeList</td>
<td>Deletes a user-defined list.</td>
</tr>
<tr>
<td>olDmGetBufferFromList</td>
<td>Removes a buffer from the start of a queue or user-defined list.</td>
</tr>
<tr>
<td>olDmGetListCount</td>
<td>Gets the number of buffers on a queue or user-defined list.</td>
</tr>
<tr>
<td>olDmGetListHandle</td>
<td>Finds the queue or user-defined list that a buffer is on.</td>
</tr>
</tbody>
</table>
Table 10: Buffer List Management Functions (cont.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>olDmGetListIds</code></td>
<td>Gets a description of the queue or list.</td>
</tr>
<tr>
<td><code>olDmPeekBufferFromList</code></td>
<td>Gets the handle of the first buffer in the queue or list but does not remove the buffer from the queue or list.</td>
</tr>
<tr>
<td><code>olDmPutBufferToList</code></td>
<td>Adds a buffer to the end of a queue or list.</td>
</tr>
</tbody>
</table>
Using the DataAcq SDK

System Operations .................................................. 55
Analog and Digital I/O Operations .............................. 59
Counter/Timer Operations ........................................ 106
Quadrature Decoder Operations ................................. 138
Simultaneous Startup .............................................. 139
This chapter provides conceptual information to describe the following operations provided by the DataAcq SDK:

- System operations, described starting on page 55
- Analog and digital I/O operations, described starting on page 59
- Counter/timer operations, described starting on page 106
- Simultaneous startup, described starting on page 139

Use this information with the reference information provided in the DataAcq SDK online help when programming your data acquisition devices; refer to page 16 for more information on launching this help file.
System Operations

The DataAcq SDK provides functions for the following general system operations:

- Initializing a device
- Specifying a subsystem
- Configuring a subsystem
- Handling errors
- Handling messages
- Releasing a subsystem and driver

The following subsections describe these operations in more detail.

Initializing a Device

To perform a data acquisition operation, your application program must initialize the device driver for the device you are using with the \texttt{olDaInitialize} function. This function returns a device handle, called \texttt{HDEV}. You need one device handle for each device. Device handles allow you to access more than one device in your system.

If you are unsure of the DT-Open Layers devices in your system, use the \texttt{olDaEnumBoards} function, which lists the device name, device driver name, and system resources used by each DT-Open Layers device in your system, or the \texttt{olDaGetBoardInfo} function, which returns the driver name, model name, and instance number of the specified board, based on its board name.

Once you have initialized a device, you can specify a subsystem, as described in the next section.
Specifying a Subsystem

The DataAcq SDK allows you to define the following subsystems:

- Analog input (A/D subsystem)
- Analog output (D/A subsystem)
- Digital input (DIN subsystem)
- Digital output (DOUT subsystem)
- Counter/timer (C/T subsystem)

A device can have multiple elements of the same subsystem type. Each of these elements is a subsystem of its own and is identified by a subsystem type and element number. Element numbering is zero-based; that is, the first instance of the subsystem is called element 0, the second instance of the subsystem is called element 1, and so on. For example if two digital I/O ports are on your device, two DIN or DOUT subsystems are available, differentiated as element 0 and element 1.

Once you have initialized the device driver for the specified device, you must specify the subsystem/element on the specified device using the `olDaGetDASS` function. This function returns a subsystem handle, called HDASS. To access a subsystem, you need one subsystem handle for each subsystem. Subsystem handles allow you to access more than one subsystem on a device.

If you are unsure of the subsystems on a device, use the `olDaEnumSubSystems` or `olDaGetDevCaps` function. `olDaEnumSubSystems` lists the names, types, and number of elements for all subsystems supported by the specified device. `olDaGetDevCaps` returns the number of elements for a specified subsystem type on a specified device.
Note: You can call any function that contains HDASS as a parameter for any subsystem. In some cases, however, the subsystem may not support the particular capability. If this occurs, the subsystem returns an error code indicating that it does not support that function.

Once you have specified a subsystem/element, you can configure the subsystem and perform a data acquisition operation, as described in the following section.

Configuring a Subsystem

You configure a subsystem by setting its parameters or capabilities. For more information on the capabilities you can query and specify, refer to the following:

- For analog and digital I/O operations, refer to page 59
- For the counter/timer operations, refer to page 106
- For simultaneous operations, refer to page 139

Once you have set up the parameters appropriately for the operation you want to perform, call the `olDaConfig` function to configure the parameters before performing the operation.

Handling Errors

An error code is returned by each function in the DataAcq SDK. An error code of 0 indicates that the function executed successfully (no error). Any other error code indicates that an error occurred. Your application program should check the value returned by each function and perform the appropriate action if an error occurs.
Chapter 3

Refer to the DataAcq SDK online help for detailed information on the returned error codes and how to proceed should they occur.

Handling Messages

The data acquisition device notifies your application of buffer movement and other events by generating messages.

To determine if the subsystem can post messages, use the \texttt{olDaGetSSCaps} function, specifying the capability \texttt{OLSSC_SUP_POSTMESSAGE}. If this function returns a nonzero value, the capability is supported.

Specify the window to receive messages using the \texttt{olDaSetWndHandle} function or the procedure to handle these messages using the \texttt{olDaSetNotificationProcedure} function.

Refer to the DataAcq SDK online help for more information on the messages that can be generated and how to proceed should they occur.

Releasing the Subsystem and the Driver

When you are finished performing data acquisition operations, release the simultaneous start list, if used, using the \texttt{olDaReleaseSSList} function. Then, release each subsystem using the \texttt{olDaReleaseDASS} function. Release the driver and terminate the session using the \texttt{olDaTerminate} function.
Analog and Digital I/O Operations

The DataAcq SDK defines the following capabilities that you can query and/or specify for analog and/or digital I/O operations:

- Data encoding
- Resolution
- Channels (including channel type, channel list, channel inhibit list, and synchronous digital I/O list)
- Ranges
- Gains
- Filters
- Data flow modes
- Triggered scan mode
- Clock sources
- Trigger sources
- Coupling Type
- Excitation current sources and values
- Buffers
- DMA and interrupt resources

The following subsections describe these capabilities in more detail.

Data Encoding

For A/D and D/A subsystems only, the DataAcq SDK defines two data encoding types: binary and twos complement.

To determine the data encoding types supported by the subsystem, use the `olDaGetSSCaps` function, specifying the capability OLSSC_SUP_BINARY for binary data encoding or
OLSSC_SUP_2SCOMP for twos complement data encoding. If this function returns a nonzero value, the capability is supported. Use the \texttt{olDaSetEncoding} function to specify the data encoding type.

**Resolution**

To determine if the subsystem supports software-programmable resolution, use the \texttt{olDaGetSSCaps} function, specifying the capability OLSSC_SUP_SWRESOLUTION. If this function returns a nonzero value, the capability is supported.

To determine the number of resolution settings supported by the subsystem, use the \texttt{olDaGetSSCaps} function, specifying the capability OLSSC_NUMRESOLUTION. To list the actual bits of resolution supported, use the \texttt{olDaEnumSSCaps} function, specifying the OL_ENUM_RESOLUTION capability.

Use the \texttt{olDaSetResolution} function to specify the number of bits of resolution to use for the subsystem.

**Channels**

Each subsystem (or element of a subsystem type) can have multiple channels. To determine how many channels the subsystem supports, use the \texttt{olDaGetSSCaps} function, specifying the OLSSC_NUMCHANNELS capability.

Some subsystems on the DT2896 and DT727 devices also provide channel expansion. To determine if the subsystem supports channel expansion, use the \texttt{olDaGetSSCaps} function, specifying the OLSSC_SUP_EXP2896 capability (for the DT2896) or the OLSSC_SUP_EXP727 capability (for the DT727). If this function returns a nonzero value, the capability is supported.
Specifying the Channel Type

The DataAcq SDK supports the following channel types:

- **Single-ended** – Use this configuration when you want to measure high-level signals, noise is insignificant, the source of the input is close to the device, and all the input signals are referred to the same common ground.

To determine if the subsystem supports the single-ended channel type, use the `olDaGetSSCaps` function, specifying the OLSSC_SUP_SINGLEENDED capability. If this function returns a nonzero value, the capability is supported.

To determine how many single-ended channels are supported by the subsystem, use the `olDaGetSSCaps` function, specifying the OLSSC_MAXSECHANS capability.

Specify the channel type as single-ended for each channel using the `olDaSetChannelType` function.

- **Differential** – Use this configuration when you want to measure low-level signals (less than 1 V), you are using an A/D converter with high resolution (greater than 12 bits), noise is a significant part of the signal, or common-mode voltage exists.

To determine if the subsystem supports the differential channel type, use the `olDaGetSSCaps` function, specifying the OLSSC_SUP_DIFFERENTIAL capability. If this function returns a nonzero value, the capability is supported.

To determine how many differential channels are supported by the subsystem, use the `olDaGetSSCaps` function, specifying the OLSSC_MAXDICHANS capability.

Specify the channel type as differential for each channel using the `olDaSetChannelType` function.
Note: For pseudo-differential analog inputs, specify the single-ended channel type; in this case, how you wire these signals determines the configuration. This option provides less noise rejection than the differential configuration, but twice as many analog input channels.

For older model devices, this setting is jumper-selectable and must be specified in the driver configuration dialog.

The channel list is not used to set the channel type.

The following subsections describe how to specify channels.

**Specifying a Single Channel**

The simplest way to acquire data from or output data to a single channel is to specify the channel for a single-value operation; refer to page 72 for more information on single-value operations.

You can also specify a single channel using a channel list, described in the next section.

**Specifying One or More Channels**

You acquire data from or output data to one or more channels using a channel list.

The DataAcq SDK provides features that allow you to group the channels in the list sequentially (either starting with 0 or with any other analog input channel) or randomly. In addition, the DataAcq SDK allows you to specify a single channel or the same channel more than once in the list. Your device, however, may limit the order in which you can enter a channel in the channel list.
To determine how the channels can be ordered in the channel list for your subsystem, use the `olDaGetSSCaps` function, specifying the OLSSC_RANDOM_CGL capability. If this function returns a nonzero value, the capability is supported; you can order the channels in the channel list in any order, starting with any channel.

If this capability is not supported, use the `olDaGetSSCaps` function, specifying the OLSSC_SUP_SEQUENTIAL_CGL capability. If this function returns a nonzero value, the capability is supported; you must order the channels in the channel list in sequential order, starting with any channel.

If this capability is not supported, use the `olDaGetSSCaps` function, specifying the OLSSC_SUP_ZEROSEQUENTIAL_CGL capability. If this function returns a nonzero value, the capability is supported; you must order the channels in the channel list in sequential order, starting with channel 0.

To determine if the subsystem supports simultaneous sample-and-hold mode use the `olDaGetSSCaps` function, specifying the OLSSC_SUP_SIMULTANEOUS_SH capability. If this function returns a nonzero value, the capability is supported. You must enter at least two channels in the channel list. Generally, the first channel is the sample channel and the remaining channels are the hold channels.

The following subsections describe how to specify channels in a channel list.

**Specifying the Channel List Size**

To determine the maximum size of the channel list for the subsystem, use the `olDaGetSSCaps` function, specifying the OLSSC_CGLDEPTH capability.

Use the `olDaSetChannelListSize` function to specify the size of the channel list.
Note: The OLSSC_CGLDEPTH capability specifies the maximum size of the channel list, channel inhibit list, synchronous digital I/O list, and gain list.

Specifying the Channels in the Channel List

Use the `olDaSetChannelListEntry` function to specify the channels in the channel list in the order you want to sample them or output data from them. (For simultaneous sampling modules, order does not matter, and you cannot enter a particular channel more than once.)

The channels are sampled or output in order from the first entry to the last entry in the channel list. Channel numbering is zero-based; that is, the first entry in the channel list is entry 0, the second entry is entry 1, and so on.

For example, if you want to sample channel 4 twice as frequently as channels 5 and 6, you could program the channel list as follows:

<table>
<thead>
<tr>
<th>Channel-List Entry</th>
<th>Channel</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>Sample channel 4.</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>Sample channel 5.</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>Sample channel 4 again.</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>Sample channel 6.</td>
</tr>
</tbody>
</table>

In this example, channel 4 is sampled first, followed by channel 5, channel 4 again, and then channel 6.
Inhibiting Channels in the Channel List

If supported, you can set up a channel-inhibit list; this feature is useful if you want to discard values acquired from specific channels, as is typical in simultaneous sample-and-hold applications.

To determine if a subsystem supports a channel-inhibit list, use the `olDaGetSSCaps` function, specifying the `OLSSC_SUP_CHANNELLIST_INHIBIT` capability. If this function returns a nonzero value, the capability is supported.

Using the `olDaSetChannelListEntryInhibit` function, you can enable or disable inhibition for each entry in the channel list. If enabled, the acquired value is discarded after the channel entry is sampled; if disabled, the acquired value is stored after the channel entry is sampled.

In the following example, the values acquired from channels 11 and 9 are discarded and the values acquired from channels 10 and 8 are stored.

<table>
<thead>
<tr>
<th>Channel-List Entry</th>
<th>Channel</th>
<th>Channel Inhibit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11</td>
<td>True</td>
<td>Sample channel 11 and discard the value.</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>False</td>
<td>Sample channel 10 and store the value.</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>True</td>
<td>Sample channel 9 and discard the value.</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>False</td>
<td>Sample channel 8 and store the value.</td>
</tr>
</tbody>
</table>
Specifying Synchronous Digital I/O Values in the Channel List

If supported, you can set up a synchronous digital I/O list; this feature is useful if you want to write a digital output value to dynamic digital output channels when an analog input channel is sampled.

To determine if the subsystem supports synchronous (dynamic) digital output operations, use the `olDaGetSSCaps` function, specifying the `OLSSC_SUP_SYNCHRONOUSDIGITALIO` capability. If this function returns a nonzero value, the capability is supported.

Use the `olDaSetSynchronousDigitalIOUsage` function to enable or disable synchronous (dynamic) digital output operation for a specified subsystem.

Once you enable a synchronous digital output operation, specify the values to write to the synchronous (dynamic) digital output channels using the `olDaSetDigitalIOListEntry` function for each entry in the channel list.

To determine the maximum digital output value that you can specify, use the `olDaGetSSCaps` function, specifying the `OLSSC_MAXDIGITALIOLIST_VALUE` capability.

As each entry in the channel list is scanned, the corresponding value in the synchronous digital I/O list is output to the dynamic digital output channels.

In the following example, when channel 7 is sampled, a value of 1 is output to the dynamic digital output channels. When channel 5 is sampled, a value of 1 is output to the dynamic digital output channels. When channels 6 and 4 are sampled, a value of 0 is output to the dynamic digital output channels.
If your device had two dynamic digital output channels and a value of 1 is output (01 in binary format), a value of 1 is written to dynamic digital output channel 0 and a value of 0 is written to dynamic digital output channel 1. Similarly, if a value of 2 is output (10 in binary format), a value of 0 is written to dynamic digital output channel 0 and a value of 1 is written to dynamic digital output channel 1.

<table>
<thead>
<tr>
<th>Channel-List Entry</th>
<th>Channel</th>
<th>Synchronous Digital I/O Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
<td>1</td>
<td>Sample channel 7 and output a value of 1 to the dynamic digital output channels.</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>Sample channel 5 and output a value of 1 to the dynamic digital output channels.</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>0</td>
<td>Sample channel 6 and output a value of 0 to the dynamic digital output channels.</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>0</td>
<td>Sample channel 4 and output a value of 0 to the dynamic digital output channels.</td>
</tr>
</tbody>
</table>

**Note:** If you are controlling sample-and-hold devices with these channels, you may need to program the first channel at the sample logic level and the following channels at the hold logic level; see your device/device driver documentation for details.
Ranges

The range capability applies to A/D and D/A subsystems only.

Note: Some D/A subsystems support both voltage and current output channels. To determine whether your subsystem supports current output channels, use the `olDaGetSSCaps` function, specifying the `OLSSC_SUP_CURRENT_OUTPUT` capability. If this function returns a nonzero value, current output channels are supported.

Depending on your subsystem, you can set the range for the entire subsystem or the range for each channel; the range is typically specified in voltage. (If you are using a current output channel, determine how the voltage range maps to your current output range and write the appropriate voltage to the output channel.)

To determine if the subsystem supports the range-per-channel capability, use the `olDaGetSSCaps` function, specifying the `OLSSC_SUP_RANGEPERCHANNEL` capability. If this function returns a nonzero value, the capability is supported.

To determine how many ranges the subsystem supports, use the `olDaGetSSCaps` function, specifying the `OLSSC_NUMRANGES` capability.

To list the minimum and maximum ranges supported by the subsystem, use the `olDaEnumSSCaps` function, specifying the `OL_ENUM_RANGES` capability.

Use `olDaSetRange` to specify the range for a subsystem. If your subsystem supports the range-per-channel capability, use `olDaSetChannelRange` to specify the range for each channel.
Notes: The channel list is not used to set the range for a channel.

For older device models, the range is jumper-selectable and must be specified in the driver configuration dialog.

Gains

The range divided by the gain determines the effective range for the entry in the channel list. For example, if your device provides a range of ±10 V and you want to measure a ±1.5 V signal, specify a range of ±10 V and a gain of 4; the effective input range for this channel is then ±2.5 V (10/4), which provides the best sampling accuracy for that channel.

The way you specify gain depends on how you specified the channels, as described in the following subsections.

Note: If your device supports autoranging for single-value operations, the device can determine the appropriate gain for your range rather than you having to specify it. Refer to page 72 for more information on autoranging.

Specifying the Gain for a Single Channel

The simplest way to specify gain for a single channel is to specify the gain in a single-value operation; refer to page 72 for more information on single-value operations.

You can also specify the gain for a single channel using a gain list, described in the next section.
Specifying the Gain for One or More Channels

You can specify the gain for one or more channels using a gain list. The gain list parallels the channel list. (The two lists together are often referred to as the channel-gain list or CGL.)

To determine if the subsystem supports programmable gain, use the `olDaGetSSCaps` function, specifying the `OLSSC_SUP_PROGRAMGAIN` capability. If this function returns a nonzero value, the capability is supported.

To determine how many gains the subsystem supports, use the `olDaGetSSCaps` function, specifying the `OLSSC_NUMGAINS` capability.

To list the gains supported by the subsystem, use the `olDaEnumSSCaps` function, specifying the `OL_ENUM_GAINS` capability.

Specify the gain for each entry in the channel list using the `olDaSetGainListEntry` function.

In the following example, a gain of 2 is applied to channel 5, a gain of 4 is applied to channel 6, and a gain of 1 is applied to channel 7.

<table>
<thead>
<tr>
<th>Channel-List Entry</th>
<th>Channel</th>
<th>Gain</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>2</td>
<td>Sample channel 5 using a gain of 2.</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>4</td>
<td>Sample channel 6 using a gain of 4.</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>1</td>
<td>Sample channel 7 using a gain of 1.</td>
</tr>
</tbody>
</table>
Note: If your subsystem does not support programmable gain, enter a value of 1 for all entries.

If your subsystem does not support the gain-per-channel capability, set all entries in the gain list to the same value.

Filters

This capability applies to A/D and D/A subsystems only.

Depending on your subsystem, you can specify a filter for each channel. To determine if the subsystem supports a filter for each channel, use the `olDaGetSSCaps` function, specifying the `OLSSC_SUP_FILTERPERCHAN` capability. If this function returns a nonzero value, the capability is supported.

To determine how many filters the subsystem supports, use the `olDaGetSSCaps` function, specifying the `OLSSC_NUMFILTERS` capability.

To list the cut-off frequency of all filters supported by the subsystem, use the `olDaEnumSSCaps` function, specifying the `OL_ENUM_FILTERS` capability.

If the subsystem supports filtering per channel, specify the filter for each channel using the `olDaSetChannelFilter` function. The filter is equal to or greater than a cut-off frequency that you supply.
Notes: The channel list is not used to set the filter for a channel.

If the subsystem supports more than one filter but does not support a filter per channel, the filter specified for channel 0 is used for all channels.

Data Flow Modes

The DataAcq SDK defines the following data flow modes for A/D, D/A, DIN, and DOUT subsystems:

- Single value
- Continuous (post-trigger, pre-trigger, and about-trigger)

The following subsections describe these data flow modes in detail.

Single-Value Operations

Single-value operations are the simplest to use but offer the least flexibility and efficiency. In a single-value operation, a single data value is read or written at a time. The result is returned immediately. You cannot specify a channel-gain list, clock source, trigger source, DMA channel, or buffer for a single-value operation. Single-value operations stop automatically when finished; you cannot stop a single-value operation manually.

To determine if the subsystem supports single-value operations, use the \texttt{olDaGetSSCaps} function, specifying the capability \texttt{OLSSC_SUP_SINGLEVALUE}. If this function returns a nonzero value, the capability is supported.

Specify the operation mode as \texttt{OL_DF_SINGLEVALUE} using the \texttt{olDaSetDataFlow} function.
Typical Single-Value Operations

Use one of the following functions to read a single value from or write a single-value to a specified channel; the device performs the operation immediately:

- **olDaGetSingleValue** – Acquires a count value from a single input channel using a specified gain, and returns the data as a 32-bit integer.

  If you later want to convert the count value to engineering units, you can use the **olDaCodeToVolts** function. Similarly, if you want to convert the engineering units to counts, you can use the **olDaVoltsToCode** function.

