©Copyright InduSoft Systems 2008

Web-Ready InduSoft Tools for Automation

**Technical Note**
**Driver Runtime**

## Introduction

This document describes the technical aspect of the Driver Runtime task of InduSoft Web Studio (IWS).

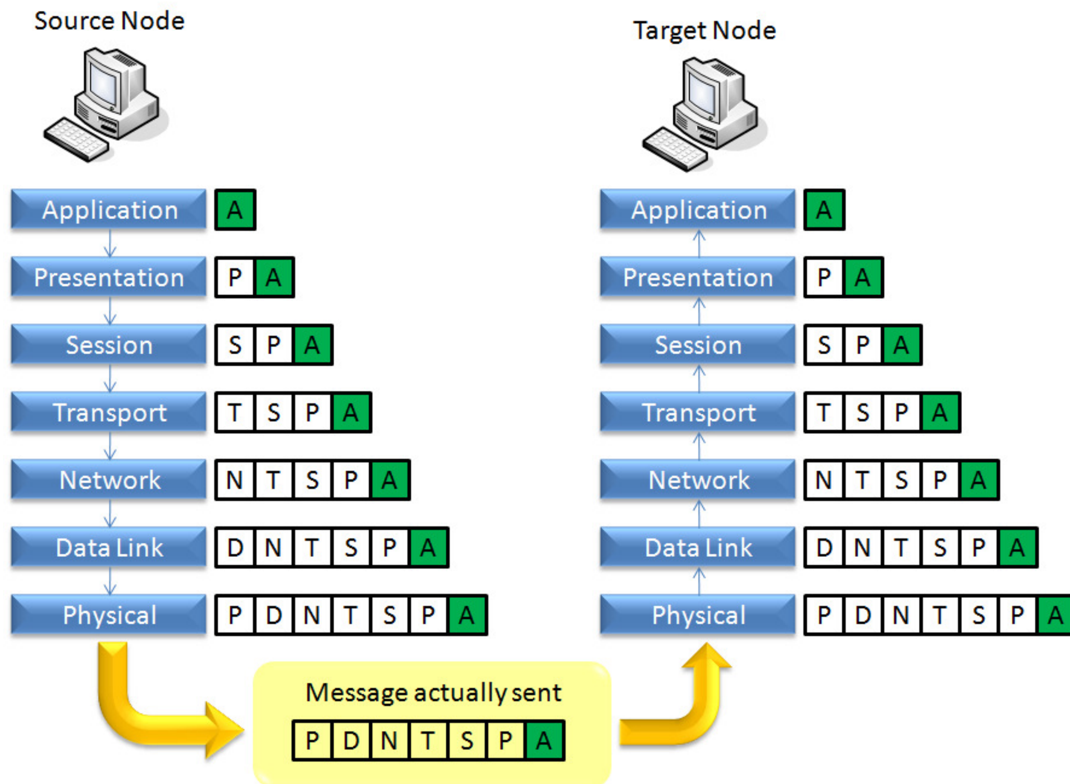
The Driver Runtime task is responsible for exchanging data with other systems, which could be another software (e.g.: PC Based Control Software) or an external device, such as a PLC, Robot, Remote I/O, Barcode reader, or any other device that supports a protocol implemented by the driver

## Basic Concepts

The following concepts are used on this document:

- **Protocol**: Set of rules that end points in a telecommunication connection use when they communicate.

- **Protocol Layers:** Typically, a protocol is composed of several layers. A layer is a collection of related functions that provides services to the layer above it and receives service from the layer below it. For instance, the OSI Model consists of seven layers: Application, Presentation, Session, Transport, Network, Data Link, and Physical. When one node (station) sends a message to another node (station), it encapsulates the message though all the layers of the protocol used. When the message arrives in the target node, it de-encapsulate it though the inverse order of layers, as illustrated in the following picture.



