

1 Help Contents



CoMonitor™ Application Monitor Help Contents

[CoDeveloper Product Overview](#)

[Simulation Features](#)

[Menu Bar Options](#)

[Screen Displays](#)

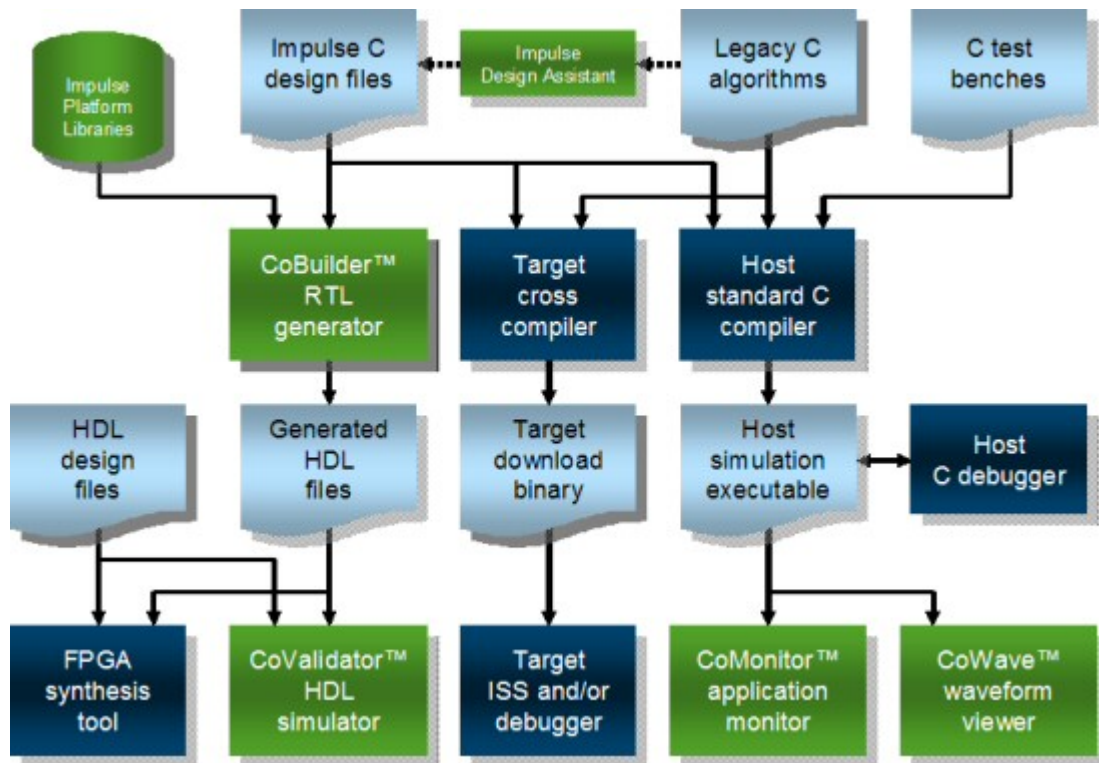
1.1 CoDeveloper Product Overview



Welcome to CoDeveloper

Welcome to CoDeveloper: advanced software technologies enabling high-performance applications on programmable hardware platforms. CoDeveloper will change the way you create applications for programmable hardware.

CoDeveloper is designed for use with leading tools for embedded software and programmable hardware development, including FPGA synthesis tools, desktop C/C++ development tools, and embedded compiler and debugger tools.



The complete CoDeveloper environment consists of a set of libraries allowing Impulse C applications to be executed in a standard desktop compiler (for simulation and debugging purposes) as well as cross-compiler and translation tools allowing Impulse C applications to be implemented on selected programmable hardware platforms. Additional tools for application profiling and co-simulation with other environments (including links to EDA tools for hardware simulation) are provided.