- **olDaGetSingleFloat** – Typically used with thermocouple inputs, acquires a single voltage or temperature value (depending on the thermocouple type) from a single input channel using a specified gain, and returns the data as a floating-point value. Refer to page 92 for more information on thermocouples.

- **olDaGetCjcTemperature** – If you want to correct and linearize thermocouple values in the application rather than in hardware, this function acquires the CJC temperature for a specified input channel, and returns the data as a floating-point value. Refer to page 92 for more information on thermocouple and CJC values.

- **olDaGetSingleValueEx** – Some devices support autoranging for single-value analog input operations, where the device determines the best gain for the specified range. If autoranging is supported, this function allows you to specify the range and analog input channel; the driver then acquires the value from the specified channel using the best gain for the range, and returns the result immediately in both counts and engineering units (such as voltage).

  To determine if the subsystem supports autoranging for single-value operations, use the **olDaGetSSCaps** function, specifying the capability OLSSC_SUP_SINGLEVALUE_AUTORANGE. If this function returns a nonzero value, the capability is supported.
• **olDaPutSingleValue** function – Outputs a single count value to a single output channel using the specified gain.

### Simultaneous Single-Value Operations

Some devices support simultaneous single-value operations, allowing you to read a single value from each input channel of an A/D subsystem or to write a single value to each output channel of a D/A subsystem. To determine if your device supports simultaneous operations, use the **olDaGetSSCaps** function, specifying the capability OLSSC_SUP_SIMULTANEOUS_SH. If this function returns a nonzero value, the capability is supported.

Use one of the following functions to read a single value from or write a single value to all the channels of the subsystem simultaneously; the device performs the operation immediately:

• **olDaGetSingleValues** – Acquires a single count value from each of the channels of the subsystem simultaneously and returns the data as 32-bit integers.

• **olDaGetSingleFloats** – Typically used with thermocouple inputs, acquires a single voltage or temperature value (depending on the thermocouple type) from each of the input channels of the subsystem simultaneously, and returns the data as floating-point values. Refer to page 92 for more information on thermocouples.

• **olDaGetCjcTemperatures** – If you want to correct and linearize thermocouple values in the application rather than in hardware, this function acquires a single CJC temperature from each of the input channels supported by the subsystem, and returns them as floating-point values. Refer to page 92 for more information on thermocouples and CJC values.
• **olDaPutSingleValues** function – Outputs a single count value to each of the channels of the subsystem simultaneously, using the specified gain. If you do not want to update a particular output channel, specify the constant DONT_UPDATE for the channel value; in this case, the channel maintains the last value that was written to it.

**Continuous Operations**

For a continuous operation, you can specify any supported subsystem capability, including a channel-gain list, clock source, trigger source, pre-trigger source, retrigger source, DMA channel, and buffer.

Call the **olDaStart** function to start a continuous operation.

To stop a continuous operation, perform either an orderly stop using the **olDaStop** function or an abrupt stop using the **olDaAbort** or **olDaReset** function.

In an orderly stop (**olDaStop**), the device finishes acquiring the specified number of samples, stops all subsequent acquisition, and transfers the acquired data to a buffer on the done queue; all subsequent triggers or retriggers are ignored. (Refer to page 95 for more information on buffers and queues.)

In an abrupt stop (**olDaAbort**), the device stops acquiring samples immediately; the acquired data is transferred to a buffer and put on the done queue; however, the buffer may not be completely filled. All subsequent triggers or retriggers are ignored.

The **olDaReset** function reinitializes the subsystem after stopping it abruptly.
Note: For analog output operations, you can also stop the operation by not sending new data to the device. The operation stops when no more data is available.

Some subsystems also allow you to pause the operation using the `olDaPause` function and to resume the paused operation using the `olDaContinue` function. To determine if pausing is supported, use the `olDaGetSSCaps` function, specifying the OLSSC_SUP_PAUSE capability. If this function returns a nonzero value, the capability is supported.

Note: To determine whether a subsystem is currently running, use the `olDaIsRunning` function. If TRUE is returned, the subsystem is currently running; if FALSE is returned, the subsystem is not currently running.

The following continuous modes are supported by the DataAcq SDK: continuous (post-trigger), continuous pre-trigger, and continuous about-trigger. These modes are described in the following subsections.

**Continuous Post-Trigger Mode**

Use continuous post-trigger when you want to acquire or output data continuously when a trigger occurs.

To determine if the subsystem supports continuous (post-trigger) operations, use the `olDaGetSSCaps` function, specifying the capability OLSSC_SUP_CONTINUOUS. If this function returns a nonzero value, the capability is supported.
For continuous (post-trigger) mode, specify the operation mode as OL_DF_CONTINUOUS using the `olDaSetDataFlow` function.

Use the `olDaSetTrigger` function to specify the trigger source that starts the operation. Refer to page 88 for more information on supported trigger sources.

When the post-trigger event is detected, the device cycles through the channel list, acquiring and/or outputting the value for each entry in the channel list; this process is defined as a scan. The device then wraps to the start of the channel list and repeats the process continuously until either the allocated buffers are filled or you stop the operation. Refer to page 62 for more information on channel lists; refer to page 95 for more information on buffers.

Figure 1 illustrates continuous post-trigger mode using a channel list of three entries: channel 0, channel 1, and channel 2. In this example, post-trigger analog input data is acquired on each clock pulse of the A/D sample clock; refer to page 85 for more information on clock sources. The device wraps to the beginning of the channel list and repeats continuously.

**Figure 1: Continuous Post-Trigger Mode**
Continuous Pre-Trigger Mode

Use continuous pre-trigger mode when you want to acquire data before a specific external event occurs.

To determine if the subsystem supports continuous pre-trigger mode, use the `olDaGetSSCaps` function, specifying the `_SUP_CONTINUOUS_PRETRIG` capability. If this function returns a nonzero value, the capability is supported.

Specify the operation mode as `OL_DF_CONTINUOUS_PRETRIG` using the `olDaSetDataFlow` function.

Pre-trigger acquisition starts when the device detects the pre-trigger source and stops when the device detects an external post-trigger source, indicating that the first post-trigger sample was acquired (this sample is ignored).

Use the `olDaSetPretriggerSource` function to specify the trigger source that starts the pre-trigger operation (generally this is a software trigger). Specify the post-trigger source that stops the operation using `olDaSetTrigger`. Refer to page 88 and to your device/driver documentation for supported pre-trigger and post-trigger sources.

Figure 2 illustrates continuous pre-trigger mode using a channel list of three entries: channel 0, channel 1, and channel 2. In this example, pre-trigger analog input data is acquired on each clock pulse of the A/D sample clock; refer to page 85 for more information on clock sources. The device wraps to the beginning of the channel list and the acquisition repeats continuously until the post-trigger event occurs. When the post-trigger event occurs, acquisition stops.
Continuous About-Trigger Mode

Use continuous about-trigger mode when you want to acquire data both before and after a specific external event occurs. This operation is equivalent to doing both a pre-trigger and a post-trigger acquisition.

To determine if the subsystem supports continuous about-trigger mode, use the `olDaGetSSCaps` function, specifying the OLSSC_SUP_CONTINUOUS_ABOUTTRIG capability. If this function returns a nonzero value, the capability is supported.

Specify the operation mode as OL_DF_CONTINUOUS_ABOUTTRIG using the `olDaSetDataFlow` function.

The about-trigger acquisition starts when the device detects the pre-trigger source. When it detects an external post-trigger source, the device stops acquiring pre-trigger data and starts acquiring post-trigger data.

Figure 2: Continuous Pre-Trigger Mode
Use the `oIDaSetPretriggerSource` function to specify the pre-trigger source that starts the pre-trigger operation (this is generally a software trigger) and `oIDaSetTrigger` to specify the trigger source that stops the pre-trigger acquisition and starts the post-trigger acquisition. Refer to page 88 and to your device/driver documentation for supported pre-trigger and post-trigger sources.

The about-trigger operation stops when the specified number of post-trigger samples has been acquired or when you stop the operation.

Figure 3 illustrates continuous about-trigger mode using a channel list of three entries: channel 0, channel 1, and channel 2. In this example, pre-trigger analog input data is acquired on each clock pulse of the A/D sample clock. The device wraps to the beginning of the channel list and the acquisition repeats continuously until the post-trigger event occurs. When the post-trigger event occurs, post-trigger acquisition begins on each clock pulse of the A/D sample clock; refer to page 85 for more information on clock sources. The device wraps to the beginning of the channel list and acquires post-trigger data continuously.

![Figure 3: Continuous About-Trigger Mode](image-url)
Triggered Scan Mode

In triggered scan mode, the device scans the entries in a channel-gain list a specified number of times when it detects the specified trigger source, acquiring the data for each entry that is scanned.

To determine if the subsystem supports triggered scan mode, use the `olDaGetSSCaps` function, specifying the OLSSC_SUP_TRIGSCAN capability. If this function returns a nonzero value, the capability is supported. Note that you cannot use triggered scan mode with single-value operations.

To enable (or disable) triggered scan mode, use the `olDaSetTriggeredScanUsage` function.

To determine the maximum number of times that the device can scan the channel-gain list per trigger, use the `olDaGetSSCaps` function, specifying the OLSSC_MAXMULTISCAN capability.

Use the `olDaSetMultiscanCount` function to specify the number of times to scan the channel-gain list per trigger.

The DataAcq SDK defines the following retrigger modes for a triggered scan; these retrigger modes are described in the following subsections:

- Scan-per-trigger
- Internal retrigger
- Retrigger extra

**Note:** If your device driver supports it, retrigger extra is the preferred triggered scan mode.
**Scan-Per-Trigger Mode**

Use scan-per-trigger mode if you want to accurately control the period between conversions of individual channels and retrigger the scan based on an internal or external event. In this mode, the retrigger source is the same as the initial trigger source.

To determine if the subsystem supports scan-per-trigger mode, use the `olDaGetSSCaps` function, specifying the OLSSC_SUP_RETRIGGER_SCAN_PER_TRIGGER capability. If this function returns a nonzero value, the capability is supported.

Specify the retrigger mode as scan-per-trigger using the `olDaSetRetriggerMode` function.

When it detects an initial trigger (post-trigger mode only), the device scans the channel-gain list a specified number of times (determined by the `olDaSetMultiscanCount` function), then stops. When the external retrigger occurs, the process repeats.

The conversion rate of each channel in the scan is determined by the frequency of the A/D sample clock; refer to page 85 for more information on clock sources. The conversion rate of each scan is determined by the period between retriggers; therefore, it cannot be accurately controlled. The device ignores external triggers that occur while it is acquiring data. Only retrigger events that occur when the device is waiting for a trigger are detected and acted on. Some devices may generate an OLDA_WM_TRIGGER_ERROR message.
**Internal Retrigger Mode**

Use internal retrigger mode if you want to accurately control both the period between conversions of individual channels in a scan and the period between each scan.

To determine if the subsystem supports internal retrigger mode, use the `olDaGetSSCaps` function, specifying the OLSSC_SUP_RETRIGGER_INTERNAL capability. If this function returns a nonzero value, the capability is supported.

Specify the retrigger mode as internal using the `olDaSetRetriggerMode` function.

The conversion rate of each channel in the scan is determined by the frequency of the A/D sample clock; refer to page 85 for more information on clock sources. The conversion rate between scans is determined by the frequency of the internal retrigger clock on the device. You specify the frequency on the internal retrigger clock using the `olDaSetRetriggerFrequency` function.

When it detects an initial trigger (pre-trigger source or post-trigger source), the device scans the channel-gain list a specified number of times (determined by the `olDaSetMultiscanCount` function), then stops. When the internal retrigger occurs, determined by the frequency of the internal retrigger clock, the process repeats.
We recommend that you set the retrigger frequency as follows:

\[
\text{Min. Retrigger} = \frac{\# \text{ of CGL entries} \times \# \text{ of CGLs per trigger}}{\text{A/D sample clock frequency}} + 2 \mu s
\]

\[
\text{Max. Retrigger} = \frac{1}{\text{Frequency Min. Retrigger Period}}
\]

For example, if you are using 512 channels in the channel-gain list (CGL), scanning the channel-gain list 256 times every trigger or retrigger, and using an A/D sample clock with a frequency of 1 MHz, set the maximum retrigger frequency to 7.62 Hz, since

\[
7.62 \text{ Hz} = \frac{1}{\frac{(512 \times 256) + 2 \mu s}{1 \text{ MHz}}}
\]

**Retrigger Extra Mode**

Use retrigger extra mode if you want to accurately control the period between conversions of individual channels and retrigger the scan on a specified retrigger source; the retrigger source can be any of the supported trigger sources.

To determine if the subsystem supports retrigger extra mode, use the `olDaGetSSCaps` function, specifying the `OLSSC_SUP_RETRIGGER_EXTRA` capability. If this function returns a nonzero value, the capability is supported.

Specify the retrigger mode as retrigger extra using the `olDaSetRetriggerMode` function.

Use the `olDaSetRetrigger` function to specify the retrigger source. Refer to page 88 and to your device/device driver documentation for supported retrigger sources.
The conversion rate of each channel in the scan is determined by the frequency of the A/D sample clock; refer to page 85 for more information on clock sources. The conversion rate of each scan is determined by the period between retrigger events.

If you are using an internal retrigger, specify the period between retrigger events using \texttt{olDaSetRetriggerFrequency} (see page 83). If you are using an external retrigger, the period between retrigger events cannot be accurately controlled. The device ignores external retrigger events that occur while it is acquiring data. Only retrigger events that occur when the device is waiting for a trigger are detected and acted on. Some devices may generate an \texttt{OLDA_WM_TRIGGER_ERROR} message.

\section*{Clock Sources}

The DataAcq SDK defines internal, external, and extra clock sources, described in the following subsections. Note that you cannot specify a clock source for single-value operations.

\begin{center}
\textbf{Note: } Some subsystems allow you to read or update multiple channels on a single clock pulse. You can determine whether multiple channels are read or updated on a single clock pulse by using the query \texttt{OLSSC_SUP_SIMULTANEOUS_CLOCKING}.
\end{center}

\subsection*{Internal Clock Source}

The internal clock is the clock source on the device that paces data acquisition or output for each entry in the channel-gain list.

To determine if the subsystem supports an internal clock, use the \texttt{olDaGetSSCaps} function, specifying the \texttt{OLSSC_SUP_INTCLOCK} capability. If this function returns a nonzero value, the capability is supported.
Specify the clock source as internal using the `olDaSetClockSource` function. Then, use the `olDaSetClockFrequency` function to specify the frequency at which to pace the operation.

To determine the maximum frequency that the subsystem supports, use the `olDaGetSSCapsEx` function, specifying the `OLSSCE_MAXTHROUGHPUT` capability. To determine the minimum frequency that the subsystem supports, use the `olDaGetSSCapsEx` function, specifying the `OLSSCE_MINTHROUGHPUT` capability.

**Note:** According to sampling theory (Nyquist Theorem), you should specify a frequency for an A/D signal that is at least twice as fast as the input’s highest frequency component. For example, to accurately sample a 20 kHz signal, specify a sampling frequency of at least 40 kHz. Doing so avoids an error condition called *aliasing*, in which high frequency input components erroneously appear as lower frequencies after sampling.

**External Clock Source**

The external clock is a clock source attached to the device that paces data acquisition or output for each entry in the channel-gain list. This clock source is useful when you want to pace at rates not available with the internal clock or if you want to pace at uneven intervals.

To determine if the subsystem supports an external clock, use the `olDaGetSSCaps` function, specifying the `OLSSC_SUP_EXTCLOCK` capability. If this function returns a nonzero value, the capability is supported.
Specify the clock source as external using the \texttt{olDaSetClockSource} function. Then, use the \texttt{olDaSetExternalClockDivider} to specify the clock divider used to determine the frequency at which to pace the operation; the clock input source divided by the clock divider determines the frequency of the clock signal.

To determine the maximum clock divider that the subsystem supports, use the \texttt{olDaGetSSCapsEx} function, specifying the \texttt{OLSSCE_MAXCLOCKDIVIDER} capability. To determine the minimum clock divider that the subsystem supports, use the \texttt{olDaGetSSCapsEx} function, specifying the \texttt{OLSSCE_MINCLOCKDIVIDER} capability.

\textbf{Extra Clock Source}

Your device driver may define extra clock sources that you can use to pace acquisition or output operations.

To determine how many extra clock sources are supported by your subsystem, use the \texttt{olDaGetSSCaps} function, specifying the \texttt{OLSSC_NUMEXTRACLOCKS} capability. Refer to your device/driver documentation for a description of the extra clock sources.

The extra clock sources may be internal or external. Refer to the previous sections for information on how to specify internal and external clocks and their frequencies or clock dividers.
Trigger Sources

The DataAcq SDK defines the following trigger sources:

- Software (internal) trigger
- External digital (TTL) trigger
- External analog threshold (positive) trigger
- External analog threshold (negative) trigger
- Analog event trigger
- Digital event trigger
- Timer event trigger
- Extra trigger

To specify a post-trigger source, use the `olDaSetTrigger` function; refer to page 76 for more information. To specify a pre-trigger source, use the `olDaSetPretriggerSource` function; see page 78 for more information. To specify a retrigger source, use the `olDaSetRetrigger` function; see page 84 for more information.

The following subsections describe these trigger sources. Note that you cannot specify a trigger source for single-value operations.

**Software (Internal) Trigger Source**

A software trigger occurs when you start the operation; internally, the computer writes to the device to begin the operation.

To determine if the subsystem supports a software trigger, use the `olDaGetSSCaps` function, specifying the capability `OLSSC_SUP_SOFTTRIG`. If this function returns a nonzero value, the capability is supported.
External Digital (TTL) Trigger Source

An external digital trigger is a digital (TTL) signal attached to the device.

To determine if the subsystem supports an external digital trigger for a continuous operation, use the `olDaGetSSCaps` function, specifying the capability OLSSC_SUP_EXTERNTRIG. If this function returns a nonzero value, the capability is supported.

To determine if the subsystem supports a positive, external digital trigger for a single-value operation, use the `olDaGetSSCaps` function, specifying the capability OLSSC_SUP_SV_POS_EXTERN_TTLTRIG. If this function returns a nonzero value, the capability is supported.

To determine if the subsystem supports a negative, external digital trigger for a single-value operation, use the `olDaGetSSCaps` function, specifying the capability OLSSC_SUP_SV_NEG_EXTERN_TTLTRIG. If this function returns a nonzero value, the capability is supported.

External Analog Threshold (Positive) Trigger Source

An external analog threshold (positive) trigger is generally either an analog signal from an analog input channel or an external analog signal attached to the device. An analog trigger occurs when the device detects a transition from a negative to positive value that crosses a threshold value. The threshold level is generally set using a D/A subsystem on the device.

To determine if the subsystem supports analog threshold triggering (positive polarity), use the `olDaGetSSCaps` function, specifying the capability OLSSC_SUP_THRESHTRIGPOS. If this function returns a nonzero value, the capability is supported.
Refer to your device/device driver documentation for a description of this trigger source.

**External Analog Threshold (Negative) Trigger Source**

An external analog threshold (negative) trigger is generally either an analog signal from an analog input channel or an external analog signal attached to the device. An analog trigger event occurs when the device detects a transition from a positive to negative value that crosses a threshold value. The threshold level is generally set using a D/A subsystem on the device.

To determine if the subsystem supports analog threshold triggering (negative polarity), use the `olDaGetSSCaps` function, specifying the capability `OLSSC_SUP_THRESHTRIGNEG`. If this function returns a nonzero value, the capability is supported.

Refer to your device/device driver documentation for a description of this trigger source.

**Analog Event Trigger Source**

For this trigger source, a trigger is generated when an analog event occurs. To determine if the subsystem supports an analog event trigger, use the `olDaGetSSCaps` function, specifying the capability `OLSSC_SUP_ANALOGEVENTTRIG`. If this function returns a nonzero value, the capability is supported.

**Digital Event Trigger Source**

For this trigger source, a trigger is generated when a digital event occurs. To determine if the subsystem supports a digital event trigger, use the `olDaGetSSCaps` function, specifying the capability `OLSSC_SUP_DIGITALEVENTTRIG`. If this function returns a nonzero value, the capability is supported.
**Timer Event Trigger Source**

For this trigger source, a trigger is generated when a counter/timer event occurs. To determine if the subsystem supports a timer event trigger, use the `olDaGetSSCaps` function, specifying the capability OLSSC_SUP_TIMEREVENTTRIG. If this function returns a nonzero value, the capability is supported.

**Extra Trigger Source**

Extra trigger sources may be defined by your device driver. To determine how many extra triggers are supported by the subsystem, use the `olDaGetSSCaps` function, specifying the capability OLSSC_NUMEXTRATRIGGERS. Refer to your device/driver documentation for a description of these triggers.

The extra trigger sources may be internal or external. Refer to the previous sections for information on how to specify internal and external triggers.

**Coupling Type**

Some analog input subsystems provide programmable AC or DC coupling to support IEPE inputs. To determine if AC coupling is supported by the subsystem, use the `olDaGetSSCaps` function, specifying the capability OLSSC_SUP_AC_COUPLING. To determine if DC coupling is supported by the subsystem, use the `olDaGetSSCaps` function, specifying the capability OLSSC_SUP_DC_COUPLING.