©Copyright InduSoft Systems 2008

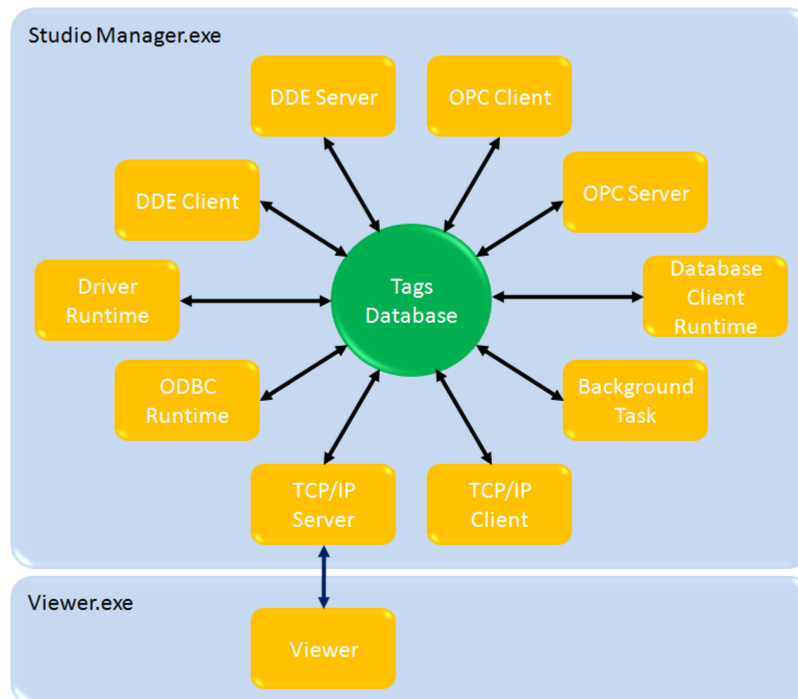
Each layer is responsible for executing its functions and for integrating properly with its adjacent layers. It is important to notice that there are standard protocols for some layers. For instance, TCP and UDP are Transport Layer protocols. Therefore, different protocols on the higher layers (Application, Presentation, Session) can share the standard transport layer when communicating data. For instance, both the MODBU driver, which implements the "Modbus over TCP" protocol as well as the ABTCP driver, which implements the "DF1 over TCP" protocol share the TCP layer for transport. The differences between the protocols implemented by these two drivers are only in the higher layers (Application, Presentation, and Session). The operating systems itself, along with its device drivers for the network adapters, implements the lower layers of these protocols. Adopting the concept of layers, a given protocol is defined by a set of Protocol Layers.

## InduSoft Web Studio internal structure

InduSoft Web Studio runtime has two main processes:

- **Studio Manager.exe**: Multi-thread process, which manages most of the runtime tasks.
- **Viewer.exe**: Single-thread process, which manages the runtime graphic interface (screens) and exchanges data with the Studio Manager via the TCP/IP Server task. The link between the Viewer and Studio Manager is implemented over TCP/IP. Therefore, the Viewer module can run either on the same station where Studio Manager is running or in a remote station (Secure Viewer Thin Client), providing a wide level of flexibility and expansibility for the graphic interface.

The following picture illustrates the internal architecture of InduSoft Web Studio:




The Tags Database is the main core of IWS runtime. It keeps the current value of all tags configured in the application. When any task changes a tag value, the task sends a message to the Tags Database updating the tag value. Then, the Tags Database sends a message to any other task(s) that are using the updated tag at this moment. Therefore, the Tags Database is the main gateway that keeps all tasks synchronized. Furthermore, the communication between the Tags Database and its tasks is by exception only – only when tags change of value. Using this method, there is no

©Copyright InduSoft Systems 2008

pooling among the tasks, optimizing the internal synchronizations and increasing the performance, based on a solid and robust platform.

For instance, when the Driver Runtime reads a new value from the device (e.g.: PLC), it sends a message to the Tags Database. The Tags Database then sends a message to all other tasks that are using that specific tag at this moment, for example, the Viewer. When the Viewer receives the value from the Tags Database (through the TCP/IP Server task), it updates the objects on the open screen that are using that tag.