Introducing Impulse C

Impulse C is a library of functions and related datatypes that provide a programming environment, and a programming model, for highly parallel applications targeting coarse-grained programmable hardware platforms. Impulse C has been designed to simplify the expression, verification, and compilation of complex applications consisting of multiple (potentially hundreds) of distinct parallel processes. Impulse C has been optimized for mixed software/hardware targets, with the goal of abstracting details of inter-process communication and allowing relatively platform-independent application design.

CoDeveloper includes the Impulse C libraries and associated software tools that help you use standard C language for the design of highly parallel applications targeting programmable hardware platforms. CoDeveloper includes an Application Manager, [Application Monitor](#), integration with common development environments (including Microsoft Visual C++, Metrowerks CodeWarrior, and the GNU tools), platform-specific compilers (including Impulse C compilers targeting FPGAs) and other resources for Impulse C application developers.

The CoDeveloper Application Manager

The CoDeveloper Application Manager can be used to quickly generate new Impulse C applications, as well as to manage existing applications and their constituent files. You can use the Impulse C Application Manager to create entirely new projects (using the Impulse C Design Assistant to help create template Impulse C files) or to help manage existing Impulse C projects. Because Impulse C is compatible with standard third-party compilers, you can use CoDeveloper in combination with C

development and debugging tools to form a complete application development design environment.

For more information about CoDeveloper and Impulse C, refer to the main CoDeveloper Help file.

See Also

[Simulation Features](#)
[Application Monitor](#)

1.1.1 Contents



CoMonitor™ Application Monitor Help Contents

[CoDeveloper Product Overview](#)
[Simulation Features](#)
[Menu Bar Options](#)
[Screen Displays](#)

1.2 Simulation Features



Desktop Simulation Using Standard Programming Environments

The CoDeveloper Application Manager and [Application Monitor](#), when used in conjunction with a standard desktop compiler, form a complete system for the compilation, execution and monitoring of Impulse C design descriptions. The system includes an Impulse C build facility (based on Make), the Application Monitor, and Impulse C instrumentation functions useful for simulating and debugging your Impulse C applications.

Impulse C applications are compiled (using the standard third-party compiler of your choice) into a native Windows executable that may be run directly or executed from with a standard third-party debugger. When launched, Impulse C executable files interact with the Application Monitor to allow detailed investigation of your application.

Impulse C instrumentation functions allow processes, streams, and signals, as well as text log messages, to be monitored as your application runs in a debugger. These features give you the power to quickly track down problems in your Impulse C applications and to analyze performance issues so you can optimize your applications.

See Also

[Simulating Impulse C Applications](#)
[Application Monitor](#)

1.2.1 Simulating Impulse C Applications

Simulating Impulse C Applications

Simulation is an important element of the application development process. During simulation your goal will be to verify that your application and its component algorithms operate as expected, both in terms of algorithmic correctness and in terms of required performance.

Simulating for correct application behavior—behavioral simulation—is relatively easy once you have developed the necessary test stimulus and written a corresponding set of software tests for your application. Because Impulse C is compatible with standard C and C++ programming environments, you can use the full power of C programming to create complex, repeatable software test suites, and you can use standard C debugging tools to track down and fix errors in your application.

When simulating your application on your development machine, it is important to understand that the parallelism you express using Impulse C's stream, message, and process model is implemented by your desktop compiler using the threading model and associated libraries associated with your chosen environment. This means that the parallel behaviors you specify and the timing relationships between independently running processes may be very different from the behaviors and timing of your application when implemented on an actual programmable platform target.