To set the coupling type, use the `olDaSetCouplingType` function.
Excitation Current Sources and Values

Some analog input subsystems provide a programmable internal or external excitation current source to support IEPE inputs. To determine if the subsystem supports an internal excitation current source, use the `olDaGetSSCaps` function, specifying the capability `OLSSC_SUP_INTERNAL_EXCITATION_CURRENT_SRC`. To determine if the subsystem supports an external excitation current source, use the `olDaGetSSCaps` function, specifying the capability `OLSSC_SUP_EXTERNAL_EXCITATION_CURRENT_SRC`.

To set the excitation current source, use the `olDaSetExcitationCurrentSource` function.

If you specify an internal excitation current source, you can set the value of the current source using the `olDaSetExcitationCurrentValue` function. If the specified value is not supported by the device, an error is reported.

To determine how many values the subsystem supports for the internal excitation current source, use the `OLSSC_NUM_EXCITATION_CURRENT_VALUES` capability with the `olDaGetSSCaps` function.

To determine the actual values that are available for the internal excitation current source, query the subsystem using the `olDaEnumSSCaps` function with the `OL_ENUM_EXCITATION_CURRENT_VALUES` capability.

Thermocouples

Some A/D subsystems support thermocouple inputs. To determine if your subsystem supports thermocouple inputs, use the `olDaGetSSCaps` function, specifying the `OLSSC_SUP_THERMOCOUPLES` capability.
Thermocouple Input Types

If the subsystem supports thermocouple inputs, specify the type of thermocouple that is connected to the input channel using the `oDaSetThermocoupleType` function. The following thermocouple types are defined:

- `OL_THERMOCOUPLE_TYPE_NONE` – Specifies voltage rather than temperature
- `OL_THERMOCOUPLE_TYPE_J` – Specifies a J thermocouple type
- `OL_THERMOCOUPLE_TYPE_K` – Specifies a K thermocouple type
- `OL_THERMOCOUPLE_TYPE_B` – Specifies a B thermocouple type
- `OL_THERMOCOUPLE_TYPE_E` – Specifies a E thermocouple type
- `OL_THERMOCOUPLE_TYPE_N` – Specifies a N thermocouple type
- `OL_THERMOCOUPLE_TYPE_R` – Specifies a R thermocouple type
- `OL_THERMOCOUPLE_TYPE_S` – Specifies a S thermocouple type
- `OL_THERMOCOUPLE_TYPE_T` – Specifies a T thermocouple type

If the thermocouple type is set to `OL_THERMOCOUPLE_TYPE_NONE`, data is returned in voltage rather than temperature. If the thermocouple type is set for any of the other defined thermocouple types, the data is returned in degrees C.

On some devices, thermocouple data is returned as integer values. On other devices, thermocouple data is returned as floating-point values (4 bytes). To determine if your subsystem returns
floating-point values, use the `olDaGetSSCaps` function, specifying the capability OLSSC_RETURNSFLOATS.

**Thermocouple Correction and Linearization**

Some devices do thermocouple correction and linearization in hardware based on the value of an internal CJC (cold junction compensation) channel. Every sample in the data stream corresponds to a single (typically, floating-point) value that represents either the temperature (in degrees C) or the voltage of the input channel, based on its thermocouple type.

Other devices return A/D input values in raw counts and the application program is responsible for correcting and linearizing the data based on the value of the CJC channel.

To determine if your subsystem does thermocouple correction and linearization in hardware, use the `olDaGetSSCaps` function, specifying the capability OLSSC_SUP_TEMPERATURE_DATA_IN_STREAM. If this query returns a nonzero value, correction and linearization is done in hardware. If this query returns a value of 0, your application program is responsible for correcting and linearizing thermocouple values.

Use the `olDaGetSSCaps` function, specifying the capability OLSSC_SUP_CJC_SOURCE_INTERNAL, to determine whether the CJC value is measured internally on the device (rather than using one of the analog input channels).

To determine if one of the analog input channels is used as the CJC channel, use the `olDaGetSSCaps` function, specifying the capability OLSSC_SUP_CJC_SOURCE_CHANNEL.
Note: Some devices that support correcting and linearizing thermocouple data in hardware also provide the option of returning CJC values in the data stream. This option is seldom used, but is provided if you want to linearize thermocouple values in your application (rather than in hardware) when using continuous operations.

To determine if the subsystem supports interleaving CJC values with A/D values in the data stream, use the `olDaGetSSCaps` function, specifying the capability `OLSSC_SUP_INTERLEAVED_CJC_IN_STREAM`.

By default, the subsystem is disabled from returning CJC values in the data stream. To enable the subsystem to return CJC values in the data stream, use the `olDaSetReturnCjcTemperatureInStream` function. When enabled, two (typically floating-point) values are returned in the data stream: the first value represents the temperature or voltage of the input channel (based on the thermocouple type of the input), and the second value represents the CJC temperature, in degrees C. Generally, in this configuration, a thermocouple type of `OL_THERMOCOUPLE_TYPE_NONE` is specified for each channel to allow direct linearization of voltage values into temperature.

---

**Buffers**

The buffering capability usually applies to A/D and D/A subsystems only. Note that you cannot use a buffer with single-value operations.

A data buffer is a memory location that you allocate in host memory. This memory location is used to store data for continuous input and output operations.
To determine if the subsystem supports buffers, use the `olDaGetSSCaps` function, specifying the capability `OLSSC_SUP_BUFFERING`. If this function returns a nonzero value, the capability is supported.

Buffers are stored on one of three queues: the ready queue, the inprocess queue, or the done queue. These queues are described in more detail in the following subsections.

**Ready Queue**

For input operations, the ready queue holds buffers that are empty and ready for input. For output operations, the ready queue holds buffers that you have filled with data and that are ready for output.

Allocate the buffers using the `olDmMallocBuffer`, `olDmAllocBuffer`, or `olDmCallocBuffer` function. `olDmAllocBuffer` allocates a buffer of samples, where each sample is 2 bytes; `olDmCallocBuffer` allocates a buffer of samples of a specified size; `olDmMallocBuffer` allocates a buffer in bytes.

For analog input operations, it is recommended that you allocate a minimum of three buffers; for analog output operations, you can allocate one or more buffers. The size of the buffers should be at least as large as the sampling or output rate; for example, if you are using a sampling rate of 100 ksamples/s (100 kHz), specify a buffer size of 100,000 samples.

Once you have allocated the buffers (and, for output operations, filled them with data), put the buffers on the ready queue using the `olDaPutBuffer` function.

For example, assume that you are performing an analog input operation, that you allocated three buffers, and that you put these buffers on the ready queue. The queues appear on the ready queue as shown in Figure 4.
When you start a continuous (post-trigger, pre-trigger, or about-trigger) operation, the data acquisition device takes the first available buffer from the ready queue and places it on the inprocess queue.

The inprocess queue holds the buffer that the specified data acquisition device is currently filling (for input operations) or outputting (for output operations). The buffer is filled or emptied at the specified clock rate.

Continuing with the previous example, when you start the analog input operation, the driver takes the first available buffer (Buffer 1, in this case), puts it on the inprocess queue, and starts filling it with data. The queues appear as shown in Figure 5.
If required, you can use the `olDaFlushFromBufferInprocess` function to transfer data from a partially-filled buffer on an inprocess queue to a buffer you create (if this capability is supported). Typically, you would use this function when your data acquisition operation is running slowly.

To determine if the subsystem supports transferring data from a buffer on the inprocess queue, use the `olDaGetSSCaps` function, specifying the `OLSSC_SUP_INPROCESSFLUSH` capability. If this function returns a nonzero value, this capability is supported.

**Note:** Some devices transfer data to the host in segments instead of one sample at a time. For example, data from the DT3010 device is transferred to the host in 64 byte segments; the number of valid samples is always a multiple of 64 depending on the number of samples transferred to the host when `olDaFlushFromBufferInprocess` was called. It is up to your application to take this into account when flushing an inprocess buffer. Refer to your device documentation for more information.
**Done Queue**

Once the data acquisition device has filled the buffer (for input operations) or emptied the buffer (for output operations), the buffer is moved from the inprocess queue to the done queue. Then, either the OLDA_WM_BUFFER_DONE message is generated when the buffer contains post-trigger data, or in the case of pre-trigger and about-trigger acquisitions, an OLDA_WM_PRETRIGGER_BUFFER_DONE message is generated when the buffer contains pre-trigger data.

---

**Note:** For pre-trigger acquisitions only, when the operation completes or you stop a pre-trigger acquisition, the OLDA_WM_QUEUE_STOPPED message is also generated.

For analog output operations only, the OLDA_WM_IO_COMPLETE message is generated when the last data point has been output from the analog output channel. In some cases, this message is generated well after the data is transferred from the buffer (when the OLDA_WM_BUFFER_DONE and OLDA_WM_QUEUE_DONE messages are generated.

---

Continuing with the previous example, the queues appear as shown in Figure 6 when you get the first OLDA_WM_BUFFER_DONE message.
Figure 6: Example of the Done Queue

Then, the driver moves Buffer 2 from the ready queue to the inprocess queue and starts filling it with data. When Buffer 2 is filled, Buffer 2 is moved to the done queue and another OLDA_WM_BUFFER_DONE message is generated.

The driver then moves Buffer 3 from the ready queue to the inprocess queue and starts filling it with data. When Buffer 3 is filled, Buffer 3 is moved to the done queue and another OLDA_WM_BUFFER_DONE message is generated. Figure 7 shows how the buffers are moved.

Figure 7: How Buffers are Moved to the Done Queue
If you transferred data from an inprocess queue to a new buffer using \texttt{olDaFlushFromBufferInprocess}, the new buffer is put on the done queue for your application to process. When the buffer on the inprocess queue finishes being filled, this buffer is also put on the done queue; the buffer contains only the samples that were not previously transferred.

\textbf{Buffer and Queue Management}

Each time it gets an OLDA\_WM\_BUFFER\_DONE message, your application program should remove the buffers from the done queue using the \texttt{olDaGetBuffer} buffer management function.

Your application program can then process the data in the buffer. For an input operation, you can copy the data from the buffer to an array in your application program using the \texttt{olDmGetBufferPtr} function. For continuously paced analog output operations, you can fill the buffer with new output data using the \texttt{olDaGetBufferPtr} function.

If you want to convert the count value to engineering units, you can use the \texttt{olDaCodeToVolts} function. Similarly, if you want to convert the engineering units to counts, you can use the \texttt{olDaVoltsToCode} function.

When you are finished processing the data, you can put the buffer back on the ready queue using the \texttt{olDaPutBuffer} function if you want your operation to continue.

For example, assume that you processed the data from Buffer 1 and put it back on the ready queue. The queues would appear as shown in Figure 8.
When the data acquisition operation is finished, use the `olDaFlushBuffers` function to transfer any data buffers left on the subsystem’s ready queue to the done queue.

Once you have processed the data in the buffers, remove the buffers from the done queue using the `olDaFreeBuffer` function.

**Buffer Wrap Modes**

Most Data Translation data acquisition devices can provide gap-free data, meaning no samples are missed when data is acquired or output. You can acquire gap-free data by manipulating data buffers so that no gaps exist between the last sample of the current buffer and the first sample of the next buffer.

**Note:** The number of DMA channels, number of buffers, and buffer size are critical to the device’s ability to provide gap-free data. It is also critical that the application process the data in a timely fashion.
If you want to acquire gap-free input data, we recommend that you specify a buffer wrap mode of none using the `olDaSetWrapMode` buffer management function. When a buffer wrap mode of none is selected, the operation continues indefinitely if you process the buffers and put them back on the ready queue in a timely manner. When no buffers are available on the ready queue, the operation stops, and an OLDA_WM_QUEUE_DONE message is generated.

If you want to continuously reuse the buffers in the queues and you are not concerned with gap-free data, specify `multiple` buffer wrap mode using `olDaSetWrapMode`. When multiple wrap mode is selected and no buffers are available on the ready queue, the driver overwrites the data in the current buffer. This process continues indefinitely until you stop it. When it reuses a buffer on the done queue, the driver generates an OLDA_WM_BUFFER_REUSED message.

If you want to output gap-free waveforms from your analog output channels, specify `single` wrap mode using `olDaSetWrapMode`. When single wrap mode is specified, a single buffer is used.

If the device has a FIFO and the buffer fits into the FIFO, the buffer is downloaded to the FIFO on the device. The driver (or device) outputs the data starting from the first location in the buffer. When it reaches the end of the buffer, the driver (or device) continues outputting data from the first location of the buffer, and the process continues indefinitely until you stop it. Typically, no messages are posted in this mode until you stop the operation.

**Note:** If the size of your buffer is less than the FIFO size and you stop the analog output operation, the operation stops after the current buffer and the next buffer have been output.
If you query the subsystem with the `OLSSC_SUP.WRAPWAVEFORM_ONLY` capability and the `olDaGetSSCaps` function returns a nonzero value, the subsystem supports waveform-based operations using the onboard FIFO only. In this case, the buffer wrap mode must be set to single. In addition, the buffer size must be less than or equal to the FIFO size. You can determine whether the subsystem supports a FIFO using the `olDaGetSSCaps` function with the capability `OLSSC_SUP_FIFO`. If this function returns a nonzero value, a FIFO is supported. You can determine the size of the FIFO using the capability `OLSSC_FIFO_SIZE_IN_K`. This query returns the actual FIFO size in kilobytes.

To determine the buffer wrap modes available for the subsystem, use the `olDaGetSSCaps` function, specifying the capability `OLSSC_SUP.WRAPSINGLE` (for single wrap mode) or `OLSSC_SUP.WRPMULTIPLE` (for multiple wrap mode). If this function returns a nonzero value, the capability is supported.

**DMA and Interrupt Resources**

You cannot use DMA or interrupt resources for single-value operations.

To determine if your subsystem supports interrupt resources, use the `olDaGetSSCaps` function, specifying the capability `OLSSC_SUP_INTERRUPT`. If this function returns a nonzero value, the capability is supported.

Generally, you specify interrupt resources on the device itself or in the driver configuration dialog.
To determine if gap-free data acquisition is supported, use the \texttt{olDaGetSSCaps} function, specifying \texttt{OLSSC_SUP_GAPFREE_NODMA} (for gap free data using no DMA channels), \texttt{OLSSC_SUP_GAPFREE_SINGLEDMA} (for gap free data using one DMA channel), or \texttt{OLSSC_SUP_GAPFREE_DUALDMA} (for gap free data using two DMA channels). If this function returns a nonzero value, the capability is supported.

To determine how many DMA channels are supported, use the \texttt{olDaGetSSCaps} function, specifying the capability \texttt{OLSSC_NUMDMACHANS}.

Use the \texttt{olDaSetDmaUsage} function to specify the number of DMA channels to use. These channels must also be specified in the driver configuration dialog.

\textbf{Note: } DMA channels are a limited resource and the request may not be honored if the requested number of channels is unavailable. For example, suppose that a device that supports both A/D and D/A subsystems provides hardware for two DMA channels, and that one DMA channel is currently allocated to the A/D subsystem. In this case, a request to the D/A subsystem to use two DMA channels will fail.
Counter/Timer Operations

The counter/timer subsystem supports general-purpose user counter/timers and may support quadrature decoders. Refer to page 138 for more information on quadrature decoders. The rest of this section describes the operation of general-purpose counter/timers.

User Counter/Timers

Each user counter/timer channel accepts a clock input signal and gate input signal and outputs a clock output signal (also called a pulse output signal), as shown in Figure 9.

Each counter/timer channel corresponds to a counter/timer (C/T) subsystem. To specify the counter to use in software, specify the appropriate C/T subsystem. For example, counter 0 corresponds to C/T subsystem element 0; counter 3 corresponds to C/T subsystem element 3.
The DataAcq SDK defines the following capabilities that you can query and/or configure for user counter/timer operations:

- Counter/timer operation mode
- Clock source
- Gate source
- Pulse output type
- Pulse output duty cycle

The following sections describe these capabilities in more detail.

**Counter/Timer Operation Mode**

The DataAcq SDK supports the following counter/timer operations:

- Event counting
- Up/down counting
- Frequency measurement
- Edge-to-edge measurement
- Continuous edge-to-edge measurement
- Rate generation (continuous pulse output)
- One-shot
- Repetitive one-shot

The following subsections describe these counter/timer operations.
**Event Counting**

Use event counting mode to count events from the counter’s associated clock input source.

To determine if the subsystem supports event counting, use the **olDaGetSSCaps** function, specifying the capability OLSSC_SUP_CTMODE_COUNT. If this function returns a nonzero value, the capability is supported.

To specify an event counting operation, use the **olDaSetCTMode** function, specifying the OL_CTMODE_COUNT parameter.

Specify the C/T clock source for the operation. In event counting mode, an external C/T clock source is more useful than the internal C/T clock source; refer to page 128 for more information on specifying the C/T clock source.

Also specify the gate type that enables the operation; refer to page 131 for more information on specifying the gate type.

Start an event counting operation using the **olDaStart** function. To read the current number of events counted, use the **olDaReadEvents** function.

To stop the event counting operation, call **olDaStop**, **olDaAbort**, or **olDaReset**. The **olDaReset** function stops the operation and reinitializes the subsystem after stopping it.

**Figure 10** shows an example of an event counting operation. In this example the gate type is low level.
Use up/down counting mode to increment or decrement the number of rising edges that occur on the counter’s associated clock input, depending on the level of the counter’s associated gate signal. If the gate signal is high, the C/T increments; if the gate signal is low, the C/T decrements.

To determine if the subsystem supports up/down counting, use the `olDaGetSSCaps` function, specifying the capability `OLSSC_SUP_CTMODE_UP_DOWN`. If this function returns a nonzero value, the capability is supported.

To specify an up/down counting operation, use the `olDaSetCTMode` function, specifying the `OL_CTMODE_UP_DOWN` parameter.
Specify the C/T clock source for the operation as external. Note that you do not specify the gate type in software.

Start an up/down counting operation using the `olDaStart` function. To read the current number of rising edges counted, use the `olDaReadEvents` function.

To stop the event counting operation, call `olDaStop`, `olDaAbort`, or `olDaReset`; `olDaReset` function stops the operation and reinitializes the subsystem after stopping it.

Figure 11 shows an example of an up/down counting operation. The counter increments when the gate signal is high and decrements when the gate signal is low.

![Figure 11: Example of Up/Down Counting](image-url)
**Frequency Measurement**

You can also use event counting mode to measure the frequency of the clock input signal for the counter, since frequency is the number of events divided by a specified duration.

To determine if the subsystem supports event counting (and therefore, frequency measurement), use the `olDaGetSSCaps` function, specifying the capability `OLSSC_SUP_CTMODE_COUNT`. If this function returns a nonzero value, the capability is supported.

You can perform a frequency measurement operation in one of two ways: using the Windows timer to specify the duration or using a pulse of a known duration as the gate input signal to a counter/timer configured for event counting mode. The following subsections describe these ways of measuring frequency.

**Using the Windows Timer**

To perform a frequency measurement operation on a single C/T subsystem using the Windows timer to specify the duration, do the following:

1. Use the `olDaSetCTMode` function, specifying the `OL_CTMODE_COUNT` parameter.

2. Specify the input clock source using `olDaSetClockSource`. In frequency measurement mode, an external C/T clock source is more useful than the internal C/T clock source; refer to page 128 for more information on the external C/T clock source.

3. Use the `olDaSetGateType` function, specifying the `OL_GATE_NONE` parameter, to set the gate type to software.

4. Use the `olDaMeasureFrequency` function to specify the duration of the Windows timer (which has a resolution of 1 ms) and to start the frequency measurement operation.
Frequency is determined using the following equation:

\[
\text{Frequency} = \frac{\text{Number of Events}}{\text{Duration of the Windows Timer}}
\]

When the operation is complete, the OLDA_WM_MEASURE_DONE message is generated. Use the LongtoFreq (lParam) macro, described in the DataAcq SDK online help, to return the measured frequency value.

**Figure 12** shows an example of a frequency measurement operation. Three events are counted from the clock input signals during a duration of 300 ms. The frequency is 10 Hz (3/.3).

**Using a Pulse of a Known Duration**

If you need more accuracy than the Windows timer provides, you can connect a pulse of a known duration to the external gate input of a counter/timer configured for event counting; refer to the device user manual for wiring details.
The following example describes how to use the DataAcq SDK to measure frequency using two C/T subsystems: one that generates a variable-width one-shot pulse as the gate input to a second C/T subsystem configured for event counting mode:

1. Set up one C/T subsystem for one-shot mode as follows:
   a. Use the \texttt{olDaSetCTMode} function, specifying the \texttt{OL_CTMODE_ONESHOT} parameter.
   b. For this C/T subsystem, specify the clock source (with \texttt{olDaSetClockSource}), the clock frequency (with \texttt{olDaSetClockFrequency} if using an internal clock source, or \texttt{olDaSetExternalClockDivider} if using an external clock source), the gate type (with \texttt{olDaSetGateType}), the type of output pulse (with \texttt{olDaSetPulseType}), and the pulse width (with \texttt{olDaSetPulseWidth}). The pulse width and period are used to determine the time that the gate is active.
   c. Configure this C/T subsystem with \texttt{olDaConfig}.
   d. Get the actual clock frequency used by this C/T subsystem with \texttt{olDaGetClockFrequency} or \texttt{olDaGetExternalClockDivider}. You will use this value in the measurement period calculation.
   e. Get the actual pulse width used by this C/T subsystem with \texttt{olDaGetPulseWidth}. You will use this value in the measurement period calculation.