Even though the Viewer is an independent process from Studio Manager.exe, it works just like another task (thread) of Studio Manager during the runtime. The only particularity of this module is the fact that it communicates with the Tags Database though the TCP/IP Server task, making the Viewer extremely modular and flexible for distributed architectures.

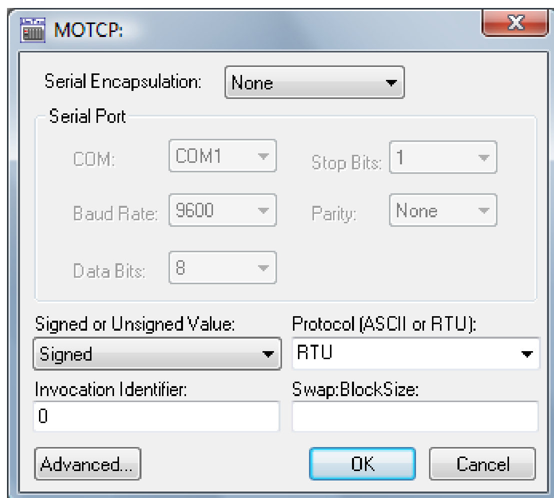
It is important to emphasize the fact that the execution of the tasks is asynchronous and the operating system executes them in a multi-tasking fashion. Even though the sequence of execution for the tasks is not preemptive, the multi-tasking nature of the operating system allied with mechanisms implemented by IWS guarantee the optimum performance during the runtime, once the execution of one task does not decrease significantly the performance of the others.

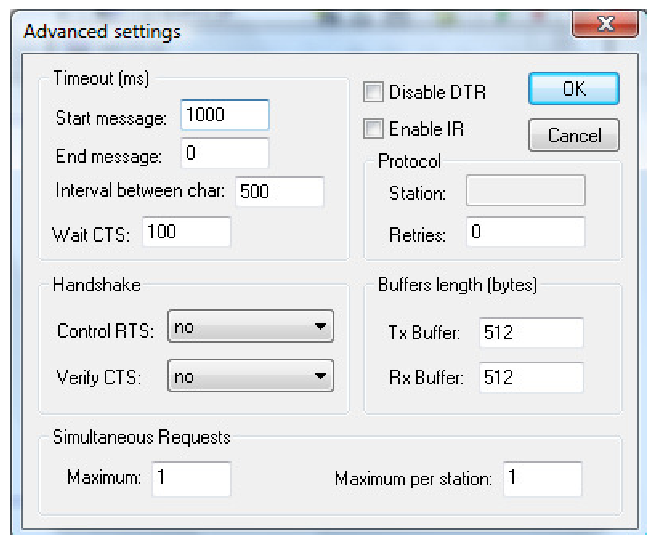
## Driver Runtime configuration

Each communication driver has its own syntax for station and register addressing. Some communication drivers have even protocol-specific parameters. However, all communication drivers supported by IWS share the same configuration interfaces.

IWS provides the following basic interfaces to configure communication drivers:

- **Driver Settings**: Allows the user to adjust the settings that will be used by all communication worksheets configured for the specific communication driver.



©Copyright InduSoft Systems 2008

The product documentation provides detailed description about the driver settings. However, the following fields deserve special attention in this document:

| Field | Description |
|---|---|
| Serial Encapsulation | This option is useful to encapsulate **serial** protocols through TCP/IP, UDP/IP or Modem. Obviously, this option applies only for serial protocols, where the user can use an external converter (e.g.: Ethernet to Serial) to de-encapsulate the protocol on the other end. This option must be set to **None** for protocols that are natively non-serial, such as Modbus over TCP/IP (MOTCP driver). |
| Timeout – Start Message | When sending messages to the device (e.g.: PLC), the driver waits for a response from it. If the response does not arrive within the timeout period of time set in this field, the message is discharged indicating time-out error, and the next message is executed. If the **Retries** field is set with a value greater than zero, the same message is sent again and the timeout time is reset. If the communication fails the number of retries set by the user, the message is discharged indicating error, and the next message is executed. |
| Simultaneous Requests | This option allows the user to configure the driver to support simultaneous connections. In other words, during the runtime, more than one instance (thread) of the driver is created in order to execute communication messages (commands) in parallel. This option is not applicable for protocols that do not support simultaneous requests in the same physical layer, such as serial protocols. This setting is described in details in the following sections of this document. |

