# Honeywell HC900 Ethernet Driver

# Table of Contents

# Honeywell HC900 Ethernet Driver

Help version 1.031

## CONTENTS



## Overview

The Honeywell HC900 Ethernet Driver provides an easy and reliable way to connect Honeywell HC900 Ethernet controllers to OPC Client applications, including HMI, SCADA, Historian, MES, ERP and countless custom applications. It is intended for use with Honeywell HC900 Hybrid Controllers and similar devices.

# Setup

## Supported Devices

HC900 Hybrid Controller

## Communication Protocol

Modbus Ethernet using Winsock V1.1 or higher.

## Device ID (PLC Network Address)

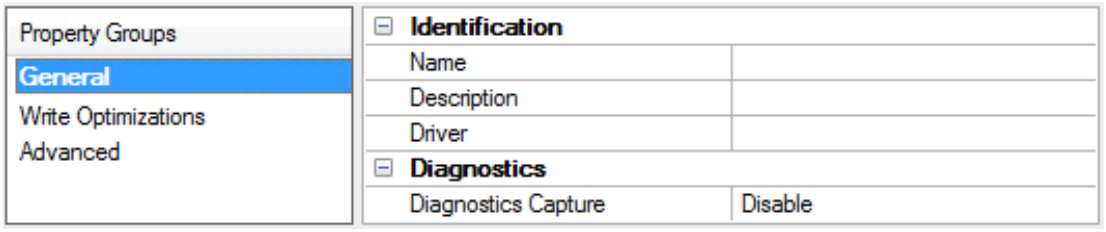
The Device ID is used to specify the device IP in standard YYY.YYY.YYY.YYY format.

## Maximum Number of Channels and Devices

The maximum number of supported channels is 16. The maximum number of supported devices is 8192.

# Channel Properties — General

This server supports the use of simultaneous multiple communications drivers. Each protocol or driver used in a server project is called a channel. A server project may consist of many channels with the same communications driver or with unique communications drivers. A channel acts as the basic building block of an OPC link. This group is used to specify general channel properties, such as the identification attributes and operating mode.




## Identification

**Name**: User-defined identity of this channel. In each server project, each channel name must be unique. Although names can be up to 256 characters, some client applications have a limited display window when browsing the OPC server's tag space. The channel name is part of the OPC browser information.
*For information on reserved characters, refer to "How To... Properly Name a Channel, Device, Tag, and Tag Group" in the server help.*

**Description**: User-defined information about this channel.
Many of these properties, including Description, have an associated system tag.

**Driver**: Selected protocol / driver for this channel. This property specifies the device driver that was selected during channel creation. It is a disabled setting in the channel properties.
**Note**: With the server's online full-time operation, these properties can be changed at any time. This includes changing the channel name to prevent clients from registering data with the server. If a client has already acquired an item from the server before the channel name is changed, the items are unaffected. If, after the channel name has been changed, the client application releases the item and attempts to re-acquire using the old channel name, the item is not accepted. With this in mind, changes to the properties should not be made once a large client application has been developed. Utilize the User Manager to prevent operators from changing properties and restrict access rights to server features.

### Diagnostics

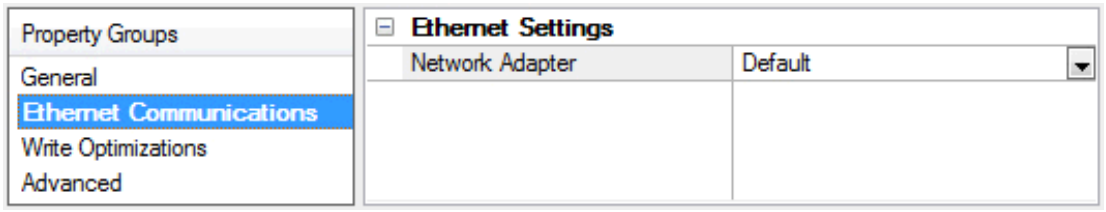
**Diagnostics Capture**: When enabled, this option makes the channel's diagnostic information available to OPC applications. Because the server's diagnostic features require a minimal amount of overhead processing, it is recommended that they be utilized when needed and disabled when not. The default is disabled.