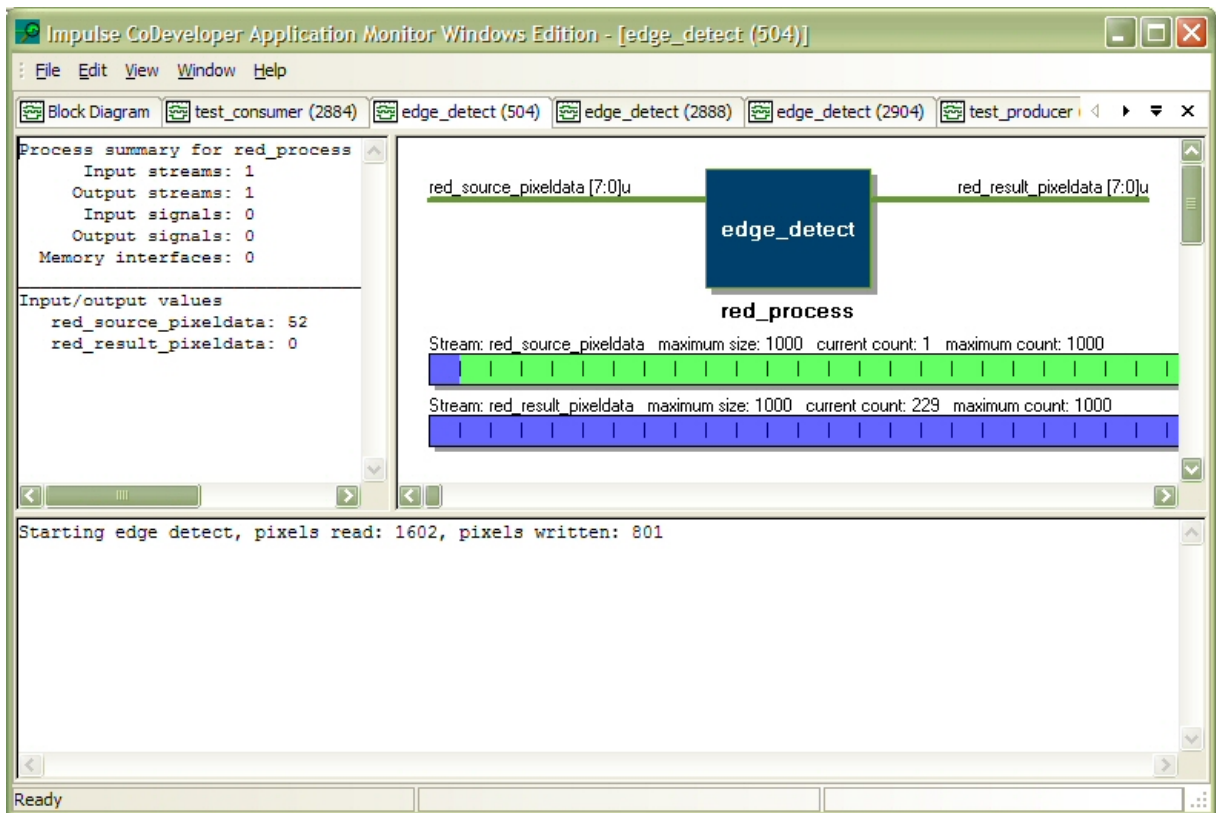
See Also

[Simulation Features](#)
[Simulation Executable](#)