2. Set up another C/T subsystem for event counting mode:
   a. Use the \texttt{olDaSetCTMode} function, specifying the \texttt{OL_CTMODE_COUNT} parameter, to set up this C/T subsystem for event counting mode (and, therefore, a frequency measurement operation).
   b. For this C/T subsystem, use \texttt{olDaSetClockSource} to specify the clock source you want to measure. For frequency measurement operations, an external C/T clock source is more useful than the internal C/T clock source; refer to page 128 for more information on the external C/T clock source.
c. For this C/T subsystem, use the `olDaSetGateType` function to specify the gate type; ensure that the gate type for this C/T subsystem matches the active period of the output pulse from the C/T subsystem configured for one-shot mode.

d. Configure this C/T subsystem with `olDaConfig`.

3. Start the counter/timer configured for event counting mode with `olDaStart`.

4. Start the counter/timer configured for one-shot mode with `olDaStart`.

5. Allow a delay approximately equal to the measurement period to allow the one-shot to finish; events are counted only during the active period of the one-shot pulse.

6. For the event-counting C/T subsystem, read the number of events counted with `olDaReadEvents`.

7. Determine the measurement period using the following equation:

\[
\text{Measurement} = \frac{1}{\text{Active Pulse Width of Period}} \times \frac{1}{\text{Actual Clock Frequency}} \times \text{One-Shot C/T}
\]

8. Determine the frequency of the clock input signal using the following equation:

\[
\text{Frequency Measurement} = \frac{\text{Number of Events}}{\text{Measurement Period}}
\]
**Edge-to-Edge Measurement**

Use edge-to-edge measurement to measure the time interval between a specified start edge and a specified stop edge. The start edge and the stop edge can occur on the rising edge of the counter’s associated gate input, the falling edge of the counter’s associated gate input, the rising edge of the counter’s associated clock input, or the falling edge of the counter’s associated clock input. When the start edge is detected, the counter starts incrementing, and continues incrementing until the stop edge is detected.

To determine if the subsystem supports edge-to-edge measurement, use the `olDaGetSSCaps` function, specifying the capability `OLSSC_SUP_CTMODE_MEASURE`. This function returns a bit value indicating how edge-to-edge measurement mode is supported for the specified device. For example, if edge-to-edge measurements are supported on the gate signal only (using both rising and falling edges), a bit value of 3 is returned. Table 5 lists the possible bit values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>–</td>
<td>Edge-to-edge measurements are not supported.</td>
</tr>
<tr>
<td>0x01</td>
<td>SUP_GATE_RISING_BIT</td>
<td>Supports edge-to-edge measurements based on the rising edge of the gate signal.</td>
</tr>
<tr>
<td>0x02</td>
<td>SUP_GATE_FALLING_BIT</td>
<td>Supports edge-to-edge measurements based on the falling edge of the gate signal.</td>
</tr>
<tr>
<td>0x04</td>
<td>SUP_CLOCK_RISING_BIT</td>
<td>Supports edge-to-edge measurements based on the rising edge of the clock signal.</td>
</tr>
<tr>
<td>0x08</td>
<td>SUP_CLOCK_FALLING_BIT</td>
<td>Supports edge-to-edge measurements based on the falling edge of the clock signal.</td>
</tr>
</tbody>
</table>
To specify an edge-to-edge measurement operation, use the `olDaSetCTMode` function, specifying the OL_CTMODE_MEASURE parameter.

Specify the C/T clock source for the operation as internal. Specify the start edge with the `olDaSetMeasureStartEdge` function and the stop edge with `olDaSetMeasureStopEdge` function. Configure the counter/timer with `olDaConfig`.

Start an edge-to-edge measurement operation using the `olDaStart` function. To read the current counter value, use the `olDaReadEvents` function.

To stop the edge-to-edge measurement operation, call `olDaStop`, `olDaAbort`, or `olDaReset`; `olDaReset` function stops the operation and reinitializes the subsystem after stopping it.

Figure 13 shows an example of an edge-to-edge measurement operation. The start edge is a rising edge on the gate signal; the stop edge is a falling edge on the gate signal.

![Figure 13: Example of Edge-to-Edge Measurement](image-url)
You can use edge-to-edge measurement to measure the following:

- **Pulse width of a signal pulse** (the amount of time that a signal pulse is in a high or a low state, or the amount of time between a rising edge and a falling edge or between a falling edge and a rising edge). You can calculate the pulse width as follows:
  - \[ \text{Pulse width} = \frac{\text{Number of counts}}{\text{Internal C/T Clock Freq}} \]

- **Period of a signal pulse** (the time between two occurrences of the same edge - rising edge to rising edge or falling edge to falling edge). You can calculate the period as follows:
  - \[ \text{Period} = \frac{1}{\text{Frequency}} \]
  - \[ \text{Period} = \frac{\text{Number of counts}}{\text{Internal C/T Clock Freq}} \]

- **Frequency of a signal pulse** (the number of periods per second). You can calculate the frequency as follows:
  - \[ \text{Frequency} = \frac{\text{Internal C/T Clock Freq}}{\text{Number of Counts}} \]

### Continuous Edge-to-Edge Measurement

In continuous edge-to-edge measurement mode, the counter automatically performs an edge-to-edge measurement operation, where the counter starts incrementing when it detects the specified start edge and stops incrementing when it detects the specified stop edge. (Refer to the description of edge-to-edge measurement mode on page 115 for more information on start and stop edges.) When the operation completes, the counter remains idle until it is next read. On the next read, the current value of the counter (from the previous edge-to-edge measurement operation) is returned and the next edge-to-edge measurement operation is started automatically.

**Note:** If you read the counter before the measurement is complete, 0 is returned.
To determine if the subsystem supports continuous edge-to-edge measurement, use the `olDaGetSSCaps` function, specifying the capability `OLSSC_SUP_CTMODE_CONT_MEASURE`. This function returns a bit value indicating how edge-to-edge measurement mode is supported for the specified device. For example, if edge-to-edge measurements are supported on the gate signal only (using both rising and falling edges), a bit value of 3 is returned. Table 6 lists the possible bit values.

### Table 6: Values for `OLSSC_SUP_CTMODE_CONT_MEASURE`

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>–</td>
<td>Continuous edge-to-edge measurements are not supported.</td>
</tr>
<tr>
<td>0x01</td>
<td>SUP_GATE_RISING_BIT</td>
<td>Supports continuous edge-to-edge measurements based on the rising edge of the gate signal.</td>
</tr>
<tr>
<td>0x02</td>
<td>SUP_GATE_FALLING_BIT</td>
<td>Supports continuous edge-to-edge measurements based on the falling edge of the gate signal.</td>
</tr>
<tr>
<td>0x04</td>
<td>SUP_CLOCK_RISING_BIT</td>
<td>Supports continuous edge-to-edge measurements based on the rising edge of the clock signal.</td>
</tr>
<tr>
<td>0x08</td>
<td>SUP_CLOCK_FALLING_BIT</td>
<td>Supports continuous edge-to-edge measurements based on the falling edge of the clock signal.</td>
</tr>
</tbody>
</table>

To specify a continuous edge-to-edge measurement operation, use the `olDaSetCTMode` function, specifying the `OL_CTMODE_CONT_MEASURE` parameter.

Specify the C/T clock source for the operation as internal. Specify the start edge with the `olDaSetMeasureStartEdge` function and the stop edge with `olDaSetMeasureStopEdge` function.
When you configure the counter/timer with `olDaConfig`, the continuous edge-to-edge measurement operation starts immediately.

To read the current counter value, use the `olDaReadEvents` function or read the counter/timer channel as part of the analog input channel-gain list, if the device supports this capability. For example, you might see results similar to the following if you read the counter/timer channel as part of the analog input channel list:

<table>
<thead>
<tr>
<th>Time</th>
<th>A/D Value</th>
<th>Counter/Timer Value</th>
<th>Status of Continuous Edge-to-Edge Measurement Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5002</td>
<td>0</td>
<td>Operation started when the C/T subsystem was configured, but is not complete</td>
</tr>
<tr>
<td>20</td>
<td>5004</td>
<td>0</td>
<td>Operation not complete</td>
</tr>
<tr>
<td>30</td>
<td>5003</td>
<td>0</td>
<td>Operation not complete</td>
</tr>
<tr>
<td>40</td>
<td>5002</td>
<td>12373</td>
<td>Operation complete</td>
</tr>
<tr>
<td>50</td>
<td>5000</td>
<td>0</td>
<td>Next operation started, but is not complete</td>
</tr>
<tr>
<td>60</td>
<td>5002</td>
<td>0</td>
<td>Operation not complete</td>
</tr>
<tr>
<td>70</td>
<td>5004</td>
<td>0</td>
<td>Operation not complete</td>
</tr>
<tr>
<td>80</td>
<td>5003</td>
<td>12403</td>
<td>Operation complete</td>
</tr>
<tr>
<td>90</td>
<td>5002</td>
<td>0</td>
<td>Next operation started, but is not complete</td>
</tr>
</tbody>
</table>
You can use a continuous edge-to-edge measurement to measure the following:

- **Pulse width of a signal pulse** (the amount of time that a signal pulse is in a high or a low state, or the amount of time between a rising edge and a falling edge or between a falling edge and a rising edge). You can calculate the pulse width as follows:
  - Pulse width = Number of counts/Internal C/T Clock Freq

- **Period of a signal pulse** (the time between two occurrences of the same edge - rising edge to rising edge or falling edge to falling edge). You can calculate the period as follows:
  - Period = 1/Frequency
  - Period = Number of counts/Internal C/T Clock Freq

- **Frequency of a signal pulse** (the number of periods per second). You can calculate the frequency as follows:
  - Frequency = Internal C/T Clock Freq/Number of Counts

To stop a continuous edge-to-edge measurement operation, call **olDaStop**, **olDaAbort**, or **olDaReset**; **olDaReset** function stops the operation and reinitializes the subsystem after stopping it.

**Rate Generation**

Use rate generation mode to generate a continuous pulse output signal from the counter; this mode is sometimes referred to as continuous pulse output or pulse train output. You can use this pulse output signal as an external clock to pace analog input, analog output, or other counter/timer operations.

To determine if the subsystem supports rate generation, use the **olDaGetSSCaps** function, specifying the capability **OLSSC_SUP_CTMODE_RATE**. If this function returns a nonzero value, the capability is supported.
To specify a rate generation mode, use the `olDaSetCTMode` function, specifying the OL_CTMODE_RATE parameter.

Specify the C/T clock source for the operation. In rate generation mode, either the internal or external C/T clock input source is appropriate depending on your application; refer to page 128 for information on specifying the C/T clock source.

Specify the frequency of the C/T clock output signal. For an internal C/T, the `olDaSetClockFrequency` function determines the frequency of the output pulse. For an external C/T clock source, the frequency of the clock input source divided by the clock divider (specified with the `olDaSetExternalClockDivider` function) determines the frequency of the output pulse.

Specify the polarity of the output pulses (high-to-low transitions or low-to-high transitions) and the duty cycle of the output pulses; refer to page 136 for more information.

Also specify the gate type that enables the operation; refer to page 131 for more information on specifying the gate type.

Start rate generation mode using the `olDaStart` function. While rate generation mode is enabled, the counter outputs a pulse of the specified type and frequency continuously. As soon as the operation is disabled, the pulse output operation stops.

To stop rate generation if it is in progress, call `olDaStop`, `olDaAbort`, or `olDaReset`; `olDaReset` stops the operation and reinitializes the subsystem after stopping it.

*Figure 14* shows an example of an enabled rate generation operation using an external C/T clock source with an input frequency of 4 kHz, a clock divider of 4, a low-to-high pulse type, and a duty cycle of 50%. (The gate type does not matter for this example.) A 1 kHz square wave is the generated output.
Figure 14: Example of Rate Generation Mode with a 50% Duty Cycle

Figure 15 shows the same example using a duty cycle of 75%.

Figure 15: Example of Rate Generation Mode with a 75% Duty Cycle
Figure 16 shows the same example using a duty cycle of 25%.

**One-Shot**

Use one-shot mode to generate a single pulse output signal from the counter when the operation is triggered (determined by the gate input signal). You can use this pulse output signal as an external digital (TTL) trigger to start analog input, analog output, or other operations.

To determine if the subsystem supports one-shot mode, use the `olDaGetSSCaps` function, specifying the capability OLSSC_SUP_CTMODE_ONESHOT. If this function returns a nonzero value, the capability is supported.

To specify a one-shot operation, use the `olDaSetCTMode` function, specifying the OL_CTMODE_ONESHOT parameter.
Specify the C/T clock source for the operation. In one-shot mode, the internal C/T clock source is more useful than an external C/T clock source; refer to page 128 for more information on specifying the C/T clock source.

Specify the polarity of the output pulse (high-to-low transition or low-to-high transition) and the duty cycle of the output pulse; refer to page 136 for more information.

**Note:** In the case of a one-shot operation, set the duty cycle to 100% to output a pulse immediately. Using a duty cycle less than 100% acts as a pulse output delay.

Also specify the gate type that triggers the operation; refer to page 131 for more information.

Configure the operation with the `olDaConfig` function.

To start a one-shot pulse output operation, use the `olDaStart` function. When the one-shot operation is triggered (determined by the gate input signal), a single pulse is output; then, the one-shot operation stops. All subsequent clock input signals and gate input signals are ignored.

Use software to specify the counter/timer mode as one-shot and wire the signals appropriately.

**Figure 17** shows an example of a one-shot operation using an external gate input (rising edge), a clock output frequency of 1 kHz (one pulse every 1 ms), a low-to-high pulse type, and a duty cycle of 100%. **Figure 18** shows the same example using a duty cycle of less than or equal to 1%.
Figure 17: Example of One-Shot Mode Using a Duty Cycle of 100%

Figure 18: Example of One-Shot Mode Using a Duty Cycle Less Than or Equal to 1%
Repetitive One-Shot

Use repetitive one-shot mode to generate a pulse output signal each time the device detects a trigger (determined by the gate input signal). You can use this mode to clean up a poor clock input signal by changing its pulse width, then outputting it.

To determine if the subsystem supports repetitive one-shot mode, use the `olDaGetSSCaps` function, specifying the capability `OLSSC_SUP_CTMODE_ONESHOT_RPT`. If this function returns a nonzero value, the capability is supported.

To specify a repetitive one-shot operation, use the `olDaSetCTMode` function, specifying the `OL_CTMODE_ONESHOT_RPT` parameter.

Specify the C/T clock source for the operation. In repetitive one-shot mode, the internal C/T clock source is more useful than an external C/T clock source; refer to page 128 for more information on specifying the C/T clock source.

Specify the polarity of the output pulses (high-to-low transitions or low-to-high transitions) and the duty cycle of the output pulses; refer to page 136 for more information. Also specify the gate type that triggers the operation; refer to page 131 for more information.

Configure the operation with the `olDaConfig` function.

To start a repetitive one-shot pulse output operation, use the `olDaStart` function. When the one-shot operation is triggered (determined by the gate input signal), a pulse is output. When the device detects the next trigger, another pulse is output.

This operation continues until you stop the operation using `olDaStop`, `olDaAbort`, or `olDaReset`; `olDaReset` stops the operation and reinitializes the subsystem after stopping it.
Note: Triggers that occur while the pulse is being output are not detected by the device.

Figure 19 shows an example of a repetitive one-shot operation using an external gate (rising edge); a clock output frequency of 1 kHz (one pulse every 1 ms), a low-to-high pulse type, and a duty cycle of 100%.

**Figure 19: Example of Repetitive One-Shot Mode Using a Duty Cycle of 100%**

Figure 20 shows the same example using a duty cycle of 50%.
Figure 20: Example of Repetitive One-Shot Mode Using a Duty Cycle of 50%

C/T Clock Sources

The DataAcq SDK defines the following clock sources for counter/timers:

- Internal C/T clock
- External C/T clock
- Internally cascaded clock
- Extra C/T clocks

The following subsections describe these clock sources.
**Internal C/T Clock**

The internal C/T clock is the clock source on the device that paces a counter/timer operation for a C/T subsystem.

To determine if the subsystem supports an internal C/T clock, use the `olDaGetSSCaps` function, specifying the `OLSSC_SUP_INTCLOCK` capability. If this function returns a nonzero value, the capability is supported.

To specify the clock source, use the `olDaSetClockSource` function.

Using the `olDaSetClockFrequency` function, specify the frequency of the clock output signal.

To determine the maximum frequency that the subsystem supports, use the `olDaGetSSCapsEx` function, specifying the `OLSSCE_MAXTHROUGHPUT` capability. To determine the minimum frequency that the subsystem supports, use the `olDaGetSSCapsEx` function, specifying the `OLSSCE_MINTHROUGHPUT` capability.

**External C/T Clock**

The external C/T clock is a clock source attached to the device that paces counter/timer operations for a C/T subsystem. The external C/T clock is useful when you want to pace at rates not available with the internal clock or if you want to pace at uneven intervals.

To determine if the subsystem supports an external C/T clock, use the `olDaGetSSCaps` function, specifying the `OLSSC_SUP_EXTCLOCK` capability. If this function returns a nonzero value, the capability is supported.
Specify the clock source using the `olDaSetClockSource` function. Specify the clock divider using the `olDaSetExternalClockDivider` function; the clock input signal divided by the clock divider determines the frequency of the clock output signal.

To determine the maximum clock divider that the subsystem supports, use the `olDaGetSSCapsEx` function, specifying the OLSSCE_MAXCLOCKDIVIDER capability. To determine the minimum clock divider that the subsystem supports, use the `olDaGetSSCapsEx` function, specifying the OLSSCE_MINCLOCKDIVIDER capability.

**Internally Cascaded Clock**

You can also internally connect or cascade the clock output signal from one counter/timer to the clock input signal of the next counter/timer in software. In this way, you can create a 32-bit counter out of two 16-bit counters.

To determine if the subsystem supports internal cascading, use the `olDaGetSSCaps` function, specifying the OLSSC_SUP_CASCADING capability. If this function returns a nonzero value, the capability is supported.

Specify whether the subsystem is internally cascaded or not (single) using the `olDaSetCascadeMode` function.

---

**Note:** If a counter/timer is cascaded, you specify the clock input and gate input for the first counter in the cascaded pair. For example, if counters 1 and 2 are cascaded, specify the clock input and gate input for counter 1.
Extra C/T Clock Source

Extra C/T clock sources may be defined by your device driver.

To determine how many extra clock sources are supported by your subsystem, use the `olDaGetSSCaps` function, specifying the `OLSSC_NUMEXTRACLOCKS` capability. Refer to your device/driver documentation for a description of these clocks.

To specify internal or external extra clock sources and their frequencies and/or clock dividers, refer to the previous subsections.

Gate Types

The active edge or level of the gate input to the counter enables or triggers counter/timer operations. The DataAcq SDK defines the following gate input types:

- Software
- High level
- Low level
- High edge
- Low edge
- Any level
- High level debounced
- Low level debounced
- High edge debounced
- Low edge debounced
- Any level debounced

To specify the gate type, use the `olDaSetGateType` function. The following subsections describe these gate types.
Software Gate Type

A software gate type enables any specified counter/timer operation immediately when the `olDaSetGateType` function is executed.

To determine if the subsystem supports a software gate, use the `olDaGetSSCaps` function, specifying the OLSSC_SUP_GATE_NONE capability. If this function returns a nonzero value, the capability is supported.

High-Level Gate Type

A high-level external gate type enables a counter/timer operation when the external gate signal is high, and disables a counter/timer operation when the external gate signal is low. Note that this gate type is used only for event counting, frequency measurement, and rate generation; refer to page 107 for more information on these modes.

To determine if the subsystem supports a high-level external gate input, use the `olDaGetSSCaps` function, specifying the OLSSC_SUP_GATE_HIGH_LEVEL capability. If this function returns a nonzero value, the capability is supported.

Low-Level Gate Type

A low-level external gate type enables a counter/timer operation when the external gate signal is low, and disables the counter/timer operation when the external gate signal is high. Note that this gate type is used only for event counting, frequency measurement, and rate generation; refer to page 107 for more information on these modes.

To determine if the subsystem supports a low-level external gate input, use the `olDaGetSSCaps` function, specifying the OLSSC_SUP_GATE_LOW_LEVEL capability. If this function returns a nonzero value, the capability is supported.
**Low-Edge Gate Type**

A low-edge external gate type triggers a counter/timer operation on the transition from the high edge to the low edge (falling edge). Note that this gate type is used only for one-shot and repetitive one-shot mode; refer to page 126 for more information on these modes.

To determine if the subsystem supports a low-edge external gate input, use the `olDaGetSSCaps` function, specifying the `OLSSC_SUP_GATE_LOW_EDGE` capability. If this function returns a nonzero value, the capability is supported.

**High-Edge Gate Type**

A high-edge external gate type triggers a counter/timer operation on the transition from the low edge to the high edge (rising edge). Note that this gate type is used only for one-shot and repetitive one-shot mode; refer to page 107 for more information on these modes.