**Note:** This property is disabled if the driver does not support diagnostics.

*For more information, refer to "Communication Diagnostics" in the server help.*

## Channel Properties — Ethernet Communications

Ethernet Communication can be used to communicate with devices.

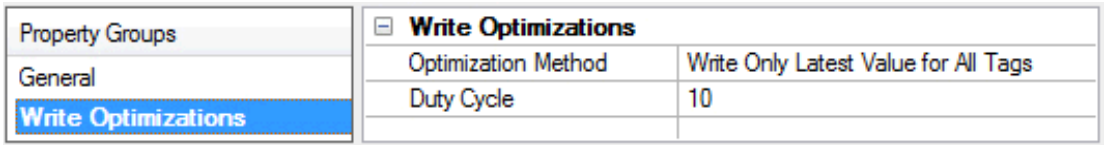



### Ethernet Settings

**Network Adapter**: Specify the network adapter to bind. When Default is selected, the operating system selects the default adapter.

## Channel Properties — Write Optimizations

As with any OPC server, writing data to the device may be the application's most important aspect. The server intends to ensure that the data written from the client application gets to the device on time. Given this goal, the server provides optimization properties that can be used to meet specific needs or improve application responsiveness.




### Write Optimizations

**Optimization Method**: controls how write data is passed to the underlying communications driver. The options are:

- **Write All Values for All Tags**: This option forces the server to attempt to write every value to the controller. In this mode, the server continues to gather write requests and add them to the server's internal write queue. The server processes the write queue and attempts to empty it by writing data to the device as quickly as possible. This mode ensures that everything written from the client applications is sent to the target device. This mode should be selected if the write operation order or the write item's content must uniquely be seen at the target device.
- **Write Only Latest Value for Non-Boolean Tags**: Many consecutive writes to the same value can accumulate in the write queue due to the time required to actually send the data to the device. If the server updates a write value that has already been placed in the write queue, far fewer writes are

needed to reach the same final output value. In this way, no extra writes accumulate in the server's queue. When the user stops moving the slide switch, the value in the device is at the correct value at virtually the same time. As the mode states, any value that is not a Boolean value is updated in the server's internal write queue and sent to the device at the next possible opportunity. This can greatly improve the application performance.

**Note**: This option does not attempt to optimize writes to Boolean values. It allows users to optimize the operation of HMI data without causing problems with Boolean operations, such as a momentary push button.

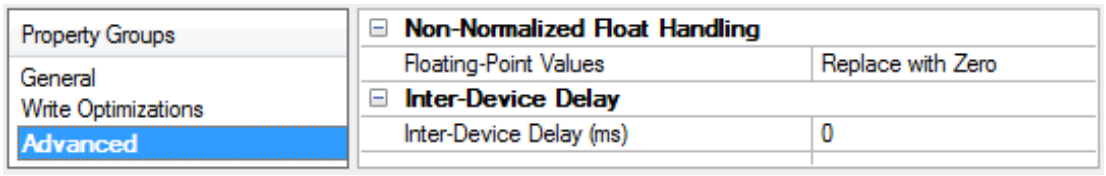
- **Write Only Latest Value for All Tags**: This option takes the theory behind the second optimization mode and applies it to all tags. It is especially useful if the application only needs to send the latest value to the device. This mode optimizes all writes by updating the tags currently in the write queue before they are sent. This is the default mode.

**Duty Cycle**: is used to control the ratio of write to read operations. The ratio is always based on one read for every one to ten writes. The duty cycle is set to ten by default, meaning that ten writes occur for each read operation. Although the application is performing a large number of continuous writes, it must be ensured that read data is still given time to process. A setting of one results in one read operation for every write operation. If there are no write operations to perform, reads are processed continuously. This allows optimization for applications with continuous writes versus a more balanced back and forth data flow.

**Note**: It is recommended that the application be characterized for compatibility with the write optimization enhancements before being used in a production environment.

## Channel Properties — Advanced

This group is used to specify advanced channel properties. Not all drivers support all properties; so the Advanced group does not appear for those devices.