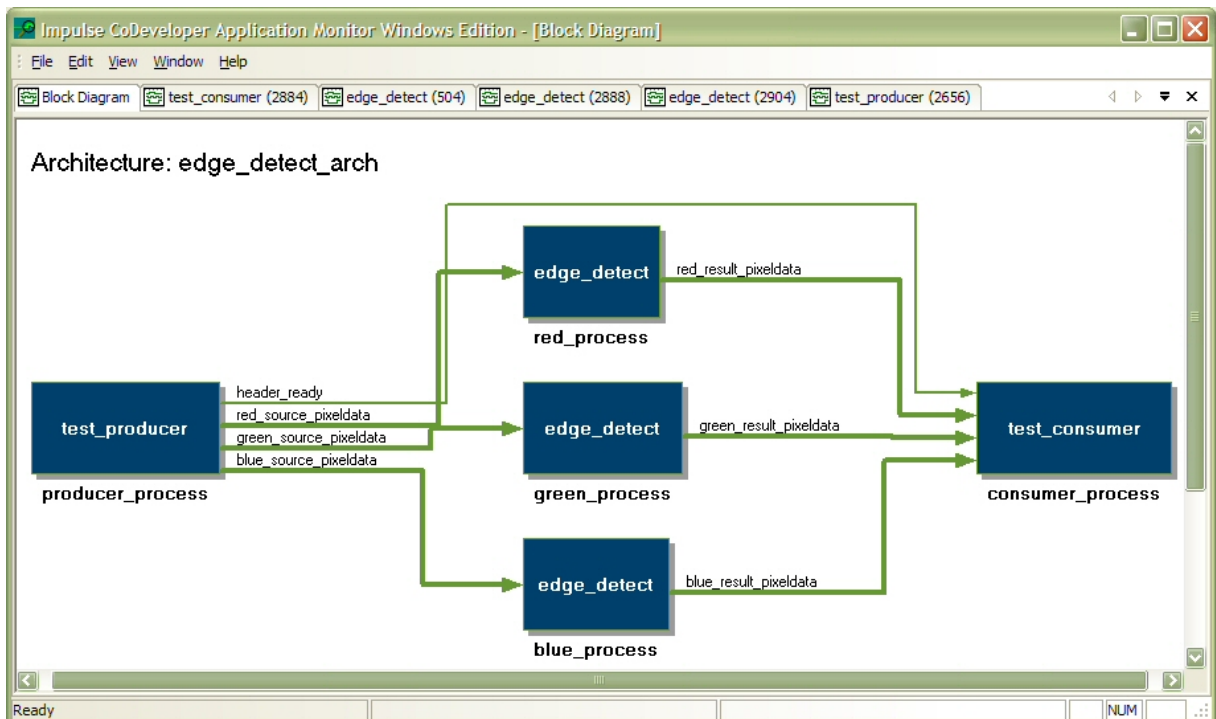
1.2.2 Application Monitor

Description

The Application Monitor allows you to observe your application as it executes, capturing messages, stream data values, and other information generated as a result of instrumenting your application. One or more Application Monitor log windows may be created by your application, with each window corresponding to one `cosim_logwindow_create` function that has been added to your application.



The Application Monitor also generates a block diagram of your application that can be helpful when analyzing process-to-process connection issues:



How You Use It

The following functions may be used in your application to format and display text messages in a log window:

cosim_logwindow_init() - Called from the configuration function of your application, this function sets up and initializes the Application Monitor communications.

cosim_logwindow_create(str) - This function creates a log window within the Application Monitor.

cosim_logwindow_write(logwindow, str) - This function writes an unformatted string value to the specified log window.

cosim_logwindow_fwrite(logwindow, formatstring, arg1, arg2...) - This function writes a formatted string value to the specified log window.

If your Impulse C application includes calls to these functions and you have invoked the Application Monitor prior to executing your simulation executable, then your executable application will communicate with the Application Monitor while it runs, displaying debugging messages and other information as appropriate.

See Also

[Simulation Executable](#)
[Screen Displays](#)

1.2.3 Simulation Executable

Description

A **Simulation Executable** is a native Windows executable file created as a result of compiling and linking your Impulse C application. Simulation executable files are intended for desktop simulation of your application, and may be run under the control of a standard C debugger. The [Application Monitor](#) may also be used as a debugging aid if appropriate instrumentation functions have been included in the application.

See Also

[Simulation Features](#)

1.3 Screen Displays



The CoMonitor Application Monitor Windows include:

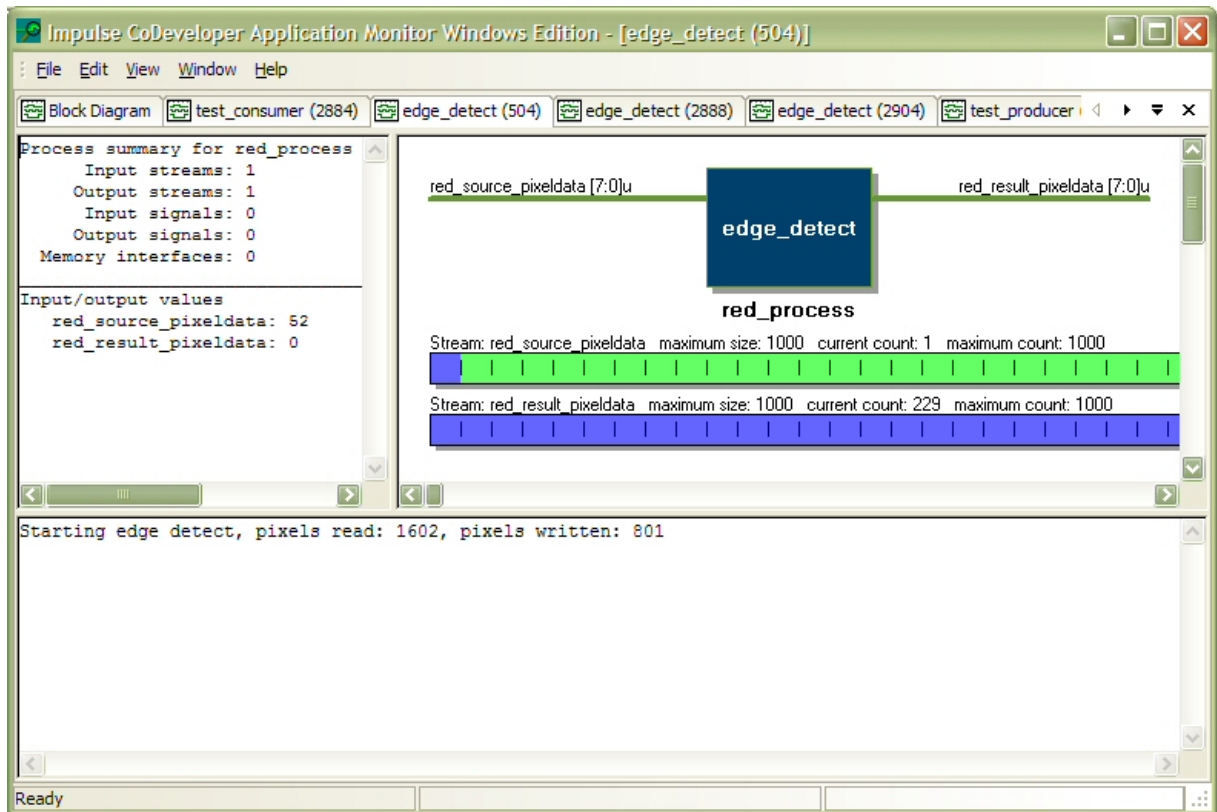
[Process Summary Window](#)
[Stream Window](#)
[Log Window](#)

1.3.1 Process Summary Window

Description

The Process Summary window displays information about the current process, including:

- The number of streams
- The number of signals
- The number of memory interfaces
- The current value for each stream and signal



The contents of the Process Summary Window are updated whenever a read or write event occurs on a stream or signal, or when a text message is sent to the Log Window.