To determine if the subsystem supports a high-edge external gate input, use the `olDaGetSSCaps` function, specifying the `OLSSC_SUP_GATE_HIGH_EDGE` capability. If this function returns a nonzero value, the capability is supported.

**Any Level Gate Type**

A level gate type enables a counter/timer operation on the transition from any level. Note that this gate type is used only for event counting, frequency measurement, and rate generation; refer to page 107 for more information on these modes.

To determine if the subsystem supports a level external gate input, use the `olDaGetSSCaps` function, specifying the `OLSSC_SUP_GATE_LEVEL` capability. If this function returns a nonzero value, the capability is supported.
**High-Level, Debounced Gate Type**

A high-level, debounced gate type enables a counter/timer operation when the external gate signal is high and debounced. Note that this gate type is used only for event counting, frequency measurement, and rate generation; refer to page 107 for more information on these modes.

To determine if the subsystem supports a high-level debounced external gate input, use the `olDaGetSSCaps` function, specifying the OLSSC_SUP_GATE_HIGH_LEVEL_DEBOUNCE capability. If this function returns a nonzero value, the capability is supported.

**Low-Level, Debounced Gate Type**

A low-level, debounced gate type enables a counter/timer operation when the external gate signal is low and debounced. Note that this gate type is used only for event counting, frequency measurement, and rate generation; refer to page 107 for more information on these modes.

To determine if the subsystem supports a low-level debounced external gate input, use the `olDaGetSSCaps` function, specifying the OLSSC_SUP_GATE_LOW_LEVEL_DEBOUNCE capability. If this function returns a nonzero value, the capability is supported.

**High-Edge, Debounced Gate Type**

A high-edge, debounced gate type triggers a counter/timer operation on the rising edge of the external gate signal; the signal is debounced. Note that this gate type is used only for one-shot and repetitive one-shot mode; refer to page 107 for more information on these modes.

To determine if the subsystem supports a high-edge debounced external gate input, use the `olDaGetSSCaps` function, specifying the
OLSSC_SUP_GATE_HIGH_EDGE_DEBOUNCE capability. If this function returns a nonzero value, the capability is supported.

**Low-Edge, Debounced Gate Type**

A low-edge, debounced gate type triggers a counter/timer operation on the falling edge of the external gate signal; the signal is debounced. Note that this gate type is used only for one-shot and repetitive one-shot mode; refer to page 107 for more information on these modes.

To determine if the subsystem supports a low-edge debounced external gate input, use the `olDaGetSSCaps` function, specifying the OLSSC_SUP_GATE_LOW_EDGE_DEBOUNCE capability. If this function returns a nonzero value, the capability is supported.

**Level, Debounced Gate Type**

A level, debounced gate type enables a counter/timer operation on the transition of any level of the external gate signal; the signal is debounced. Note that this gate type is used only for event counting, frequency measurement, and rate generation; refer to page 107 for more information on these modes.

To determine if the subsystem supports a high-edge debounced external gate input, use the `olDaGetSSCaps` function, specifying the OLSSC_SUP_GATE_LEVEL_DEBOUNCE capability. If this function returns a nonzero value, the capability is supported.
Pulse Output Types and Duty Cycles

The DataAcq SDK defines the following pulse output types:

- **High-to-low transitions** – The low portion of the total pulse output period is the active portion of the counter/timer clock output signal.

  To determine if the subsystem supports high-to-low transitions on the pulse output signal, use the `olDaGetSSCaps` function, specifying the OLSSC_SUP_PLS_HIGH2LOW capability. If this function returns a nonzero value, the capability is supported.

- **Low-to-high transitions** – The high portion of the total pulse output period is the active portion of the counter/timer pulse output signal.

  To determine if the subsystem supports low-to-high transitions on the pulse output signal, use the `olDaGetSSCaps` function, specifying the OLSSC_SUP_PLS_LOW2HIGH capability. If this function returns a nonzero value, the capability is supported.

Specify the pulse output type using the `olDaSetPulseType` function.

The duty cycle (or pulse width) indicates the percentage of the total pulse output period that is active. A duty cycle of 50, then, indicates that half of the total pulse is low and half of the total pulse output is high.

You can determine whether the pulse width is programmable by querying the OLSSC_FIXED_PULSE_WIDTH capability. In a non-zero result is returned, the pulse width is fixed and cannot be programmed. If 0 is returned, the device supports a programmable pulse width.

Specify the pulse width using the `olDaSetPulseWidth` function.
Figure 21 illustrates a low-to-high pulse with a duty cycle of approximately 30%.

Figure 21: Example of a Low-to-High Pulse Output Type
Quadrature Decoder Operations

On some devices, quadrature decoders may be supported by the counter/timer subsystem.

Quadrature decoders let you accept inputs (A, B, and Index) from a quadrature encoder device and determine relative or absolute position of the inputs and/or rotational speed.

Using the `olDaSetQuadDecoder` function, you can specify the following parameters for a quadrature decoder operation:

- The pre-scale value that is used to filter the onboard clock. Using a pre-scale value can remove ringing edges and unwanted noise for more accurate results.
- The mode of operation (X1 or X4 mode) that matches the quadrature encoder mode.
- The index mode, which either enables the Index signal or disables the Index signal. If enabled, the value of the decoder is reset to 0 whenever a selected edge (high or low) of the Index signal goes high or low. If disabled, the Index signal has no effect.

You can read the value of the quadrature decoder using the `olDaReadEvents` function to determine relative or absolute position.

To determine the rotation of a quadrature encoder, use the following formula:

$$\text{Rotation degrees} = \frac{\text{Count} \times 360\text{ degrees}}{4 \times N}$$

where $N$ is the number of pulses generated by the quadrature encoder per rotation. For example, if every rotation of the quadrature encoder generated 10 pulses, and the value read from the quadrature decoder is 20, the rotation of the quadrature encoder is 180 degrees ($20/40 \times 360\text{ degrees}$).
Simultaneous Startup

If supported, you can synchronize subsystems to perform simultaneous startup. Note that you cannot perform simultaneous startup on subsystems configured for single-value operations unless you are using a simultaneous sampling module.

To determine if the subsystems support simultaneous startup, use the `olDaGetSSCaps` function for each subsystem, specifying the OLSSC_SUP_SIMULTANEOUS_START capability. If this function returns a nonzero value, the capability is supported.

You can synchronize the triggers of subsystems by specifying the same trigger source for each of the subsystems that you want to start simultaneously and wiring them to the device, if appropriate.

Use the `olDaGetSSLList` function to allocate a simultaneous start list. Then, use the `olDaPutDassToSSLList` function to put the subsystems that you want to start simultaneously on the start list.

To determine the device handles given to each subsystem on the simultaneous start list, use the `olDaEnumSSLList` function.

Pre-start the subsystems using the `olDaSimultaneousPreStart` function. Pre-starting a subsystem ensures a minimal delay once the subsystems are started. Once you call the `olDaSimultaneousPreStart` function, do not alter the settings of the subsystems on the simultaneous start list.

Start the subsystems using the `olDaSimultaneousStart` function. When started, both subsystems are triggered simultaneously.
Note: Do not call olDaStart when using simultaneous start lists, since the subsystems are already started.

When you are finished with the operations, call the olDaReleaseSSList function to free the simultaneous start list. Then, call the olDaReleaseDASS function for each subsystem to free it before calling olDaTerminate.

To stop the simultaneous operations, call olDaStop (for an orderly stop), olDaAbort (for an abrupt stop) or olDaReset (for an abrupt stop that reinitializes the subsystem).
Programming Flowcharts

Single-Value Input Operations ........................................ 143
Single-Value Output Operations ..................................... 146
Continuous Buffered Input Operations ............................ 148
Continuous Buffered Output Operations ......................... 150
Event Counting Operations .......................................... 152
Up/Down Counting Operations ...................................... 154
Frequency Measurement Operations ............................... 156
Edge-to-Edge Measurement Operations ............................ 158
Continuous Edge-to-Edge Measurement Operations ............ 160
Pulse Output Operations ............................................. 162
Quadrature Decoder Operations ...................................... 164
Simultaneous Operations ............................................. 166
If you are unfamiliar with the capabilities of your device and/or subsystem, query the device as follows:

- To determine the number and types of DT-Open Layers devices and drivers installed, use the `olDaEnumBoards` function.
- To determine the subsystems supported by the device, use the `olDaEnumSubSystems` or `olDaGetDevCaps` function.
- To determine the capabilities of a subsystem, use the `olDaGetSSCaps` or `olDaGetSSCapsEx` function, specifying one of the capabilities listed in Table 2 on page 25.
- To determine the gains, filters, ranges, and resolutions if more than one is available, use the `olDaEnumSSCaps` function.

Then, follow the flowcharts presented in the remainder of this chapter to perform the desired operation.

---

**Notes:** Depending on your device, some of the settings may not be programmable. Refer to your device driver documentation for details.

Although the flowcharts do not show error checking, it is recommended that you check for errors after each function call.

Some steps represent several substeps; if you are unfamiliar with the detailed operations involved with any one step, refer to the indicated page for detailed information. Optional steps appear in shaded boxes.
**Single-Value Input Operations**

1. **Initialize the device driver and get the device handle with `olDaInitialize`.**
2. **Get a handle to the subsystem with `olDaGetDASS`.**
3. **Set the data flow to `OL_DF_SINGLEVALUE` using `olDaSetDataFlow`.**
4. **Set the subsystem parameters (see page 168).**
5. **Set up channel parameters (see page 169).**
6. **Configure the subsystem using `olDaConfig`.**
7. **Go to the next page.**

Specify A/D for an analog input subsystem or DIN for a digital input subsystem.
Continued from previous page.

Does subsystem support simultaneous mode?  
Yes  
Does subsystem support floating-point values?  
Yes  
Read CJC values?  
Yes  
Acquire a single CJC value for each input channel using oIDaGetCjcTemperatures.
No  
Acquire a single floating-point value from each input channel using oIDaGetSingleFloats.

No  
Does subsystem support floating-point values?  
Yes  
Read CJC value?  
Yes  
Acquire a single CJC value for the input channel using oIDaGetCjcTemperature.
No  
Acquire a single floating-point value from the input channel using oIDaGetSingleFloat.

No  
Acquire a single integer value from the input channel using oIDaGetSingleValue or oIDaGetSingleValueEx.

Go to the next page.
Acquire another value(s)?

- Yes: Repeat the procedure on previous page.
- No: Release the subsystem using `olDaReleaseDASS`.

Release the driver and terminate the session using `olDaTerminate`.
Single-Value Output Operations

1. Initialize the device driver and get the device handle with `olDaInitialize`.

2. Get a handle to the subsystem with `olDaGetDASS`.

3. Set the data flow to `OL_DF_SINGLEVALUE` using `olDaSetDataFlow`.

4. Set the subsystem parameters (see page 168).

5. Set up channel parameters (see page 169).

6. Configure the subsystem using `olDaConfig`.

Specify D/A for an analog output subsystem or DOUT for a digital output subsystem.

Go to the next page.
Does subsystem support simultaneous mode?

Yes

Output a single value from each output channel using `olDaPutSingleValues`.

No

Output a single value from the specified output channel using `olDaPutSingleValue`.

Yes

No another value(s)?

Yes

Release the subsystem using `olDaReleaseDASS`.

No

Release the driver and terminate the session using `olDaTerminate`.

Continued from previous page.
Continuous Buffered Input Operations

Initialize the device driver and get the device handle with `olDaInitialize`.

Get a handle to the A/D subsystem with `olDaGetDASS`.

Set the data flow using `olDaSetDataFlow`.

Set the DMA channel usage using `olDaSetDmaUsage`.

Set the subsystem parameters (see page 168).

Set up the channel parameters (see page 169).

Set up the channel list (see page 170).

Go to the next page.

Specify OL_DF_CONTINUOUS (the default value) for post-trigger operations, OL_DF_CONTINUOUS_PRETRIG for continuous pre-trig operations, or OL_DF_CONTINUOUS_ABOUTTRIG for continuous about-trigger operations).
Set up the clocks, triggers, and pre-triggers (see page 171).

If you want to use triggered scan mode, set up the scan (see page 173.)

Set up buffering (see page 174).

Configure the A/D subsystem using `olDaConfig`.

Start the operation with `olDaStart`.

Deal with messages and buffers (see page 176).

Stop the operation (see page 182).

Clean up the subsystem (see page 183).

Continued from previous page.

After configuration, if using an internal clock, you can use `olDaGetClockFrequency` to get the actual frequency that the internal sample clock can achieve; if using an external clock, you can use `olDaGetExternalClockDivider` to get the actual clock divider that the device can achieve; if using internal retrigger mode, you can use `olDaGetRetriggerFrequency` to get the actual frequency that the internal retrigger clock can achieve.
Continuous Buffered Output Operations

1. Initialize the device driver and get the device handle with `olDaInitialize`.

2. Get a handle to the D/A subsystem with `olDaGetDASS`.

3. Set the data flow to `OL_DF_CONTINUOUS` using `olDaSetDataFlow`.

4. Set the DMA channel usage using `olDaSetDmaUsage`.

5. Set the subsystem parameters (see page 168).

6. Set up the channel parameters (see page 169).

7. Set up the channel list (see page 170).

8. Go to the next page.
Set up the clocks and triggers (see page 171).

Set up buffering (see page 175).

Configure the D/A subsystem using `olDaConfig`.

Start the operation with `olDaStart`.

Deal with messages and buffers (see page 179).

Stop the operation (see page 182).

Clean up the subsystem (see page 183).

After configuration, if using an internal clock, you can use `olDaGetClockFrequency` to get the actual frequency that the internal output clock can achieve; or if using an external clock, you can use `olDaGetExternalClockDivider` to get the actual clock divider that the device can achieve.
Event Counting Operations

1. Initialize the device driver and get the device handle with `olDaInitialize`.
2. Get a handle to the C/T subsystem with `olDaGetDASS`.
3. Set the cascade mode using `olDaSetCascadeMode`.
4. Set up the clocks and gates (see page 181).
5. Specify the mode as OL_CTMODE_COUNT using `olDaSetCTMode`.
6. Configure the subsystem using `olDaConfig`.
7. Go to the next page.
Start the operation using `olDaStart`.

Read the events counted using `olDaReadEvents`.

Get update of events total?

Yes

Stop the operation (see page 182).

No

Clean up the subsystem (see page 184).
Up/Down Counting Operations

1. Initialize the device driver and get the device handle with `olDaInitialize`.
2. Get a handle to the C/T subsystem with `olDaGetDASS`.
3. Specify the clock source as `OL_CLK_EXTERNAL` using `olDaSetClockSource`.
4. Specify an external clock divider with `olDaSetExternalClockDivider`.
5. Specify the mode as `OL_CTMODE_UP_DOWN` using `olDaSetCTMode`.
6. Configure the subsystem using `olDaConfig`.
7. Go to the next page.
Start the operation using `olDaStart`.

Read the events counted using `olDaReadEvents`.

**Continued from previous page.**

Get update of events total?

- Yes
- No

Stop the operation (see page 182).

Clean up the subsystem (see page 184).
Frequency Measurement Operations

Note: If you need more accuracy than the system timer provides, refer to page 111.

Initialize the device driver and get the device handle with `olDaInitialize`.

Get a handle to the C/T subsystem with `olDaGetDASS`.

Set the cascade mode using `olDaSetCascadeMode`.

Set up the clocks (see page 171).

Specify the mode as OL_CTMODE_COUNT using `olDaSetCTMode`.

Go to the next page.
Configure the subsystem using \texttt{olDaConfig}.

Start the frequency measurement operation using \texttt{olDaMeasureFrequency}.

Get measure done message?

Yes

Message is in the form \texttt{OLDA_WM_MEASURE_DONE}.

Use the \texttt{LongtoFreq (lParam)} macro to get the measured frequency value:

\begin{verbatim}
float = Freq;
Freq = LongtoFreq (lParam);
\end{verbatim}

No

Clean up the subsystem (see page 184).
Edge-to-Edge Measurement Operations

Initialize the device driver and get the device handle with `olDaInitialize`.

Get a handle to the C/T subsystem with `olDaGetDASS`.

Specify the mode as `OL_CTMODE_MEASURE` using `olDaSetCTMode`.

Specify the clock source as `OL_CLK_INTERNAL` using `olDaSetClockSource`.

Specify the start edge using `olDaSetMeasureStartEdge`.

Specify the stop edge using `olDaSetMeasureStopEdge`.

Configure the subsystem using `olDaConfig`.

Go to the next page.

Specify `OL_GATE_RISING` for a rising edge on the counter’s associated gate input, `OL_GATE_FALLING` for a falling edge on the counter’s associated gate input, `OL_CLOCK_RISING` for a rising edge on the counter’s associated clock input, or `OL_CLOCK_FALLING` for a falling edge on the counter’s associated clock input.
Continued from previous page.

Start the operation using **olDaStart**.

- Message is in the form OLDA_WM_EVENT_DONE. Note that if you want to perform another edge-to-edge measurement, you can call **olDaStart** again or use the OLDA_WM_EVENT_DONE handler to call **olDaStart** again.
- Yes: Read the value of the lParam parameter in the OLDA_WM_EVENT_DONE message to determine the value of the counter.
- No: Clean up the subsystem (see page 184).
Continuous Edge-to-Edge Measurement Operations

1. Initialize the device driver and get the device handle with `olDaInitialize`.
2. Get a handle to the C/T subsystem with `olDaGetDASS`.
3. Specify the mode as `OL_CTMODE_CONT_MEASURE` using `olDaSetCTMode`.
4. Specify the clock source as `OL_CLK_INTERNAL` using `olDaSetClockSource`.
5. Specify the start edge using `olDaSetMeasureStartEdge`.
6. Specify the stop edge using `olDaSetMeasureStopEdge`.
7. Configure the subsystem using `olDaConfig`.

Specify `OL_GATE_RISING` for a rising edge on the counter’s associated gate input, `OL_GATE_FALLING` for a falling edge on the counter’s associated gate input, `OL_CLOCK_RISING` for a rising edge on the counter’s associated clock input, or `OL_CLOCK_FALLING` for a falling edge on the counter’s associated clock input.

Go to the next page.
Continued from previous page.

Start the operation using `olDaStart`.

Read the value of the counter/timer using `olDaReadEvents` or through the channel-gain list.

Yes

Read value of counter again?

No

Stop the operation (see page 182).

Clean up the subsystem (see page 184).

On each read of the counter/timer, the current value of the counter/timer channel is returned and the next edge-to-edge measurement mode operation starts. If the current edge-to-edge measurement operation is still in progress, 0 is returned.
Pulse Output Operations

Initialize the device driver and get the device handle with `olDaInitialize`.

Get a handle to the C/T subsystem with `olDaGetDASS`.

Set the cascade mode using `olDaSetCascadeMode`.

Set up the clocks and gates (see page 181).

Specify the mode using `olDaSetCTMode`.

Go to the next page.

Specify `OL_CTMODE_RATE` for rate generation (continuous pulse output), `OL_CTMODE_ONESHOT` for single one-shot, or `OL_CTMODE_ONESHOT_RPT` for repetitive one-shot.
Stop the operation (see page 182).

Specify the output pulse type using `olDaSetPulseType`.

Specify the duty cycle of the output pulse using `olDaSetPulseWidth`.

Configure the subsystem using `olDaConfig`.

Start the operation using `olDaStart`.

Stop the operation (see page 182).

Clean up the subsystem (see page 184).

This step is not needed for single one-shot operations.
Quadrature Decoder Operations

1. Initialize the device driver and get the device handle with `olDaInitialize`.
2. Get a handle to the C/T subsystem with `olDaGetDASS`.
3. Set the clock source to OL_CLK_EXTERNAL using `olDaSetClockSource`.
4. Set up the quadrature decoder operation using `olDaSetQuadDecoder`.
5. Configure the subsystem using `olDaConfig`.
6. Start the operation using `olDaStart`.
7. Go to the next page.
Continued from previous page.

Read the events counted using `olDaReadEvents`.

Get update of events total?

Yes →

No →

Stop the operation (see page 182).

Clean up the subsystem (see page 184).
Simultaneous Operations

Configure the subsystem that you want to run simultaneously.

Allocate a simultaneous start list using `olDaGetSSLList`.

Put each subsystem to be simultaneously started on the start list using `olDaPutDassToSSLList`.

Prestart the subsystems on the simultaneous start list with `olDaSimultaneousPreStart`.

Start the subsystems on the simultaneous start list with `olDaSimultaneousStart`.

Go to the next page.

See the previous flow diagrams in this chapter; you cannot perform single-value operations simultaneously on multiplexed A/D modules.
Deal with messages (see page 176 for analog input operations; see page 179 for analog output operations).

Stop the operation (see page 182).

Clean up the subsystem (see page 183 for analog I/O operations).
Set Subsystem Parameters

- **olDaSetChannelType**: Specify the channel type (single-ended or differential). Specify single-ended if you are using pseudo-differential channels.

- **olDaSetResolution**: Specify the resolution.

- **olDaSetEncoding**: For A/D and D/A subsystems, specify the data encoding type.

- **olDaSetRange**: For A/D and D/A subsystems that support a single range for the entire subsystem, specify the voltage range for the entire subsystem.

- **olDaSetReturnCjcTemperature InStream**: For A/D subsystems that support thermocouple inputs, enable or disable the ability to return CJC data in the data stream. By default, this ability is disabled.
Set up Channel Parameters

- `olDaSetChannelRange`
  For A/D and D/A subsystems that support a range per channel, specify the voltage range for the selected A/D or D/A channel.