**Non-Normalized Float Handling**: A non-normalized value is defined as Infinity, Not-a-Number (NaN), or as a Denormalized Number. The default is Replace with Zero. Drivers that have native float handling may default to Unmodified. Non-normalized float handling allows users to specify how a driver handles non-normalized IEEE-754 floating point data. Descriptions of the options are as follows:

- **Replace with Zero**: This option allows a driver to replace non-normalized IEEE-754 floating point values with zero before being transferred to clients.
- **Unmodified**: This option allows a driver to transfer IEEE-754 denormalized, normalized, non-number, and infinity values to clients without any conversion or changes.

**Note:** This property is disabled if the driver does not support floating point values or if it only supports the option that is displayed. According to the channel's float normalization setting, only real-time driver tags (such as values and arrays) are subject to float normalization. For example, EFM data is not affected by this setting.

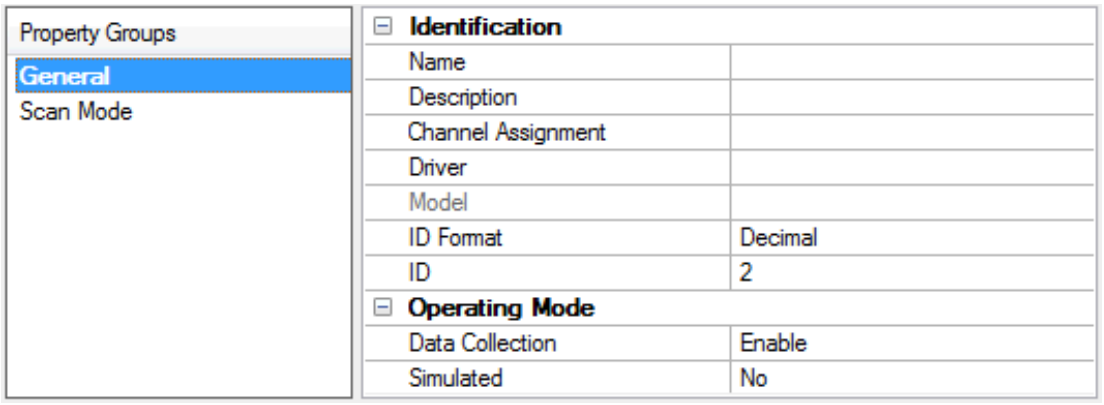
*For more information on the floating point values, refer to "How To ... Work with Non-Normalized Floating Point Values" in the server help.*

**Inter-Device Delay**: Specify the amount of time the communications channel waits to send new requests to the next device after data is received from the current device on the same channel. Zero (0) disables the delay.

**Note:** This property is not available for all drivers, models, and dependent settings.

## Device Properties — General

A device represents a single target on a communications channel. If the driver supports multiple controllers, users must enter a device ID for each controller.




### Identification

**Name**: This property specifies the name of the device. It is a logical user-defined name that can be up to 256 characters long, and may be used on multiple channels.
**Note**: Although descriptive names are generally a good idea, some OPC client applications may have a limited display window when browsing the OPC server's tag space. The device name and channel name become part of the browse tree information as well. Within an OPC client, the combination of channel name and device name would appear as "ChannelName.DeviceName".
*For more information, refer to "How To... Properly Name a Channel, Device, Tag, and Tag Group" in server help.*

**Description**: User-defined information about this device.
Many of these properties, including Description, have an associated system tag.

**Channel Assignment**: User-defined name of the channel to which this device currently belongs.

**Driver**: Selected protocol driver for this device. This property specifies the driver selected during channel creation. It is disabled in the channel properties.

**Model**: This property specifies the specific type of device that is associated with this ID. The contents of the drop-down menu depends on the type of communications driver being used. Models that are not supported by a driver are disabled. If the communications driver supports multiple device models, the model selection can only be changed when there are no client applications connected to the device.

**Note:** If the communication driver supports multiple models, users should try to match the model selection to the physical device. If the device is not represented in the drop-down menu, select a model that conforms closest to the target device. Some drivers support a model selection called "Open," which allows users to communicate without knowing the specific details of the target device. For more information, refer to the driver help documentation.