**Note:** The Simultaneous Requests dialog is enabled by default only for drivers where this option has been fully tested. Therefore, this option is not enabled by default for some drivers that could support it (in theory). In order to enable this option, you can edit the <DriverName>.INI file for the driver from the \DRV sub-folder of InduSoft Web Studio, including the following settings in the [CommParam] section:
EnableSimultaneousRequests=1
SimultaneousRequestsMaximumMax=32
SimultaneousRequestsPerStationMax=16

- **Main Driver Sheet**: Single worksheet per communication driver. The main advantage of the Main Driver Sheet (MSD) is to provide a user-friendly interface to configure all communication addresses of the application. The user just need to associate one tag to each station/address and configure one of the following actions: **Read**, **Write**, or **Read+Write**. When the driver runs, the Main Driver sheet creates virtual groups for reading commands, not exceeding the maximum block size supported by the protocol. The reading commands are executed using the Read Block method (reads a block of addresses in each message), triggered whenever the System Tag BlinkSlow toggles (600ms by default). The writing commands are executed using the Write Item method (writes only one tag value to one address in each message), triggered when the respective tag(s) change of value. Some communication drivers do not support the MDS interface.

- **Standard Driver Sheet**: The user can configure several worksheets per communication driver. The main advantage of the Standard Driver Sheet (SDS) is to provide a flexible solution to divide the communication addresses in different groups manually and control how these groups will be managed dynamically, during the runtime. The addresses configured in each SDS cannot exceed the maximum block size supported by the protocol. Otherwise, it will fail. The user can control the actions of each SDS using the following commands:

| Command | Action | Method |
|---|---|---|
| **Write on Tag Change** | When any tag configured on the SDS changes of value, its value is written to the device. | WRITE ITEM |
| **Write Trigger** | When the tag configured in this field changes of value (e.g.: toggles), the values of all tags configured in the SDS are written to the device. | WRITE BLOCK |
| **Read Trigger** | When the tag configured in this field changes of value (e.g.: toggles), the values of all addresses configured in the SDS are read from the device. | READ BLOCK |
| **Enable Read When Idle** | The values of all addresses configured in the SDS are read from the device when the driver is idle. The driver is idle when it does not have messages from the other commands to be executed at a given moment. | READ BLOCK |

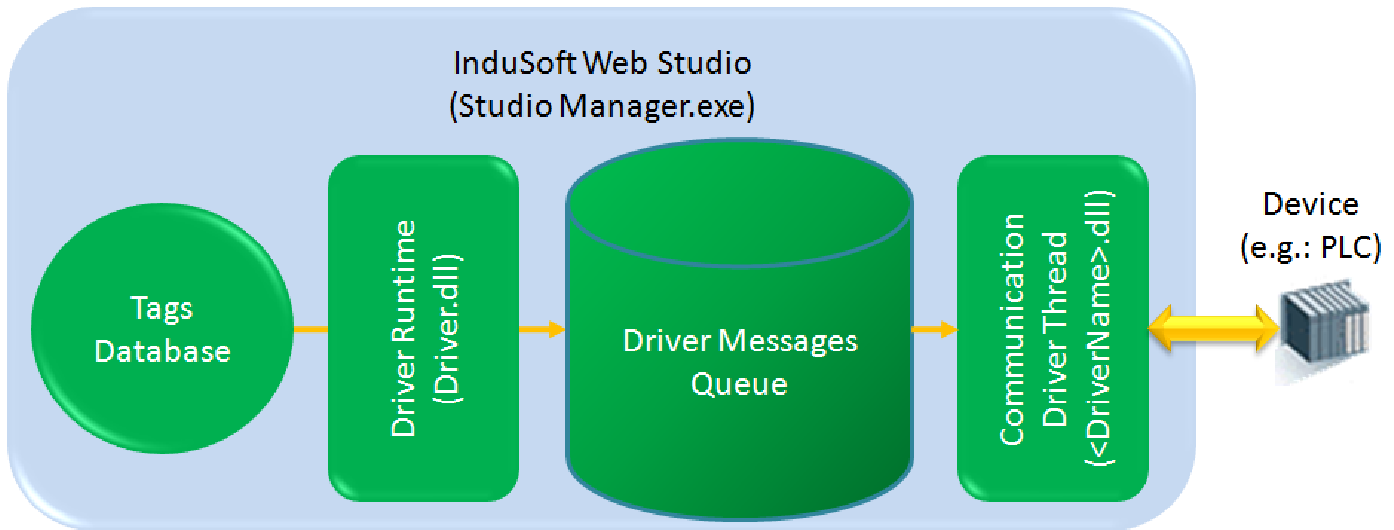
**Note**: Please consult the product documentation for details description about all settings available for communication drivers.