- `olDaSetThermocoupleType`
  If supported, specify the thermocouple type for each A/D channel.

- `olDaSetChannelFilter`
  If supported, specify the filter for each A/D or D/A channel.

- `olDaSetCouplingType`
  If supported, specify the coupling type for the specified A/D channel.

- `olDaSetExcitationCurrentSource`
  If supported, specify the excitation current source for the specified A/D channel.

- `olDaSetChannelRange` (decision point)
  Using an internal excitation current source?
  
  - Yes
    `olDaSetExcitationCurrentValue`
    Specify the value of the internal excitation current source.
Set Up Channel List Parameters

olDaSetChannelListSize

Specify the size of the channel list, gain list, channel inhibit list, and synchronous digital I/O list.

olDaSetChannelListEntry

Set up the channel list for the subsystem.

olDaSetGainListEntry

Specify the gain for each channel in the channel list (the gain list parallels the channel list). Use a gain of 1 for channels that do not support programmable gain.

olDaSetChannelListEntryInhibit

Enable or disable inhibition for the specified channel entries. If inhibited, the acquired values from the specified entries are discarded.

olDaSetSynchronousDigitalIOUsage

Enable/disable a synchronous digital output operation. If using a DOUT subsystem that is currently in use, the handle to the DOUT subsystem must be freed with olDaReleaseDASS before configuring the this subsystem with olDaConfig.

olDaSetDigitalIOListEntry

For A/D subsystems only, specify the values to output to the dynamic digital output channels as each entry in the channel list is sampled.
Set Clocks, Triggers, and Pre-Triggers

Using an internal clock?

Yes

Specify OL_CLK_INTERNAL to select the internal clock or OL_CLK_EXTRA to select an extra available internal clock.

olDaSetClockSource

olDaSetClockFrequency

No

Specify the frequency of the internal clock. The driver sets the actual frequency as closely as possible to the number specified.

olDaSetClockSource

olDaSetExternalClockDivider

Specify OL_CLK_EXTERNAL to select the external clock or OL_CLK_EXTRA to select an extra available external clock.

Specify a clock divider to apply to the external clock source. The driver sets the actual clock divider as closely as possible to the number specified.

olDaSetTrigger

Specify the source of the trigger to start post-trigger acquisition or output, or for A/D subsystems only, to stop pre-trigger acquisition if used. Refer to your device driver documentation for details.

Go to the next page.
Continued from previous page.

Using pre- or about-trigger mode?

Yes

oldDaSetPretriggerSource

For A/D subsystems only, specify the trigger (usually software) to start the pre-trigger or about-trigger acquisition.
**Set Up Triggered Scan**

Enable triggered scan mode.

Specify the retrigger mode: OL_RETRIGGER_INTERNAL (internal retrigger clock is the retrigger; any supported trigger source is initial trigger), OL_RETRIGGER_SCAN_PER_TRIGGER (retrigger source same as initial trigger source), or OL_RETRIGGER_EXTRA (external retrigger source is the retrigger; any supported retrigger source is the initial trigger).

Set the frequency of the retrigger clock. The driver sets the actual frequency as closely as possible to the number specified.

Specify the retrigger source. Refer to your device driver documentation for details.

Specify the number of times to scan the channel-gain list per trigger/retrigger.
Set Up Input Buffering

Using main window to handle messages?

Yes

Specify the window in which to post messages.

Specify the procedure to handle Windows messages.

Specify the buffer wrapping mode. If OL_WRP_NONE, data is written to multiple allocated input buffers continuously; when no more empty buffers are available, the operation stops (gap-free data guaranteed). If OL_WRP_MULTIPLE, data is written to multiple allocated input buffers continuously; if no more empty buffers are available, the device overwrites the data in the current buffer, starting with the first location in the buffer (data is not guaranteed to be gap-free).

Allocate more buffers?

No

Allocate more buffers?

Yes

A minimum of three buffers is recommended for continuous input operations.
Set Up Output Buffering

Using main window to handle messages?

Yes

Specify the window in which to post messages.

No

olDaSetWndHandle

Specify the procedure to handle Windows messages.

olDaSetNotificationProcedure

Specify the buffer wrapping mode. If OL_WRP_NONE, data is written from multiple output buffers continuously; when no more output buffers are available, the operation stops. If OL_WRP_MULTIPLE, data is output from multiple output buffers continuously; if no more output buffers are available, the device outputs data from the current buffer, starting with the first location in the buffer. If OL_WRP_SINGLE, data from a single output buffer is downloaded to the FIFO of the device (if supported by the device) and is written out starting from the first location of the buffer; when the end of the buffer is reached, the device starts outputting data from the first location of the buffer.

Specify the buffer wrapping mode. If OL_WRP_NONE, data is written from multiple output buffers continuously; when no more output buffers are available, the operation stops. If OL_WRP_MULTIPLE, data is output from multiple output buffers continuously; if no more output buffers are available, the device outputs data from the current buffer, starting with the first location in the buffer. If OL_WRP_SINGLE, data from a single output buffer is downloaded to the FIFO of the device (if supported by the device) and is written out starting from the first location of the buffer; when the end of the buffer is reached, the device starts outputting data from the first location of the buffer.

olDaSetWrapMode

olDmAllocBuffer, olDmMallocBuffer, or olDmCallocBuffer

Fill the buffer.

olDmSetValidSamples

Allocate more buffers?

Yes

olDaPutBuffer

Put the buffer on the ready queue.

olDaSetNotificationProcedure

Allocate more buffers?
Deal with Messages and Buffers for Input Operations

- Get error message?
  - Yes: Report the error.
  - No: Get buffer reused message?
    - Yes: You may want to increment a counter.
    - No: Get queue done message?
      - Yes: Report that the operation has stopped. You might also want to clean up the subsystem (see page 183).
      - No: Get buffer done message?
        - Yes: Process data?
          - Yes: olDaGetBuffer
          - No: olDmGetValidSamples
        - No: Go to the next page.
  - No: Go to the next page.

The following error messages can be reported: OLDA_WM_OVERRUN or OLDA_WM_TRIGGER_ERROR.

The buffer reused message is OLDA_WM_BUFFER_REUSED.

The queue done messages are OLDA_WM_QUEUE_DONE and OLDA_WM_QUEUE_STOPPED.

The buffer done message is OLDA_WM_BUFFER_DONE or OLDA_WM_PRETRIGGER_BUFFER_DONE.

Retrieve the buffer from the done queue.

Determine the number of samples in the buffer.
Continued from previous page.

Get a pointer to the buffer.

    \textbf{olDmGetBufferPtr}

Process the data/buffer in your program.

Convert the data from counts to voltage using \texttt{olDaCodeToVolts} or from voltage to counts using \texttt{olDaVoltsToCode}, if desired.

    \textbf{olDaPutBuffer}

Recycle the buffer if you want the subsystem to fill it again when using OL_WRP_NONE or OL_WRP_MULTIPLE. See page 178 if you want to transfer data from an inprocess buffer. For a burst of data, you may want to clean up after processing; refer to page 184 for more information.

Wait for message?

Yes

Return to the top of page 176.

No

Continued from previous page.
### Transfer Data from an Inprocess Buffer

Determine the number of buffers on the inprocess queue (at least one buffer must be on the inprocess queue to perform this operation).

Use `olDmAllocBuffer` to allocate a buffer of samples, where each sample is 2 bytes; use `olDmMallocBuffer` to allocate a buffer of samples of a specified size; or use `olDmMallocBuffer` to allocate a buffer in bytes.

Copy the data from the inprocess buffer to the allocated buffer for immediate processing. An OLDA_WM_BUFFER_DONE message is generated when the operation completes.

See page 176 to deal with the buffers.
Deal with Messages and Buffers for Output Operations

Get error message?

Yes → Report the error.

The following error messages can be reported: OLDA_WM_UNDERRUN or OLDA_WM_TRIGGER_ERROR.

No →

Get buffer reused message?

Yes → You may want to increment the counter.

The buffer reused message is OLDA_WM_BUFFER_REUSED.

No →

Get queue done message?

Yes → Report that the operation has stopped. You might also want to clean up the subsystem (see page 183).

The queue done messages are OLDA_WM_QUEUE_DONE and OLDA_WM_QUEUE_STOPPED.

No →

Get buffer done message?

Yes →

Refill buffers?

Yes → olDaGetBuffer

The buffer done message is OLDA_WM_BUFFER_DONE. Retrieve the buffer from the done queue.

No →

Go to the next page.

No →

Go to the next page.
Continued from previous page.

`olDmGetBufferPtr`  
Get a pointer to the buffer.

Fill the buffer.

`olDmSetValidSamples`  
Recycle the buffer if you want the subsystem to empty it again when using `OL_WRP_NONE` or `OL_WRP_MULTIPLE`.

`olDaPutBuffer`  

IO complete message returned?  
Yes: The IO complete message is `OLDA_WM_IO_COMPLETE`. It is generated when the last data point has been output from the analog output channel. Note that in some cases, this message is generated well after the data is transferred from the buffer (when the `OLDA_WM_BUFFER_DONE` and `OLDA_WM_QUEUE_DONE` messages are generated).

Wait for message?  
Yes: Return to the top of page 179.

Continued from previous page.

Wait for message?

Yes: Return to the top of page 179.
Set Clocks and Gates for Counter/Timer Operations

Using an internal clock?

Yes

Specify OL_CLK_INTERNAL to select the internal clock or OL_CLK_EXTRA to select an extra available internal clock.

Specify the frequency of the output pulse from the internal clock. The driver sets the actual frequency as closely as possible to the number specified.

No

Specify OL_CLK_EXTERNAL to select the external clock or OL_CLK_EXTRA to select an extra available external clock.

Specify a clock divider to apply to the external clock source to set the frequency of the output pulse. The driver sets the actual clock divider as closely as possible to this number.

olDaSetGateType

Specify the gate to enable or trigger a counter/timer operation. Specify OL_GATE_NONE for a software gate, OL_GATE_HIGH_LEVEL for a high-level gate, OL_GATE_LOW_LEVEL, OL_GATE_HIGH_EDGE, OL_GATE_LOW_EDGE, OL_GATE_LEVEL for any level gate, OL_GATE_HIGH_LEVEL_DEBOUNCE for a debounced high-level gate, OL_GATE_LOW_LEVEL_DEBOUNCE, OL_GATE_HIGH_EDGE_DEBOUNCE, OL_GATE_LOW_EDGE_DEBOUNCE, or OL_GATE_LEVEL_DEBOUNCE.
**Stop the Operation**

- **Pause the operation?**
  - Yes → **olDaPause**
  - No → Continue a paused operation?
    - Yes → **olDaContinue**
    - No → Stop in an orderly way?
      - Yes → **olDaStop**
      - No → Reinitialize?
        - Yes → **olDaReset**
        - No → **olDaAbort**

**olDaStop** stops the operation on the subsystem in the recommended way; the current inprocess buffers are filled or emptied and put on the done queue. The driver posts at least one buffer done and queue stopped message.

**olDaAbort** and **olDaReset** stop the operation on the subsystem immediately; the current buffers are not filled or emptied before they are put on the done queue. **olDaReset** also reinitializes the subsystem to a known state and flushes all buffers to the done queue.
Clean Up Buffered I/O Operations

olDaFlushBuffers
- Flush all buffers on the ready and/or inprocess queues to the done queue.

olDaGetQueueSize
- Determine the number of buffers on the done queue.

olDaGetBuffer
- Retrieve each buffer on the done queue.

olDmFreeBuffer
- Free each buffer retrieved from the done queue.

olDaReleaseSSList
- For simultaneous operations only, release the simultaneous start list.

olDaReleaseDASS
- Release each subsystem.

olDaTerminate
- Release the device driver and terminate the session.
Clean Up Counter/Timer Operations

- **olDaReleaseSSList**: For simultaneous operations only, release the simultaneous start list.
- **olDaReleaseDASS**: Release each subsystem.
- **olDaTerminate**: Release the device driver and terminate the session.
Product Support
Should you experience problems using the DataAcq SDK, follow these steps:

1. Read all the appropriate sections of this manual. Make sure that you have added any “Read This First” information to your manual and that you have used this information.

2. Check for a README file on the Data Acquisition OMNI CD. If present, read this file for the latest installation and usage information.

3. Check that you have installed your hardware devices properly. For information, refer to the documentation supplied with your devices.

4. Check that you have installed the device drivers for your hardware devices properly. For information, refer to the documentation supplied with your devices.

5. Check that you have installed your software properly. For information, refer to Chapter 2 starting on page 21.

If you are still having problems, Data Translation’s Technical Support Department is available to provide technical assistance. To request technical support, go to our web site at http://www.datatranslation.com and click on the Support link.

When requesting technical support, be prepared to provide the following information:

- Your product serial number
- The hardware/software product you need help on
- The version of the CD you are using
- Your contract number, if applicable

If you are located outside the USA, contact your local distributor; see our web site (www.datatranslation.com) for the name and telephone number of your nearest distributor.
Sample Code

Single-Value Analog Input ......................... 188
Continuous Analog Input ......................... 193
Single-Value Analog Input

The following code fragments illustrate the steps required to perform a single-value analog input operation. Refer to the example program svadc.c in the directory C:\Program Files\Data Translation\Win32\SDK\Examples\ for the entire program.

This program calls a user-defined function called GetDriver(), which enumerates the devices installed in the system.

Declare Variables and User Functions

This code fragment defines the variables used and the user-defined GetDriver() function; note that this program uses the device’s default values for channel type, resolution, data encoding, range, and channel filter.

typedef struct tag_board {
    HDRVR hdrvr;        /* driver handle  */
    HDASS hdass;        /* subsystem handle */
    ECODE status;       /* board error status */
    char name[STRLEN];  /* string for board name */
    char entry[STRLEN]; /* string for board name */
} BOARD;

typedef BOARD FAR* LPBOARD;

static BOARD board;

BOOL __export FAR PASCAL GetDriver(lpszName,lpszEntry,lParam)

    LPSTR lpszName;     /* board name */
    LPSTR lpszEntry;    /* system.ini entry */
    LPARAM lParam;      /* optional user data */
    UINT channel = 0;
    DBL gain = 1.0;
Sample Code

**Initialize the Driver**

The following code fragment, in `WinMain()`, calls the CHECKERROR error handler macro and the `olDaEnumBoards` function, which initializes the first available DT-Open Layers device. `olDaEnumBoards` calls `GetDriver()`, which lists the name of the device:

```c
DBL min,max;
float volts;
long value;
UINT encoding,resolution;

board.hdrvr = NULL;
CHECKERROR(olDaEnumBoards(GetDriver,
   (LPARAM) (LPBOARD) &board));
```

This code fragment is in `GetDriver()` and gets the device name:

```c
{
LPBOARD lboard = (LPBOARD)(LPVOID)lParam;
/* fill in board strings */
lstrcpyn(lboard->name,lpszName,STRLEN);
lstrcpyn(lboard->entry,lpszEntry,STRLEN);
```

This code is in `WinMain()` and checks for errors within the callback function:

```c
CHECKERROR (board.status);
/* check for NULL driver handle - means no boards */
if (board.hdrvr == NULL){
   MessageBox(HWND_DESKTOP, "No DT-Open Layer boards!!!", "Error",
      MB_ICONEXCLAMATION | MB_OK);
return ((UINT)NULL);
}
```
This code fragment is in **WinMain()** and initializes the device:

```c
lpboard->status = olDaInitialize(lpszName,
       &lpboard->hdrvr);
if   (lpboard->hdrvr != NULL)
    return FALSE;
/* false to stop enumerating */
else
    return TRUE;           /* true to continue */
```

**Get a Handle to the Subsystem**

The following code fragment gets a handle to the A/D subsystem and checks for errors:

`CHECKERROR(olDaGetDASS(board.hdrvr, OLSS_AD, 0,
                   &board.hdass));`

**Set the DataFlow to Single Value**

The following code fragment sets the dataflow mode of the A/D subsystem to single value and checks for errors.

`CHECKERROR (olDaSetDataFlow(board.hdass,
                              OL_DF_SINGLEVALUE));`

**Configure the Subsystem**

The following code fragment configures the A/D subsystem and checks for errors.

`CHECKERROR (olDaConfig(board.hdass));`
Acquire a Single Value

The following code fragment acquires a single analog input value from channel 0 of the A/D subsystem (using a gain of 1) and checks for errors.

```c
CHECKERROR (olDaGetSingleValue(board.hdass, &value, channel, gain));
```

Convert the Value to Voltage

The following code fragment uses the default range, encoding, and resolution of the A/D subsystem to convert the acquired value into voltage and to check for errors. Note that this step is optional.

```c
CHECKERROR (olDaGetRange(board.hdass,&max,&min));
CHECKERROR (olDaGetEncoding(board.hdass, &encoding));
CHECKERROR (olDaGetResolution(board.hdass, &resolution));

/* Convert value to volts */
if (encoding != OL_ENC_BINARY)
{
    /* convert to offset binary by inverting the */
    /* sign bit */
    value ^= 1L << (resolution-1);

    /* zero upper bits */
    value &= (1L << resolution) - 1;
}
volts=(float)max-(float)min)/(1L<<resolution)*
    value+float)min;

/* display value with message box */
Appendix A

```c
sprintf(str,"Single Value AD Op.\nADC Input = %.3f V", volts);
MessageBox(HWND_DESKTOP, str, board.name,
    MB_ICONINFORMATION | MB_OK);

Release the Subsystem and Terminate the Session

The following code fragment releases the A/D subsystem, terminates the session, and checks for errors:

CHECKERROR (olDaReleaseDASS(board.hdass));
CHECKERROR (olDaTerminate(board.hdrvr));

Handle Errors

The following code fragment handles the errors from the DataAcq SDK and displays the error codes. Note that this step is optional but recommended.

#define STRLEN 80 /* String size for general text*/
    /* manipulation. */
char str[STRLEN]; /* Global string for general */
    /* text manipulation */

#define SHOW_ERROR(ecode) 
    MessageBox(HWND_DESKTOP,olDaGetErrorString(ecode,
        str,STRLEN),"Error",
        MB_ICONEXCLAMATION|MB_OK);

#define CHECKERROR(ecode) \
if ((board.status = (ecode)) != OLNOERROR) \
{ \
    SHOW_ERROR(board.status);
    olDaReleaseDASS(board.hdass); \
    olDaTerminate(board.hdrvr); \
    return ((UINT)NULL); 
}
```
Continuous Analog Input

The following code fragments illustrate the steps required to perform a continuous (post-trigger) analog input operation. Refer to the example program contadc.c in the directory C:\Program Files\Data Translation\Win32\SDK\Examples\ for the entire program.

This program calls two user-defined functions: GetDriver(), which enumerates the devices installed in the system, and OutputBox(), which creates a dialog box to handle information and error window messages from the A/D subsystem.

Declare Variables and User Functions

This code fragment defines the variables used and the user-defined GetDriver() function; note that this program uses the device’s default values for channel type, resolution, data encoding, range, and channel filter.

typedef struct tag_board {
    HDRVR hdrvr;        /* driver handle */
    HDASS hdass;        /* subsystem handle */
    ECODE status;       /* board error status */
    HBUF  hbuf;         /* subsystem buffer handle */
    WORD FAR* lpbuf;    /* buffer pointer */
    char name[STRLEN];  /* string for board name */
    char entry[STRLEN]; /* string for board name */
} BOARD;

typedef BOARD FAR* LPBOARD;

static BOARD board;
static ULNG count = 0;
BOOL __export FAR PASCAL GetDriver(lpszName, pszEntry, lParam)
LPSTR lpszName;      /* board name         */
LPSTR lpszEntry;     /* system.ini entry   */
LPARAM lParam;        /* optional user data */

BOOL __export FAR PASCAL InputBox(hDlg, message,
    wParam, lParam)

HWND   hDlg;
/* window handle of the dialog box */
UINT   message;
/* type of message */
WPARAM wParam;
/* message-specific information */
LPARAM lParam;

DBL min,max;
float volts;
long value;
ULNG samples;
UINT encoding,resolution;

DBL freq;
UINT size,dma,gainsup;
int i;

UINT channel = 0;
DBL gain = 1.0;
Initialize the Driver

The following code fragment, in `WinMain()`, calls the CHECKERROR error handler macro and the `olDaEnumBoards` function, which initializes the first available DT-Open Layers device. `olDaEnumBoards` calls `GetDriver()`, which lists the name of the device:

```c
board.hdrv = NULL;
CHECKERROR(olDaEnumBoards(GetDriver,
   (LPARAM)(LPBOARD)&board));
CHECKERROR (board.status);

/* check for NULL driver handle - means no boards */
if (board.hdrv == NULL){
   MessageBox(HWND_DESKTOP, " No Open Layer boards!!",
   "Error", MB_ICONEXCLAMATION | MB_OK);
return ((UINT)NULL);
}

This code is in `GetDriver()` and gets the device name:

```c
{
LPBOARD lpboard = (LPBOARD)(LPVOID)lParam;

/* fill in board strings */
lstrcpyn(lpboard->name,lpszName,STRLEN);
lstrcpyn(lpboard->entry,lpszEntry,STRLEN);
```
This code is in `WinMain()` and initializes the device:

```c
lpboard->status = oDaInitialize(lpszName, 
    &lpboard->hdrvr);
if   (lpboard->hdrvr != NULL)
    return FALSE; /* false to stop enumerating */
else
    return TRUE; /* true to continue */
}
```