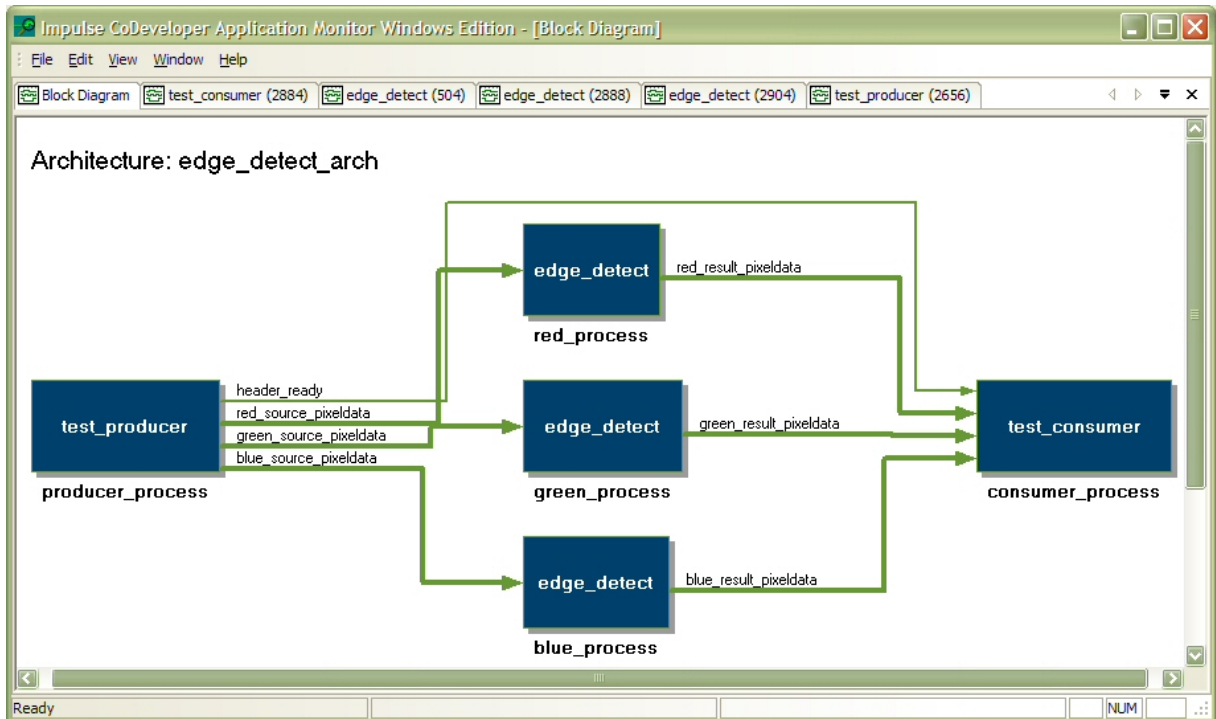
See Also

- [Stream Window](#)
- [Log Window](#)
- [Block Diagram Window](#)

1.3.2 Block Diagram Window

Description

The Block Diagram window displays information about the complete application, by generating a graphical representation of each Impulse C process and its connections to other processes:



By clicking on any block shown in the diagram, you can quickly jump to a specific [Process Summary window](#) for debugging and analysis purposes.

See Also

[Process Summary Window](#)
[Stream Window](#)
[Log Window](#)

1.3.3 Stream Window

Description

The Stream Window displays information about the current process and its associated streams, including:

- A diagram of the process and its interfaces
- Stream diagrams displaying, for each stream:
 - The size of the stream (depth of the stream buffer)
 - The current stream utilization (number of buffers containing data)
 - The maximum observed utilization

The contents of the Stream Window are updated whenever a read or write event occurs on a stream or signal.

See Also

[Process Summary Window](#)
[Log Window](#)

1.3.4 Log Window

Description

The Log Window displays text messages generated by calls to **cosim_logwindow_write** and **cosim_logwindow_fwrite**.

See Also

[Process Summary Window](#)
[Stream Window](#)

1.3.5 Menu Bar Options

Description

The CoMonitor main menu includes the following categories:

[Edit](#)
[View](#)
[Window](#)

1.3.5.1 Edit Menu

Description

The edit menu items perform standard Windows editing functions. A brief summary of each is given below.

Copy

This option copies a marked portion of text (block-marked using your mouse or your Shift and arrow keys) from a text editing window to the Clipboard. This action replaces the previous contents of the Clipboard. The shortcut key for this option is **Ctrl+C**.

Select All

This option block-marks the entire contents of the currently active text editing window. The **Copy** option can then be performed on the marked text.

See Also

[View Menu](#)
[Window Menu](#)

1.3.5.2 View Menu

Description

The View menu includes the following items:

Status Bar

This option toggles the display of the status bar at the bottom of the main window.