## Driver Runtime internal structure

The Driver Runtime is a common task shared by all communication drivers supported by IWS. Each communication driver is a modular component (<DriverName>.dll) which implements a specific protocol. For instance, the MOTCP communication driver implements the Modbus over TCP/IP protocol. The Driver Runtime task is able to handle more than one communication drivers simultaneously. The maximum number of communication drivers supported simultaneously by the Driver Runtime task is limited by the product license.

Each communication driver can create one (or more) instances during the runtime, as described in the following sections.

The simplest scenario for the communication driver interface is one instance of a communication driver exchanging data with one device, as illustrated in the following picture:




The Driver Runtime task can trigger messages (reading or writing commands) at a faster rate than the communication driver is able to execute. Therefore, there is a buffer between the Driver Runtime and the communication driver, called

©Copyright InduSoft Systems 2008

Driver Messages Queue. The Driver Runtime inserts messages in the queue when reading or writing commands are triggered. The communication driver removes the messages from the queue after executing them.

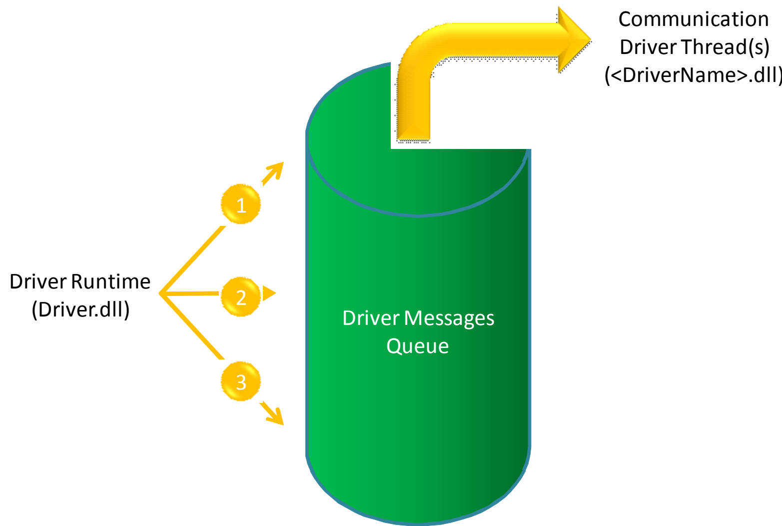
### Driver Messages Queue

The order of the messages included by the Driver Runtime in the Driver Messages Queue depends on two factors: **Priority** and **Timestamp**. Messages with higher priority are executed before messages with lower priority, regardless of their Timestamps (time when the messages were triggered). Messages with the same priority are executed according to their timestamp, in a FIFO (First In, First Out) manner.