### Get a Handle to the Subsystem

The following code fragment gets a handle to the A/D subsystem and checks for errors:

```c
CHECKERROR (oDaGetDASS(board.hdrvr, OLSS_AD, 0, 
    &board.hdass));
```

### Set the DataFlow to Continuous

The following code fragment sets the dataflow mode of the A/D subsystem to single value and checks for errors:

```c
CHECKERROR (oDaSetDataFlow(board.hdass, 
    OL_DF_CONTINUOUS));
```

### Specify the Channel List and Channel Parameters

The following code fragment specifies a channel-gain list size of 1; specifies channel 0 in the channel-gain list; if the subsystem supports programmable gain, specifies a gain of 1 for this entry; and checks for errors:

```c
/* Specify a channel-list size of 1.*/
CHECKERROR (oDaSetChannelListSize(board.hdass,1));

/* Specify a channel 0 in the channel list.*/
CHECKERROR (olDaSetChannelListEntry(board.hdass, 0, channel));

/* Check if the subsystem supports programmable */
/* gain. */
CHECKERROR (olDaGetSSCaps(board.hdass, OLSSC_SUP_PROGRAMGAIN, gainsup));
/* Set the gain for entry 0 in the channel list */
/* if the board supports it. */
if (gainsup)
    CHECKERROR (olDaSetGainListEntry(board.hdass, 0, gain));

Specify the Clocks

The following code fragment specifies the frequency of the internal A/D sample clock and checks for errors:

/* Check the maximum frequency for the internal */
/* clock */
CHECKERROR(olDaGetSSCapsEx(board.hdass, OLSSCE_MAXTHROUGHPUT, &freq));

/* set 1000 Hz frequency */
freq = min (1000.0, freq);
CHECKERROR (olDaSetClockFrequency(board.hdass, freq));
Specify DMA Usage

The following code fragment specifies one DMA channel for the A/D subsystem and checks for errors:

```c
/* Check the number of DMA channels supported. */
CHECKERROR(olDaGetSSCaps(board.hdass,
    OLSSC_NUMDMACHANS, &dma));

/* Set one dma channel.*/
dma = min (1, dma);
CHECKERROR (olDaSetDmaUsage(board.hdass,dma));
```

Set Up Window Handle and Buffering

The following code fragment specifies a handle to the message window, set up buffers, and check for errors:

```c
/* Specify window handle. */
CLOSEONERROR (olDaSetWndHandle(board.hdass, hDlg,
    (UINT)NULL));

/*Specify the buffer wrap mode as multiple */
CHECKERROR (olDaSetWrapMode(board.hdass,
    OL_WRP_MULTIPLE));

/*Specify the buffers and put them on the ready */
/*queue. */
size = (UINT)freq/10;
/* Specify the buffer size. */

/* Allocate three input buffers and */
/* put the buffers on the ready queue. */
```
for (i=0;i<3;i++)
{
    CHECKERROR (olDmCallocBuffer(0,0,(ULNG) size,
        2,&board.hbuf));
    CHECKERROR (olDmGetBufferPtr(board.hbuf,
        (LPVOID FAR*) &board.lpbuf));
    CHECKERROR (olDaPutBuffer(board.hdass,
        board.hbuf));
}

**Configure the Subsystem**

The following code fragment configures the A/D subsystem and checks for errors.

CHECKERROR (olDaConfig(board.hdass));

**Start the Continuous Analog Input Operation**

The following code fragment acquires a single analog input value from channel 0 of the A/D subsystem (using a gain of 1) and checks for errors:

CLOSEONERROR (olDaStart(board.hdass));

**Deal with Messages and Buffers**

The following code fragment deals with messages and buffers for the A/D subsystem:

/* Use a dialog box to collect information */
/* and error messages from the subsystem. */
DialogBox(hInstance, (LPCSTR)INPUTBOX,
    HWND_DESKTOP, InputBox);

/* This function processes messages for the */
/* input dialog box. */
switch (message) {

    case WM_INITDIALOG:
        /* message: initialize dialog box*/
        /* set the title to the board name */
        SetWindowText(hWnd, board.name);
        return (TRUE); /* A message was returned. */

    case OLDA_WM_BUFFER_REUSED:
        /* message: buffer reused*/
        break;

    case OLDA_WM_BUFFER_DONE:
        /* message: buffer done*/

        /* Get buffer off the done queue. */
        CHECKERROR (olDaGetBuffer(board.hdass,
                        &board.hbuf));
        /*If there is a buffer, get subsystem */
        /*information for code to volts conversion */
        if (board.hbuf != NULL){
            CLOSEONERROR (olDaGetRange(board.hdass,
                          &max,&min));
            CLOSEONERROR (olDaGetEncoding(board.hdass,
                          &encoding));
            CLOSEONERROR (olDaGetResolution (board.hdass,
                          &resolution));

            /* get max samples in input buffer */
            CLOSEONERROR (olDmGetMaxSamples(board.hbuf,
                          &samples));

            /* get last sample in buffer */
            value = board.lpbuf[samples-1];

            /* Get pointer to buffer data */
            CHECKERROR (olDmGetBufferPtr (board.hbuf,
                            (LPVOID FAR*) &board.lpbuf));

}
/* Put buffer back on the ready queue. */
CHECKERROR (olDaPutBuffer(board.hdass, 
    board.hbuf));

case OLDA_WM_QUEUE_DONE:
/* using wrap multiple or none */
/* if this message is received, */
/* acquisition has stopped. */
EndDialog(hDlg, TRUE);

case OLDA_WM_QUEUE_STOPPED:
/* using wrap multiple or none */
/* if this message is received, */
/* acquisition has stopped. */
EndDialog(hDlg, TRUE);

case OLDA_WM_TRIGGER_ERROR:
/* Process trigger error message */
MessageBox(hDlg,"Trigger error: acquisition 
    stopped", "Error", MB_ICONEXCLAMATION | 
    MB_OK);
EndDialog(hDlg, TRUE);

case OLDA_WM_OVERRUN_ERROR:
/* Process underrun error message */
MessageBox(hDlg,"Input overrun error: 
    acquisition stopped", "Error", 
    MB_ICONEXCLAMATION | MB_OK);
EndDialog(hDlg, TRUE);

case WM_COMMAND:
/* message: received a command */
#ifdef WIN32
switch ( LOWORD(wParam) ) {
#else
switch (wParam) {

Appendix A

```c
#ifndef
case IDOK:
case IDCANCEL:
    CLOSEONERROR (olDaAbort(board.hdass));
    EndDialog(hDlg, TRUE);
    return (TRUE); /* Did process a message. */
}
break;
}
return (FALSE); /* Didn't get a message */
```

**Convert Values to Voltage**

The following code fragment uses the default range, encoding, and resolution of the A/D subsystem to convert the acquired values into voltages and to check for errors. Note that this step is optional.

```c
/* convert value to volts */
if (encoding != OL_ENC_BINARY) {

    /* convert to offset binary by inverting */
    /* sign bit */
    value ^= 1L << (resolution-1);

    /* zero upper bits */
    value &= (1L << resolution) - 1;
}
volts = ((float)max-(float)min)/(1L<<resolution) * 
    value + (float)min;

/* display value */
sprintf(str,"%.3f Volts",volts);
SetDlgItemText (hDlg, IDD_VALUE, str);
```

202
Clean Up

The following code fragment flushes the buffers, releases the subsystem, and terminates the program when the continuous A/D operation is complete:

```c
/* Get the input buffers from the done queue and */
/* free the input buffers */
CHECKERROR (olDaFlushBuffers(board.hdass));

for (i=0;i<3;i++)
{
    CHECKERROR (olDaGetBuffer(board.hdass, &board.hbuf));
    CHECKERROR (olDmFreeBuffer(board.hbuf));
}

/* release the subsystem and terminate */
/* the session */
CHECKERROR (olDaReleaseDASS(board.hdass));
CHECKERROR (olDaTerminate(board.hdrvr));
```

Handle Errors

The following code fragment handles the errors from the DataAcq SDK and displays the error codes. Note that this step is optional but recommended.