**ID**: This property specifies the device's station / node / identity / address. The type of ID entered depends on the communications driver being used. For many drivers, the ID is a numeric value. Drivers that support a Numeric ID provide users with the option to enter a numeric value whose format can be changed to suit the needs of the application or the characteristics of the selected communications driver. The ID format can be Decimal, Octal, and Hexadecimal. If the driver is Ethernet-based or supports an unconventional station or node name, the device's TCP/IP address may be used as the device ID. TCP/IP addresses consist of four values that are separated by periods, with each value in the range of 0 to 255. Some device IDs are string based. There may be additional properties to configure within the ID field, depending on the driver.

### Operating Mode

**Data Collection**: This property controls the device's active state. Although device communications are enabled by default, this property can be used to disable a physical device. Communications are not attempted when a device is disabled. From a client standpoint, the data is marked as invalid and write operations are not accepted. This property can be changed at any time through this property or the device system tags.

**Simulated**: This option places the device into Simulation Mode. In this mode, the driver does not attempt to communicate with the physical device, but the server continues to return valid OPC data. Simulated stops physical communications with the device, but allows OPC data to be returned to the OPC client as valid data. While in Simulation Mode, the server treats all device data as reflective: whatever is written to the simulated device is read back and each OPC item is treated individually. The item's memory map is based on the group Update Rate. The data is not saved if the server removes the item (such as when the server is reinitialized). The default is No.

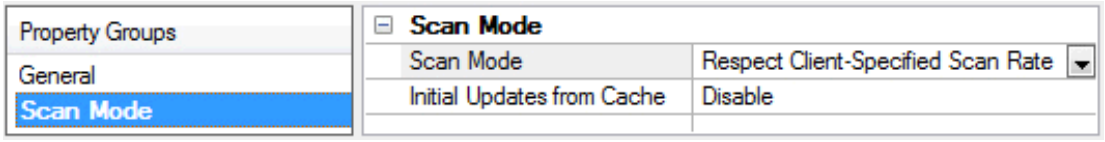
**Notes**:

1. This System tag (_Simulated) is read only and cannot be written to for runtime protection. The System tag allows this property to be monitored from the client.
2. In Simulation mode, the item's memory map is based on client update rate(s) (Group Update Rate for OPC clients or Scan Rate for native and DDE interfaces). This means that two clients that reference the same item with different update rates return different data.

Simulation Mode is for test and simulation purposes only. It should never be used in a production environment.

## Device Properties — Scan Mode

The Scan Mode specifies the subscribed-client requested scan rate for tags that require device communications. Synchronous and asynchronous device reads and writes are processed as soon as possible; unaffected by the Scan Mode properties.




**Scan Mode**: specifies how tags in the device are scanned for updates sent to subscribed clients. Descriptions of the options are:

- **Respect Client-Specified Scan Rate**: This mode uses the scan rate requested by the client.
- **Request Data No Faster than Scan Rate**: This mode specifies the maximum scan rate to be used. The valid range is 10 to 99999990 milliseconds. The default is 1000 milliseconds.
  **Note**: When the server has an active client and items for the device and the scan rate value is increased, the changes take effect immediately. When the scan rate value is decreased, the changes do not take effect until all client applications have been disconnected.
- **Request All Data at Scan Rate**: This mode forces tags to be scanned at the specified rate for subscribed clients. The valid range is 10 to 99999990 milliseconds. The default is 1000 milliseconds.
- **Do Not Scan, Demand Poll Only**: This mode does not periodically poll tags that belong to the device nor perform a read to get an item's initial value once it becomes active. It is the client's responsibility to poll for updates, either by writing to the _DemandPoll tag or by issuing explicit device reads for individual items. *For more information, refer to "Device Demand Poll" in server help*.
- **Respect Tag-Specified Scan Rate**: This mode forces static tags to be scanned at the rate specified in their static configuration tag properties. Dynamic tags are scanned at the client-specified scan rate.