Because the Driver Runtime inserts the messages in the queue in the order that they must be executed, based on the priority and timestamp of each message, the Communication Driver executes the message on the top of the queue, removing it from the queue after executing it, regardless if the communication was successful or not (e.g.: time-out).

**Note:** If the user specified a number of retires in the communication driver Settings→Advanced dialog, the communication driver keeps retrying to execute the message the number of times specified by the user in case the communication fails. When the number of retries expires, the message is removed from the queue, even if the communication was not successful.

The following picture illustrates how the Driver Runtime task inserts the messages in the queue according to their priority/timestamp and how the Communication Driver executes the message from the top of the queue and removes it after executing it.



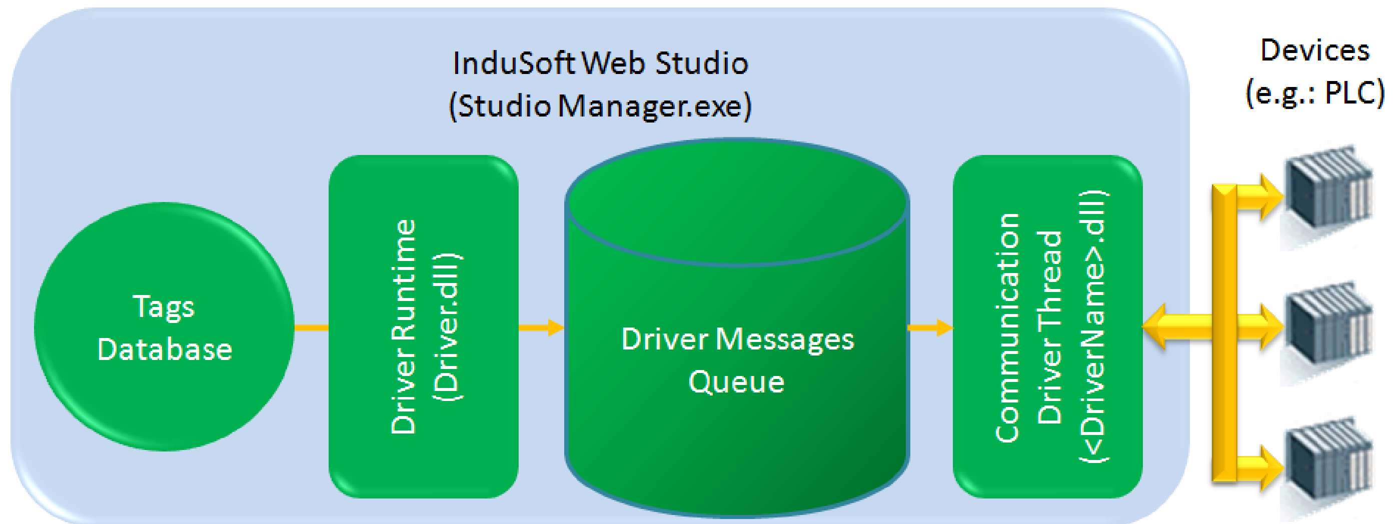

The following table describes the type of messages with different priorities, in descentent order of priority. The priority 1 is the highest priority (on the top of the queue) while the priority 4 is the lowest priority (on the bottom of the queue):

| Priority | Worksheet Type | Command |
|---|---|---|
| Priority 1 (highest priority) | SDS (Standard Driver Sheet) | Any command (Write on Tag Change, Write Trigger, Read Trigger, or Enable Read When Idle), as long as the check-box **Increase Priority** is checked. |
| Priority 2 | SDS (Standard Driver Sheet) | **Write on Tag Change** |
| | MDS (Main Driver Sheet) | **Write** |
| Priority 3 (lowest priority) | SDS (Standard Driver Sheet) | **Write Trigger** and **Read Trigger** |
| | MDS (Main Driver Sheet) | **Read** |
| | SDS (Standard Driver Sheet) | **Enable Read When Idle** |

©Copyright InduSoft Systems 2008

**Note:** Prior to InduSoft Web Studio v6.1+SP5 (Build 61.5.00.00), messages from the **Enable Read When Idle** commands had an even lower priority (priority 4). Then, these messages would not even be sent to the queue, unless there was at least one driver instance (thread) available (idle) to execute it. This behavior was not necessary wrong, but it led to confusion, so it was modified for newer versions.