```c
/* Error handling macros */

#define STRLEN 80 /* String size for general */
    /* text manipulation */
char str[STRLEN]; /* Global string for general */
    /* text manipulation. */
```
#define SHOW_ERROR(ecode)
MessageBox(HWND_DESKTOP,olDaGetErrorString(ecode,
    (str,STRLEN),"Error", MB_ICONEXCLAMATION |
    MB_OK);
#define CHECKERROR(ecode)
    if ((board.status = (code)) != OLNOERROR)
    {
        SHOW_ERROR(board.status);
        olDaReleaseDASS(board.hdass);
        olDaTerminate(board.hdrvr);
        return ((UINT)NULL);
    }
#define CLOSEONERROR(ecode)
    if ((board.status = (code)) != OLNOERROR)
    {
        SHOW_ERROR(board.status);
        olDaReleaseDASS(board.hdass);
        olDaTerminate(board.hdrvr);
        EndDialog(hDlg, TRUE);
        return (TRUE);"}
### Index

**A**

A/D subsystem 56
aborted an operation
A/D 75
continuous edge-to-edge measurement 120
D/A 75
digital input 75
digital output 75
edge-to-edge measurement 116
event counting 108
rate generation 121
repetitive one-shot 126
simultaneous 140
up/down counting 110
about-trigger mode 79
AC coupling 91
aliasing 86
analog event trigger 90
analog input 59
buffers 95
channel list size 63
channel type 61
channels in the channel list 64
clock sources 85
coupling type 91
data encoding 59
data flow 72
DMA channels 104
excitation current sources 92
filters 71
gain 69
inhibiting channels 65
ranges 68
resolution 60
single channel 62
synchronous digital I/O values in the channel list 66
thermocouples 92
trigger sources 88
triggered scan mode 81
analog input channel configuration
differential 61
pseudo-differential 62
single-ended 61
analog output 59
buffers 95
channel list size 63
channel type 61
channels in the channel list 64
clock sources 85
data encoding 59
data flow 72
DMA channels 104
filters 71
gain 69
ranges 68
resolution 60
single channel 62
trigger sources 88
analog threshold (negative) trigger 90
analog threshold (positive) trigger 89
architecture 53
autoranging 44, 73
B
binary data format 59
buffer list management functions 50
buffers 95
done queue 99
flowcharts 176, 179
ready queue 96
transferring data from an inprocess buffer 178
wrap modes 102

C
C/T clock sources 128
external C/T clock 129
extra C/T clock 131
internal C/T clock 129
internally cascaded C/T clock 130
C/T subsystem 56
calling conventions 19
capabilities
used with oIDaGetSSCaps 25
used with oIDaGetSSCapsEx 34
cascading counters 130
channel list
inhibition 65
size 63
specification 64
specifying synchronous digital I/O values 66
channel type 61
channels
setting up parameters 169
CJC source
external 94
internal 94
CJC values in data stream 95
cleaning up operations
A/D 183
counter/timer 184
D/A 183
digital I/O 183
clock input signal 106
clock sources
external 86, 154
external C/T clock 129
extra 87
extra C/T clock 131
internal 85, 158, 160
internal C/T clock 129
internally cascaded C/T clock 130
configuration functions 36
configuring a subsystem 57
continuing an operation
A/D 76
D/A 76
digital input 76
digital output 76
continuous edge-to-edge measurement operations
flowcharts 160
continuous operations 75
continuous (post-trigger) mode 76
continuous about-trigger mode 79
continuous pre-trigger mode 78
flowcharts for A/D 148
flowcharts for D/A 150
flowcharts for digital input 148
flowcharts for digital output 150
sample code for continuous A/D 193
continuous pulse output 120
conventions used 10
conventions, calling 19
conversion rate 82, 83, 85
correcting thermocouple data 94
counter/timer operations 106
   C/T clock sources 128
      channels 106
duty cycle 136
gate types 131
operation modes 107
pulse output types 136
counting events 108, 109, 115, 117
coupling type 91
current sources and values 92

D

D/A subsystem 56
data acquisition functions 22
   configuration 36
   conversion 47
   information 22
   initialization and termination 35
   operation 43
data buffers 95
data conversion functions 47
data encoding 59
data flow modes 72
data management functions 48
   buffer list management 50
   buffer management 48
data transfer, from an inprocess buffer 178
DC coupling 91
debounced gate type
   any level 135
   high-edge 134
   high-level 134
   low-edge 135
   low-level 134
device, initialization 55
differential inputs 61
digital event trigger 90
digital input 59
   buffers 95
   channel list size 63
   channel type 61
clock sources 85
data flow 72
DMA channels 104
gain 69
   resolution 60
   single channel 62
   specifying the channels in the
      channel list 64
   trigger sources 88
digital output 59
   buffers 95
   channel list size 63
   channel type 61
   clock sources 85
   data flow 72
DMA channels 104
gain 69
   resolution 60
   single channel 62
   specifying the channels in the
      channel list 64
   trigger sources 88
DIN subsystem 56
DMA channels 104
done queue 99
DOUT subsystem 56
DT-Open Layers 14
duty cycle 136
E
edge-to-edge measurement operations 115
  continuous 117
  flowcharts 158
encoding 59
error checking 142
error codes 57
event counting operations 108
  flowcharts 152
event trigger
  analog 90
  digital 90
  timer 91
example programs 17
excitation current sources and values 92
external analog threshold (negative) trigger 90
external analog threshold (positive) trigger 89
external C/T clock 129
external clock 154
  source 86
external digital (TTL) trigger 89
external excitation current source 92
external retrigger mode 82
extra C/T clock 131
extra clock sources 87
extra retrigger mode 84
extra trigger sources 91

F
filters 71
floating-point values 94
flowcharts 141
  cleaning up A/D operations 183
  cleaning up counter/timer operations 184
  cleaning up D/A operations 183
  cleaning up digital I/O operations 183
continuous A/D operations 148
continuous D/A operations 150
continuous digital input operations 148
continuous digital output operations 150
continuous edge-to-edge measurement operations 160
dealing with messages and buffers 176, 179
edge-to-edge measurement operations 158
event counting operations 152
frequency measurement operations 156
pulse output operations 162
quadrature decoder 164
setting up buffers for A/D operations 174
setting up buffers for D/A operations 175
setting up buffers for digital input operations 174
setting up buffers for digital output operations 175
setting up channel lists 170
setting up channel parameters 169
setting up clocks 171, 181
setting up gates 181
setting up pre-triggers 171
setting up subsystem parameters 168
setting up triggered scans 173
setting up triggers 171
simultaneous operations 166
single-value input operations 143
single-value output operations 146
stopping operations 182
transferring data from an inprocess buffer 178
up/down counting operations 154
frequency measurement operations 111
flowcharts 156
using a pulse of a known duration 112
using the Windows timer 111
functions
buffer list management 50
buffer management 48
configuration 36
data conversion 47
information 22
initialization and termination 35
operation 43
summary 21

G
gain list 70
gains 69
gate input signal 106, 131
gate types 131
any level 133
high-edge 133
high-edge, debounced 134
high-level 132
high-level, debounced 134
level, debounced 135
low-edge 133
low-edge, debounced 135
low-level 132
low-level, debounced 134
software 132

H
handling errors 57
handling messages 58
help, launching online 16
high-edge gate type 133
high-edge, debounced gate type 134
high-level debounced gate type 134
high-level gate type 132
high-to-low pulse output 136

I
IEPE inputs 91, 92
information functions 22
inhibiting channels 65
initialization functions 35
initializing a device 55
inprocess buffer, transferring data 178
inprocess queue 97
input configuration
differential analog 61
single-ended analog 61
internal C/T clock 129
internal clock 158, 160
sources 85
internal excitation current source and values 92
internal retrigger mode 83
internal trigger 88
interrupts 104

L
level gate type 133
level, debounced gate type 135
linearization 94
list management functions 50
LongtoFreq macro 157, 159
low-edge gate type 133
low-edge, debounced gate type 135
low-level gate type 132
low-level, debounced gate type 134
low-to-high pulse output 136

M
macro 157, 159
measuring frequency 111
message handling 58
flowcharts for A/D 176
flowcharts for D/A 179
flowcharts for digital input 176
flowcharts for digital output 179
messages
OLDA_WM_BUFFER_DONE 99
OLDA_WM_BUFFER_REUSED 103
OLDA_WM_MEASURE_DONE 112
OLDA_WM_PRETRIGGER_BUFFER_DONE 99
OLDA_WM_QUEUE_DONE 103
OLDA_WM_QUEUE_STOPPED 99
multiple wrap mode 103

N
no wrap mode 103

Nyquist Theorem 86

O
OL_ENUM_EXCITATION_CURRENT_VALUES 92
OL_ENUM_FILTERS 71
OL_ENUM_GAINS 70
OL_ENUM_RANGES 68
OLDA_WM_BUFFER_DONE 99, 176, 179
OLDA_WM_BUFFER_REUSED 103, 176, 179
OLDA_WM_IO_COMPLETE 99, 180
OLDA_WM_MEASURE_DONE 112
OLDA_WM_OVERRUN 176
OLDA_WM_PRETRIGGER_BUFFER_DONE 99, 176
OLDA_WM_QUEUE_DONE 103, 176, 179
OLDA_WM_QUEUE_STOPPED 99, 176, 179
OLDA_WM_TRIGGER_ERROR 82, 85, 176, 179
OLDA_WM_UNDERRUN 179
olDaAbort 45, 182
in A/D operations 75
in continuous edge-to-edge measurement operations 120
in D/A operations 75
in digital input operations 75
in digital output operations 75
in edge-to-edge measurement operations 116
in event counting operations 108
in rate generation operations 121
in repetitive one-shot operations 126
Index

in simultaneous operations 140
in up/down counting operations 110
olDaCodeToVolts 47, 73, 101, 177
olDaConfig 45, 57
  in continuous A/D operations 149
  in continuous D/A operations 151
  in continuous digital input operations 149
  in continuous digital output operations 151
  in edge-to-edge measurement operations 158, 160
  in event counting operations 152, 164
  in frequency measurement operations 157
  in pulse output operations 163
  in single-value operations 143, 146
  in up/down counting operations 154
olDaContinue 45, 182
  in A/D operations 76
  in D/A operations 76
  in digital input operations 76
  in digital output operations 76
olDaEnumBoards 22, 55, 142
olDaEnumBoardsEx 22
olDaEnumLists 50
olDaEnumSSCaps 23, 142
olDaEnumSSLList 24
olDaEnumSubSystems 23, 56, 142
olDaFlushBuffers 45, 102, 183
olDaFlushFromBufferInprocess 46, 98, 101, 178
olDaFreeBuffer 102
olDaGetBoardInfo 23, 55
olDaGetBuffer 45, 101, 183
  in continuous A/D operations 176
  in continuous D/A operations 179
  in continuous digital input operations 176
  in continuous digital output operations 179
olDaGetCascadeMode 42
olDaGetChannelFilter 39
olDaGetChannelListEntry 38
olDaGetChannelListEntryInhibit 38
olDaGetChannelListSize 38
olDaGetChannelRange 39
olDaGetChannelType 39
olDaGetCjcTemperature 44, 73, 144
olDaGetCjcTemperatures 44, 74, 144
olDaGetClockFrequency 40
olDaGetClockSource 40
olDaGetCouplingType 41
olDaGetCTMode 42
olDaGetDASS 35, 56
  in continuous A/D operations 148
  in continuous D/A operations 150
  in continuous digital input operations 148
  in continuous digital output operations 150
  in edge-to-edge measurement operations 158, 160
  in event counting operations 152, 164
  in frequency measurement operations 156
  in pulse output operations 162
  in single-value operations 143, 146
  in up/down counting operations 154
olDaGetDASSInfo 23
olDaGetDataFlow 36
olDaGetDevCaps 23, 56, 142
olDaGetDeviceName 23
olDaGetDigitalIOListEntry 38
Index

olDaGetDmaUsage 37
olDaGetDriverVersion 24
olDaGetEncoding 39
olDaGetErrorString 24
olDaGetExcitationCurrentValue 41
olDaGetExternalClockDivider 40
olDaGetGainListEntry 38
olDaGetGateType 42
olDaGetMeasureStartEdge 42
olDaGetMeasureStopEdge 43
olDaGetMultiscanCount 37
olDaGetNotificationProcedure 36
olDaGetPretriggerSource 40
olDaGetPulseType 42
olDaGetPulseWidth 42
olDaGetQueueSize 24, 178, 183
olDaGetRange 39
olDaGetResolution 39
olDaGetRetrigger 40
olDaGetRetriggerFrequency 37
olDaGetRetriggerMode 37
olDaGetReturnCjcTemperatureInStream 42
olDaGetSingleFloat 44, 73, 144
olDaGetSingleFloats 44, 74, 144
olDaGetSingleValue 43, 73, 144
olDaGetSingleValueEx 44, 73, 144
olDaGetSingleValues 43, 74, 144
olDaGetSSCaps 23, 142
olDaGetSSCapsEx 23, 142
olDaGetSSLList 46, 139, 166
olDaGetSynchronousDigitalIOUsage 38
olDaGetThermocoupleType 41
olDaGetTrigger 40
olDaGetTriggeredScanUsage 37
olDaGetVersion 24
olDaGetWndHandle 36
olDaGetWrapMode 37
olDaInitialize 35, 55
olDaIsRunning 24, 76
olDaMeasureFrequency 46, 111, 157
olDaPause 45, 182
olDaPutBuffer 45, 96, 101
olDaPutDassToSSLList 46, 139, 166
olDaPutSingleValue 44, 74, 147
olDaPutSingleValues 45, 75, 147

olDaGetWndHandle 36
olDaGetWrapMode 37
olDaInitialize 35, 55
olDaIsRunning 24, 76
olDaMeasureFrequency 46, 111, 157
olDaPause 45, 182
olDaPutBuffer 45, 96, 101
olDaPutDassToSSLList 46, 139, 166
olDaPutSingleValue 44, 74, 147
olDaPutSingleValues 45, 75, 147

in continuous A/D operations 148
in continuous D/A operations 150
in continuous digital input operations 148
in continuous digital output operations 150
in edge-to-edge measurement operations 158, 160
in event counting operations 152, 164
in frequency measurement operations 156
in pulse output operations 162
in single-value operations 143, 146
in up/down counting operations 154

olDaGetDmaUsage 37
olDaGetDriverVersion 24
olDaGetEncoding 39
olDaGetErrorString 24
olDaGetExcitationCurrentValue 41
olDaGetExternalClockDivider 40
olDaGetGainListEntry 38
olDaGetGateType 42
olDaGetMeasureStartEdge 42
olDaGetMeasureStopEdge 43
olDaGetMultiscanCount 37
olDaGetNotificationProcedure 36
olDaGetPretriggerSource 40
olDaGetPulseType 42
olDaGetPulseWidth 42
olDaGetQueueSize 24, 178, 183
olDaGetRange 39
olDaGetResolution 39
olDaGetRetrigger 40
olDaGetRetriggerFrequency 37
olDaGetRetriggerMode 37
olDaGetReturnCjcTemperatureInStream 42
olDaGetSingleFloat 44, 73, 144
olDaGetSingleFloats 44, 74, 144
olDaGetSingleValue 43, 73, 144
olDaGetSingleValueEx 44, 73, 144
olDaGetSingleValues 43, 74, 144
olDaGetSSCaps 23, 142
olDaGetSSCapsEx 23, 142
olDaGetSSLList 46, 139, 166
olDaGetSynchronousDigitalIOUsage 38
olDaGetThermocoupleType 41
olDaGetTrigger 40
olDaGetTriggeredScanUsage 37
olDaGetVersion 24
olDaReadEvents 46, 108, 110, 116, 119, 153, 155, 165
in continuous edge-to-edge measurement operations 161

olDaReleaseDASS 36, 58, 140
in A/D operations 183
in counter/timer operations 184
in D/A operations 183
in digital input operations 183
in digital output operations 183
in single-value operations 145, 147

olDaReleaseSSList 46, 140, 183, 184

olDaReset 45, 182
in A/D operations 75
in continuous edge-to-edge measurement operations 120
in D/A operations 75
in digital input operations 75
in digital output operations 75
in edge-to-edge measurement operations 116
in event counting operations 108
in rate generation operations 121
in repetitive one-shot operations 126
in simultaneous operations 140
in up/down counting operations 110

olDaSetCascadeMode 42, 130
in event counting operations 152
in frequency measurement operations 156
in pulse output operations 162

olDaSetChannelFilter 39, 71, 169

olDaSetChannelListEntry 38, 64, 170
olDaSetChannelListEntryInhibit 38, 65, 170

olDaSetChannelListSize 38, 63, 170

olDaSetChannelRange 39, 68, 169

olDaSetChannelType 39, 61, 168

olDaSetClockFrequency 40, 86, 121, 129, 171
in C/T operations 181

olDaSetClockSource 40, 86, 87, 129, 130, 154, 158, 160, 164, 171
in C/T operations 181

olDaSetCouplingType 41, 91, 169

olDaSetCTMode 42
in continuous edge-to-edge measurement operations 118
in edge-to-edge measurement operations 116, 158, 160
in event counting operations 108, 113, 152
in frequency measurement operations 156
in one-shot mode 113
in one-shot operations 111, 123
in pulse output operations 162
in rate generation operations 121
in repetitive one-shot operations 126
in up/down counting operations 109, 154

olDaSetDataFlow 36
in continuous (post-trigger) operations 77
in continuous A/D operations 148
in continuous about-trigger operations 79
in continuous D/A operations 150
in continuous digital input operations 148
in continuous digital output operations 150
in continuous pre-trigger operations 78
Index

in single-value operations 72, 143, 146

olDaSetDigitalIOListEntry 38, 66, 170
olDaSetDmaUsage 37, 105
in continuous A/D operations 148
in continuous D/A operations 150
in continuous digital input operations 148
in continuous digital output operations 150

olDaSetEncoding 39, 60, 168
olDaSetExcitationCurrentSource 41, 92, 169
olDaSetExcitationCurrentValue 41, 92, 169
olDaSetExternalClockDivider 40, 87, 121, 130, 154, 171
in C/T operations 181

olDaSetGainListEntry 38, 70, 170
olDaSetGateType 42, 131, 132, 181
olDaSetMeasureStartEdge 42, 158, 160
olDaSetMeasureStopEdge 43, 158, 160

olDaSetMultiscanCount 37, 81, 173

olDaSetNotificationProcedure 36, 58
in continuous A/D operations 174
in continuous D/A operations 175
in continuous digital input operations 174
in continuous digital output operations 175

olDaSetPretriggerSource 40, 78, 80, 88, 172
olDaSetPulseType 42, 136, 163
olDaSetPulseWidth 42, 136, 163
olDaSetQuadDecoder 43, 138, 164

olDaSetRange 39, 68, 168
olDaSetResolution 39, 60, 168
olDaSetRetrigger 40, 84, 88, 173
olDaSetRetriggerFrequency 37, 83, 173
olDaSetRetriggerMode 37, 82, 83, 84, 173
olDaSetReturnCjcTemperatureInStream 41, 95, 168
olDaSetSynchronousDigitalIOUUsage 38, 66, 170
olDaSetThermocoupleType 41, 93, 169
olDaSetTrigger 40, 77, 88, 171
olDaSetTriggeredScanUsage 37, 81, 173
olDaSetWndHandle 36, 58
in continuous A/D operations 174
in continuous D/A operations 175
in continuous digital input operations 174
in continuous digital output operations 175

olDaSetWrapMode 37, 103
in continuous A/D operations 174
in continuous D/A operations 175
in continuous digital input operations 174
in continuous digital output operations 175

olDaSimultaneousPreStart 46, 139, 166
olDaSimultaneousStart 46, 139, 166
olDaStart 45
in A/D operations 75, 149
in D/A operations 75, 151
in digital input operations 75, 149
in digital output operations 75, 151
in edge-to-edge measurement operations 116, 159, 161
in event counting operations 108, 153, 164
in one-shot operations 124
in pulse output operations 163
in rate generation operations 121
in repetitive one-shot operations 126
in simultaneous operations 140
in up/down counting operations 110, 155

olDaStop 45, 182
in A/D operations 75
in continuous edge-to-edge measurement operations 120
in D/A operations 75
in digital input operations 75
in digital output operations 75
in edge-to-edge measurement operations 116
in event counting operations 108
in rate generation operations 121
in repetitive one-shot operations 126
in simultaneous operations 140
in up/down counting operations 110

olDaTerminate 36, 58, 140, 183
in C/T operations 184
in digital input operations 183
in digital output operations 183
in single-value operations 145, 147

olDaVoltsToCode 47, 73, 101, 177

olDmAllocBuffer 49, 96
in A/D operations 174
in D/A operations 175
in digital input operations 174
in digital output operations 175

in inprocess buffer operations 178

olDmCallocBuffer 49, 96
in A/D operations 174
in D/A operations 175
in digital input operations 174
in digital output operations 175
in inprocess buffer operations 178

olDmCopyBuffer 49
olDmCopyFromBuffer 49
olDmCopyToBuffer 49
olDmCreateList 50
olDmEnumBuffers 50
olDmFreeBuffer 49, 183
olDmFreeList 50
olDmGetBufferFromList 50
olDmGetBufferPtr 49, 101
in continuous A/D operations 177
in continuous D/A operations 180
in continuous digital input operations 177
in continuous digital output operations 180

olDmGetBufferSize 49
olDmGetDataBits 49
olDmGetDataWidth 49
olDmGetErrorString 49
olDmGetListCount 50
olDmGetListHandle 50
olDmGetListIDs 51
olDmGetMaxSamples 49
olDmGetTimeDateStamp 49
olDmGetValidSamples 49

in continuous A/D operations 176
in continuous digital input operations 176

olDmGetVersion 49
**Index**

- **olDmMallocBuffer** 49, 96
  - in A/D operations 174
  - in D/A operations 175
  - in digital input operations 174
  - in digital output operations 175
  - in inprocess buffer operations 178
- **olDmPeekBufferFromList** 51
- **olDmPutBufferToList** 51
- **olDmReAllocBuffer** 49
- **olDmReAllocBuffer** 50
- **olDmSetDataWidth** 49
- **olDmSetValidSamples** 49, 175
  - in continuous D/A operations 180
  - in continuous digital output operations 180
- **OLSSC_SUP_SEQUENTIAL_CGL** 63
- **OLSSC_SUP_EXTCLOCK** 30
- **OLSSC_CGLDEPTH** 27, 63
- **OLSSC_FIFO_SIZE_IN_K** 34, 104
- **OLSSC_FIXED_PULSE_WIDTH** 32, 136
- **OLSSC_MAXDICHANS** 28
- **OLSSC_MAXDIGITALIOLIST_VALUE** 27, 66
- **OLSSC_MAXMULTISCAN** 26, 81
- **OLSSC_MAXSECHANS** 61
- **OLSSC_NONCONTIGUOUS_CHANNELNUM** 27
- **OLSSC_NUM_DMACHANS** 105
- **OLSSC_NUM_EXCITATION_CURRENT_VALUES** 30, 92
- **OLSSC_NUM_RANGES** 68
- **OLSSC_NUM_RESOLUTION** 60
- **OLSSC_NUMCHANNELS** 28, 60
- **OLSSC_NUMDMACHANS** 26
- **OLSSC_NUMEXTRACLOCKS** 30, 87, 131
- **OLSSC_NUMEXTRATRIGGERS** 29, 91
- **OLSSC_NUMFILTERS** 28, 71
- **OLSSC_NUMGAINS** 27, 70
- **OLSSC_NUMRANGES** 28
- **OLSSC_NUMRESOLUTIONS** 28
- **OLSSC_RANDOM_CGL** 63
- **OLSSC_RETURNS_FLOATS** 31, 94
- **OLSSC_SUP_2SCOMP** 29, 59
- **OLSSC_SUP_AC_COUPLING** 30, 91
- **OLSSC_SUP_ANALOGEVENTTRIG** 29, 90
- **OLSSC_SUP_BINARY** 29, 59
- **OLSSC_SUP_BUFFERING** 25, 96
- **OLSSC_SUP_CASCADING** 32, 130
- **OLSSC_SUP_CHANNELLIST_INHIBIT** 27, 65
- **OLSSC_SUP_CJC_SOURCE_CHANNEL** 31, 94
- **OLSSC_SUP_CJC_SOURCE_INTERNAL** 31, 94
- **OLSSC_SUP_CONTINUOUS** 25, 76
- **OLSSC_SUP_CONTINUOUS_ABOUTTRIG** 25, 79
- **OLSSC_SUP_CONTINUOUS_PRETRIG** 25, 78
- **OLSSC_SUP_CTMODE_CONT_MEASURE** 32, 118
- **OLSSC_SUP_CTMODE_COUNT** 32, 108, 111
- **OLSSC_SUP_CTMODE_MEASURE** 32, 115
- **OLSSC_SUP_CTMODE_ONESHOT** 32, 123
OLSSC_SUP_CTMODE_ONESHOT_RPT 32, 126
OLSSC_SUP_CTMODE_RATE 32, 120
OLSSC_SUP_CTMODE_UP_DOWN 32, 109
OLSSC_SUP_CURRENT_OUTPUTS 28
OLSSC_SUP_DC_COUPLING 30
OLSSC_SUP_DIFFERENTIAL 28, 61
OLSSC_SUP_DIGITALEVENTTRIG 29, 90
OLSSC_SUP_EXP2896 28, 60
OLSSC_SUP_EXP727 28, 60
OLSSC_SUP_EXTCLOCK 86, 129
OLSSC_SUP_EXTERNAL_EXCITATION_CURRENT_SRC 30, 92
OLSSC_SUPEXTERNTRIG 29, 89
OLSSC_SUP_FIF0 34, 104
OLSSC_SUP_FILTERPERCHAN 28, 71
OLSSC_SUP_GAPFREE_DUALDMA 26, 105
OLSSC_SUP_GAPFREE_NODMA 26, 105
OLSSC_SUP_GAPFREE_SINGLEDMA 26, 105
OLSSC_SUP_GATE_GATE_LEVEL 33
OLSSC_SUP_GATE_HIGH_EDGE 33, 133
OLSSC_SUP_GATE_HIGH_EDGE_DEBOUNCE 33, 135
OLSSC_SUP_GATE_HIGH_LEVEL 33, 132
OLSSC_SUP_GATE_HIGH_LEVEL_DEBOUNCE 33, 134
OLSSC_SUP_GATE_LEVEL 133
OLSSC_SUP_GATE_LEVEL_DEBOUNCE 33
OLSSC_SUP_GATE_LOW_EDGE 33, 133
OLSSC_SUP_GATE_LOW_EDGE_DEBOUNCE 33
OLSSC_SUP_GATE_LOW_LEVEL 33, 132
OLSSC_SUP_GATE_LOW_LEVEL_DEBOUNCE 33
OLSSC_SUP_GATE_NONE 33, 132
OLSSC_SUP_INTERRUPT 33, 104
OLSSC_SUP_MAXRETRIGGER 34
OLSSC_SUP_MAXSECHANS 28
OLSSC_SUP_MINRETRIGGER 34
OLSSC_SUP_PAUSE 25, 76
OLSSC_SUP_PLS_HIGH2LOW 32, 136
OLSSC_SUP_PLS_LOW2HIGH 32, 136
OLSSC_SUP_PLS_LOW2HIGH 32, 136
OLSSC_SUP_POSTMESSAGE 25, 58
OLSSC_SUP_PROCESSOR 34
OLSSC_SUP_PROGRAMGAIN 27, 70
OLSSC_SUP_QUADRATURE_DECODER 32
OLSSC_SUP_RANDOM_CGL 27
OLSSC_SUP_RANGEPERCHANNEL 28, 68
OLSSC_SUP_RETRIGGER_EXTRA 27, 84
OLSSC_SUP_RETRIGGER_INTERNAL 26, 83
OLSSC_SUP_RETRIGGER_SCAN_PER_TRIGGER 26, 82
OLSSC_SUP_SEQUENTIAL_CGL 27
OLSSC_SUP_SIMULTANEOUS_CLOCKING 30, 85
OLSSC_SUP_SIMULTANEOUS_SH 27, 63, 74
OLSSC_SUP_SIMULTANEOUS_START 25, 139
OLSSC_SUP_SINGLEENDED 28, 61
OLSSC_SUP_SINGLEVALUE 25, 72
OLSSC_SUP_SINGLEVALUE_AUTORANGE 27, 73
OLSSC_SUP_SOFTTRIG 29, 88
OLSSC_SUP_SV_NEG_EXTERN_TTL TRIG 89
OLSSC_SUP_SV_POS_EXTERN_TTL TRIG 29, 89
OLSSC_SUP_SWCAL 34
OLSSC_SUP_SWRESOLUTION 28, 60
OLSSC_SUP_SYNCHRONOUS_DIGITALIO 27
OLSSC_SUP_SYNCHRONOUS_DIGITALIO 66
OLSSC_SUP_TEMPERATURE_DATA_IN_STREAM 31, 94
OLSSC_SUP_THERMOCOUPLES 31, 92
OLSSC_SUP_THRESHTRIGNEG 29, 90
OLSSC_SUP_THRESHTRIGPOS 29, 89
OLSSC_SUP_TIMEREVENTTRIG 29, 91
OLSSC_SUP_TRIGSCAN 26, 81
OLSSC_SUP_WAVEFORM_MODE 25
OLSSC_SUP_WRPMULTIPLE 25, 104
OLSSC_SUP_WRP_SINGLE 25, 104
OLSSC_SUP_WRPWAVEFORM_ONLY 26, 104
OLSSC_SUP_ZEROSEQUENTIAL_CGL 27, 63
OLSSCE_BASECLOCK 34
OLSSCE_CJC_MILLIVOLTS_PER_DEGREE_C 35
OLSSCE_MAXCLOCKDIVIDER 34, 87, 130
OLSSCE_MAXTHROUGHPUT 34, 86, 129
OLSSCE_MINCLOCKDIVIDER 34, 87, 130
OLSSCE_MINTHROUGHPUT 34, 86, 129

one-shot mode 123
online help, launching 16
operation functions 43
operation modes, counter/timer 107
continuous edge-to-edge measurement 117
display measurement 115
event counting 108
frequency measurement 111
one-shot pulse output 123
rate generation 120
repetitive one-shot pulse output 126
up/down counting 109
operations
analog input 59
analog output 59
counter/timer 106
digital input 59
digital output 59
quadrature decoder 138
simultaneous 139
outputting pulses
  continuously 120
  one-shot 123
  repetitive one-shot 126
overview of the DataAcq SDK 14

P
parameters
  setting up buffers for A/D operations 174
  setting up buffers for D/A operations 175
  setting up buffers for digital input operations 174
  setting up buffers for digital output operations 175
  setting up channel lists 170
  setting up clocks 171
  setting up clocks for counter/timer operations 181
  setting up gates 181
  setting up pre-triggers 171
  setting up the channels 169
  setting up the subsystem 168
  setting up triggered scans 173
  setting up triggers 171
pausing an operation
  A/D 76
  D/A 76
digital input 76
digital output 76
post-trigger mode 76
pre-trigger mode 78
programmable gain 71
programming flowcharts 141
pseudo-differential channels 62
pulse output 106
  duty cycle 136
  flowcharts 162
  one-shot 123
  output types 136
  rate generation (continuous) 120
  repetitive one-shot 126
pulse train output 120
pulse types
  high-to-low 136
  low-to-high 136
pulse width 136

Q
quadrature decoder 138
quadrature decoder flowcharts 164
quadrature decoder operations 138
queues
  done 99
  inprocess 97
  ready 96

R
ranges 68
rate generation mode 120
ready queue 96
releasing the subsystem and driver 58
repetitive one-shot mode 126
requirements 15
resetting an operation
  A/D 75
  D/A 75
continuous edge-to-edge measurement 120
digital input 75
digital output 75
edge-to-edge measurement 116
event counting 108
rate generation 121
repetitive one-shot 126
simultaneous 140
up/down counting 110
resolution 60
retrigger modes
  internal retrigger 83
  retrigger extra 84
  scan-per-trigger 82

S
sample code
  for continuous A/D 193
  for single-value A/D 188
scan-per-trigger retrigger mode 82
service and support procedure 186
setting up buffers
  for A/D operations 174
  for D/A operations 175
  for digital input operations 174
  for digital output operations 175
setting up channel lists 170
setting up channel parameters 169
setting up clocks 171
setting up clocks for counter/timer operations 181
setting up gates 181
setting up pretriggers 171
setting up subsystem parameters 168
setting up triggered scans 173
setting up triggers 171
simultaneous operations 139
  flowcharts 166
single wrap mode 103
single-ended inputs 61
single-value operations 72
  for a simultaneous device 74
input flowcharts 143
output flowchart 146
  sample code for analog input 188
software architecture 53
software gate type 132
software trigger 88
specifying a single channel 62
specifying a subsystem 56
specifying AC or DC coupling 91
specifying buffers 95
specifying channels in the channel list 64
specifying clock sources 85
specifying DMA channels 104
specifying filters 71
specifying gains 69
specifying ranges 68
specifying single-value operations 72
specifying single-value operations for a simultaneous device 74
specifying synchronous digital I/O values in the channel list 66
specifying the data flow 72
specifying thermocouples 92
specifying trigger sources 88
specifying triggered scan mode 81
starting an operation
A/D 75
D/A 75
digital input 75
digital output 75
digital input 75
digital output 75
digital input 75
digital output 75
digital input 75
digital output 75
edge-to-edge measurement 116
event counting 108
frequency measurement 111
one-shot 124
rate generation 121
repetitive one-shot 126
up/down counting 110

T
technical support 186
terminating the session 58
termination functions 35
thermocouple inputs 92
thermocouples 92
returning CJC values in the data stream 95
type 93
threshold (negative) trigger 90
threshold (positive) trigger 89
timer event trigger 91
transferring data from an inprocess buffer 178
trigger source 88
analog event 90
digital event trigger 90
external analog threshold (negative) 90
external analog threshold (positive) 89
external digital (TTL) 89
extra trigger 91
software (internal) 88
timer event trigger 91
triggered scan mode 81
troubleshooting checklist 186
troubleshooting procedure 186
twos complement data format 59
type, channel 61

U
up/down counting operations 109
flowcharts 154
Index

W

wrap modes 102
  multiple 103
  none 103
  single 103