See Also

[Edit Menu](#)
[Window Menu](#)

1.3.5.3 Window Menu

Description

The Window menu items perform standard Windows functions. A brief summary of each is given below.

Cascade

This option cascades all open windows in the main application window.

Tile

This option tiles all open windows in the main application window.

See Also

[Edit Menu](#)
[View Menu](#)

1.4 Function Reference

Description

Impulse C includes a library of predefined functions useful for application monitoring. Simulation-related functions (prefixed with **cosim_**) allow you to instrument your application and gain quick access to internal application data during desktop simulation.

Predefined Functions

[cosim_logwindow_create](#)
[cosim_logwindow_fwrite](#)
[cosim_logwindow_init](#)
[cosim_logwindow_write](#)

1.4.1 cosim_logwindow_create

Function cosim_logwindow_create

```
cosim_logwindow cosim_logwindow_create(const char * name);
```

Header File

cosim_log.h

Description

This function creates an [Application Monitor](#) log window. This function may be called from any place in an application.

Arguments

The arguments to **cosim_logwindow_create** are as follows:

| | |
|------------|---|
| char *name | A programmer-defined name. This name may be any text string, and is used for visually identifying a specific log window when using the Application Monitor. |
|------------|---|

Return Value

A pointer to the created logwindow. This return value may subsequently be used as an argument to the functions [cosim_logwindow_write](#) and [cosim_logwindow_fwrite](#).

Notes

See Also

[cosim_logwindow_write](#)
[cosim_logwindow_fwrite](#)
[cosim_logwindow_init](#)
[Application Monitor](#)

1.4.2 cosim_logwindow_fwrite

Function cosim_logwindow_fwrite

```
int cosim_logwindow_fwrite(cosim_logwindow log, const char * format,  
...);
```

Header File

cosim_log.h

Description

This function writes formatted messages to an [Application Monitor](#) log window. This function may be called from any place in an application.

Arguments

The arguments to **cosim_logwindow_fwrite** are as follows:

| | |
|---------------------|--|
| cosim_logwindow log | A log window pointer as returned by a preceding call to function cosim_logwindow_create . |
| const char *format | A null-terminated character string representing the format string for the message to be written, followed by zero or more arguments. |

Return Value

Returns 1 if successful, otherwise 0.

Notes

The [cosim_logwindow_fwrite](#) function allows formatted messages to be displayed in the specified log window. Format strings and formatting characters follow the same rules as the **printf** function.

See Also

[cosim_logwindow_create](#)
[cosim_logwindow_write](#)
[cosim_logwindow_init](#)
[Application Monitor](#)

1.4.3 cosim_logwindow_init

Function [cosim_logwindow_init](#)

```
int cosim_logwindow_init();
```

Header File

cosim_log.h

Description

This function initializes the [Application Monitor](#) and must be called from an application's configuration subroutine.

Arguments

None.

Return Value

Returns 1 if successful, otherwise 0.

See Also

[cosim_logwindow_create](#)
[cosim_logwindow_write](#)
[cosim_logwindow_fwrite](#)
[Application Monitor](#)

1.4.4 `cosim_logwindow_write`

Function `cosim_logwindow_write`

```
int cosim_logwindow_write(cosim_logwindow log, const char * msg);
```

Header File

`cosim_log.h`

Description

This function writes messages to an [Application Monitor](#) log window. This function may be called from any place in an application.

Arguments

The arguments to `cosim_logwindow_write` are as follows:

| | |
|----------------------------------|---|
| <code>cosim_logwindow log</code> | A log window pointer as returned by a preceding call to function cosim_logwindow_create . |
| <code>const char *msg</code> | A null-terminated character string representing the message to be written. |

Return Value

Returns 1 if successful, otherwise 0.

See Also

[cosim_logwindow_create](#)
[cosim_logwindow_fwrite](#)
[cosim_logwindow_init](#)
[Application Monitor](#)