The same communication driver can exchange data with several devices, as illustrated in the following picture:




Each instance of a communication driver executes the messages from the Driver Messages Queue **synchronously**. It means that the Communication Driver does not execute the next message on the queue until the current message is completely executed.

This behavior is suitable for many systems of small and medium size. However, in large systems, it might not provide the expected performance because the Communication Driver becomes a bottleneck when communicating with several devices, or even with one single device. When the Communication Driver is exchanging data with more than one device and at least one of them is not available (e.g.: disconnected from the network), it can decrease significantly the performance of the whole communication interface because the Communication Driver does not execute the messages for the device(s) properly connected to the network while it is executing the messages for the device(s) disconnected. Since the time-out time is typically in order of seconds, it can cause serious delays in the communication.

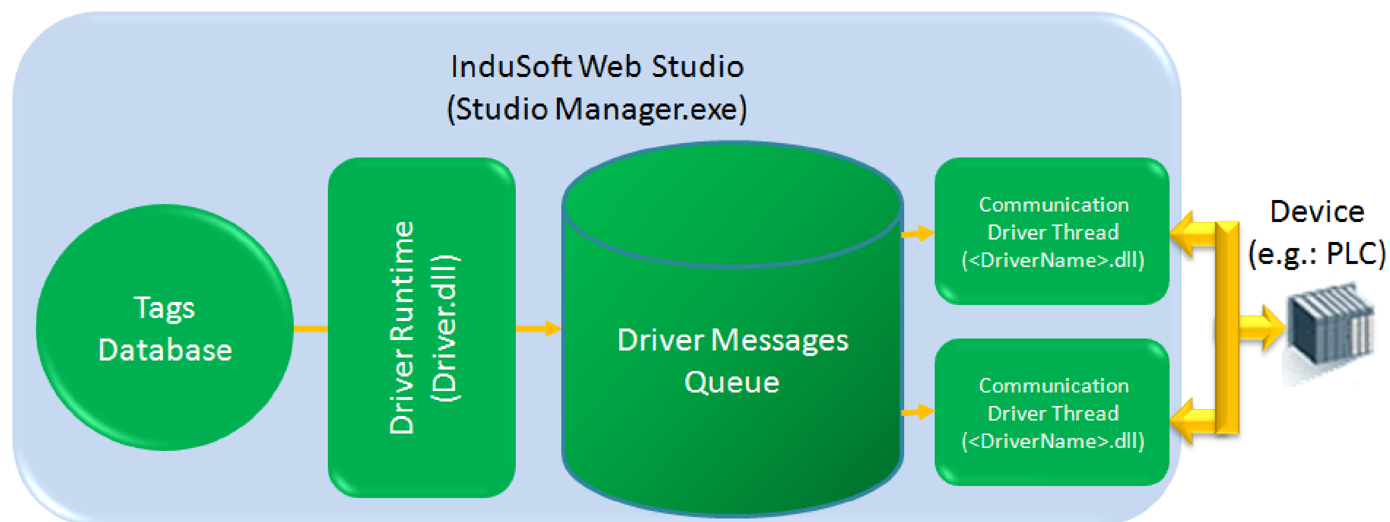
IWS provides a solution for these architectures using the simultaneous connections for the same communication driver.

## Simultaneous Connections

The user can configure the number a simultaneous connections for the same communication driver on the Settings→Advanced dialog interface of the driver. During the runtime, IWS creates an independent instance (thread) of the communication driver for each simultaneous connection configured by the user.