**Initial Updates from Cache**: When enabled, this option allows the server to provide the first updates for newly activated tag references from stored (cached) data. Cache updates can only be provided when the new item reference shares the same address, scan rate, data type, client access, and scaling properties. A device read is used for the initial update for the first client reference only. The default is disabled; any time a client activates a tag reference the server attempts to read the initial value from the device.

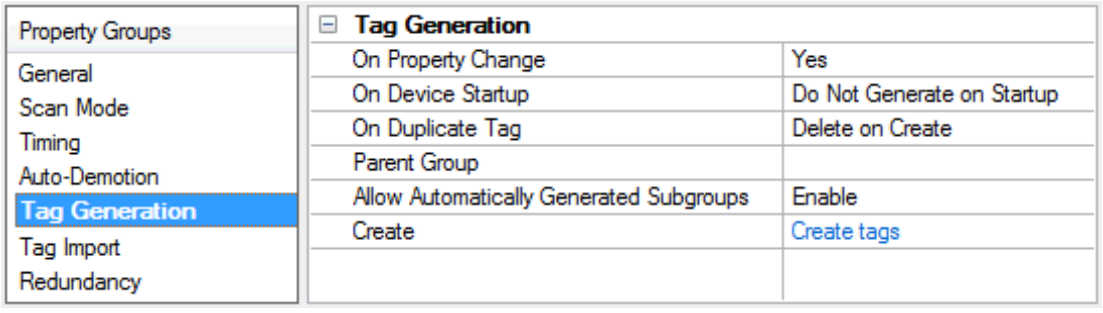
## Device Properties — Tag Generation

The automatic tag database generation features make setting up an application a plug-and-play operation. Select communications drivers can be configured to automatically build a list of tags that correspond to device-specific data. These automatically generated tags (which depend on the nature of the supporting driver) can be browsed from the clients.

If the target device supports its own local tag database, the driver reads the device's tag information and uses the data to generate tags within the server. If the device does not natively support named tags, the driver creates a list of tags based on driver-specific information. An example of these two conditions is as follows:

1. If a data acquisition system supports its own local tag database, the communications driver uses the tag names found in the device to build the server's tags.
2. If an Ethernet I/O system supports detection of its own available I/O module types, the communications driver automatically generates tags in the server that are based on the types of I/O modules plugged into the Ethernet I/O rack.

**Note**: Automatic tag database generation's mode of operation is completely configurable. For more information, refer to the property descriptions below.



**On Property Change**: If the device supports automatic tag generation when certain properties change, the **On Property Change** option is shown. It is set to **Yes** by default, but it can be set to **No** to control over when tag generation is performed. In this case, the **Create tags** action must be manually invoked to perform tag generation.

**On Device Startup**: This property specifies when OPC tags are automatically generated. Descriptions of the options are as follows:

- **Do Not Generate on Startup**: This option prevents the driver from adding any OPC tags to the tag space of the server. This is the default setting.
- **Always Generate on Startup**: This option causes the driver to evaluate the device for tag information. It also adds tags to the tag space of the server every time the server is launched.
- **Generate on First Startup**: This option causes the driver to evaluate the target device for tag information the first time the project is run. It also adds any OPC tags to the server tag space as needed.

**Note**: When the option to automatically generate OPC tags is selected, any tags that are added to the server's tag space must be saved with the project. Users can configure the project to automatically save from the **Tools** | **Options** menu.

**On Duplicate Tag**: When automatic tag database generation is enabled, the server needs to know what to do with the tags that it may have previously added or with tags that have been added or modified after the communications driver since their original creation. This setting controls how the server handles OPC tags that were automatically generated and currently exist in the project. It also prevents automatically generated tags from accumulating in the server.

For example, if a user changes the I/O modules in the rack with the server configured to **Always Generate on Startup**, new tags would be added to the server every time the communications driver detected a new I/O module. If the old tags were not removed, many unused tags could accumulate in the server's tag space. The options are:

- **Delete on Create**: This option deletes any tags that were previously added to the tag space before any new tags are added. This is the default setting.
- **Overwrite as Necessary**: This option instructs the server to only remove the tags that the communications driver is replacing with new tags. Any tags that are not being overwritten remain in the server's tag space.
- **Do not Overwrite**: This option prevents the server from removing any tags that were previously generated or already existed in the server. The communications driver can only add tags that are completely new.
- **Do not Overwrite, Log Error**: This option has the same effect as the prior option, and also posts an error message to the server's Event Log when a tag overwrite would have occurred.