The following picture illustrates the internal structure of the communication interface of IWS when the user configures two simultaneous connections for the same device:

©Copyright InduSoft Systems 2008

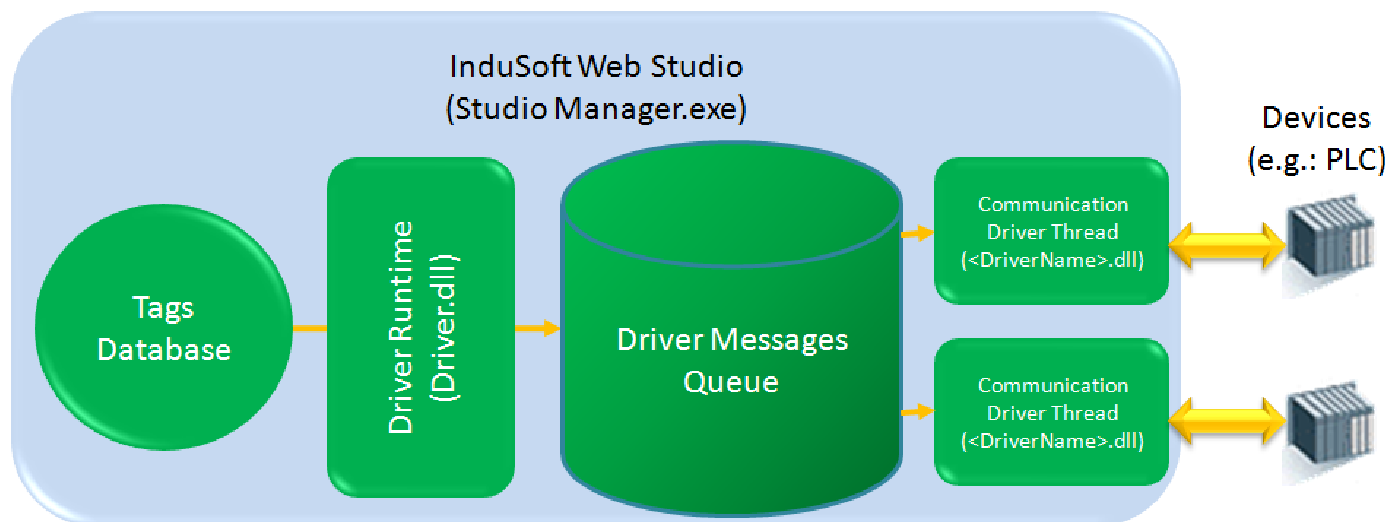



It is important to mention that the instances (threads) of the driver are created in memory, in a transparent way for the user.

The different instances (threads) of the Communication Driver work in parallel. When either instance executes one message, it removes it from the queue and looks for the next suitable message on the queue for it (from the top to the bottom of the queue, according to the priority and timestamp of the messages).

IWS implements mechanisms to make sure that different instances of the driver do not execute the same message simultaneously. Moreover, the user can configure the maximum number of simultaneous connections **per device** in the Settings→Advanced dialog of the driver. Therefore, when one instance (thread) of the communication driver looks for a message in the Driver Messages Queue, it might not take the topmost message, if it is addressed for a device which is already in communication with the maximum number of connections (other communication driver instances) configured by the user. In this case, the communication driver that is idle looks for the topmost message for a device that is not in communication with the maximum number of connections configured by the user – not necessarily the topmost message in the queue.

The following picture illustrates the internal architecture when the user configures two simultaneous connections and one per device. In this case, there will be one instance (thread) of the communication driver for each device. Therefore, if one device is not responding to the driver requests, it will not decrease the performance of the communication with the other device(s).




**Note**: Simultaneous connections cannot be implemented with serial protocols because the physical layer does not support asynchronous requests on the serial line (peer-to-peer). Moreover, drivers that use third-party libraries will support simultaneous connections only if the third-party library supports it too.

## Map of Revision

| Revision | Author | Date | Comments |
|---|---|---|---|
| A | Fabio Terezinho | January 11, 2008 | ▪ Initial Draft Revision |
| B | Fabio Terezinho | March 17, 2009 | ▪ Modified the priority for the command Enable Read When Idle, based on the changes implemented in the Build 61.5.0.0. |