**Note:** Removing OPC tags affects tags that have been automatically generated by the communications driver as well as any tags that have been added using names that match generated tags. Users should avoid adding tags to the server using names that may match tags that are automatically generated by the driver.

**Parent Group**: This property keeps automatically generated tags from mixing with tags that have been entered manually by specifying a group to be used for automatically generated tags. The name of the group can be up to 256 characters. This parent group provides a root branch to which all automatically generated tags are added.

**Allow Automatically Generated Subgroups**: This property controls whether the server automatically creates subgroups for the automatically generated tags. This is the default setting. If disabled, the server generates the device's tags in a flat list without any grouping. In the server project, the resulting tags are named with the address value. For example, the tag names are not retained during the generation process.

**Note**: If, as the server is generating tags, a tag is assigned the same name as an existing tag, the system automatically increments to the next highest number so that the tag name is not duplicated. For example, if the generation process creates a tag named "AI22" that already exists, it creates the tag as "AI23" instead.

**Create**: Initiates the creation of automatically generated OPC tags. If the device's configuration has been modified, **Create tags** forces the driver to reevaluate the device for possible tag changes. Its ability to be accessed from the System tags allows a client application to initiate tag database creation.

**Note**: **Create tags** is disabled if the Configuration edits a project offline.

## Automatic Tag Database Generation

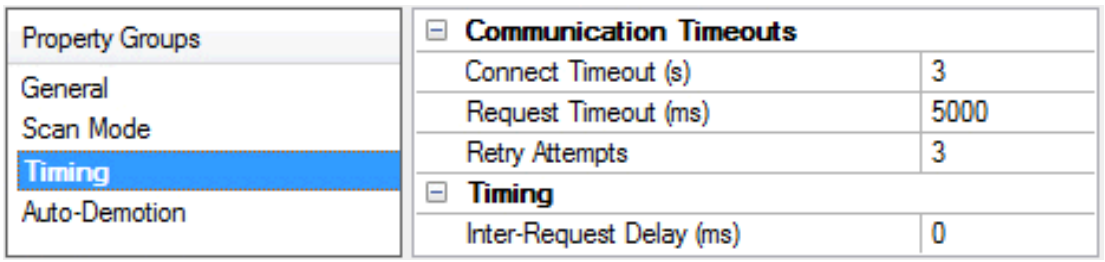
The Honeywell HC900 Ethernet Driver can create tags for the application automatically, either individually or in combination. The preferred method is Tag Import, wherein the driver reads controller configuration data from one or more CSV files exported by the Honeywell Hybrid Control Designer application (version 2.1 or later). Users receive the exact set of tags that are being used by the controller. Alternatively, the driver's Automatic Tag Database Generation feature offers the Tag Generation method, wherein the driver is given a general description of the controller's configuration (such as the number of loops and set point programmers used) and then generates a set of tags based on that description. Users may add or subtract tags manually after the tag generation operation is completed.

The driver may be configured to use Automatic Tag Database Generation either through the Add Device Wizard or an existing device object's Properties Editor. Users can access tag import and tag generation properties in the Add Device Wizard and the Add Device Wizard Summary Tag Import and Tag Generation property groups. After device creation, the properties can be edited from the device Property Editor's Tag Import and Tag Generation property groups. The driver creates the tags upon completion of the Device Wizard or once "OK" or "Apply" is pressed in the Property Editor. The driver will place messages in the server's event log as the tag generation process proceeds.

**See Also:** Tag Import and Tag Generation.

## Device Properties — Timing

The device Timing properties allow the driver's response to error conditions to be tailored to fit the application's needs. In many cases, the environment requires changes to these properties for optimum performance. Factors such as electrically generated noise, modem delays, and poor physical connections can influence how many errors or timeouts a communications driver encounters. Timing properties are specific to each configured device.



### Communications Timeouts

**Connect Timeout**: This property (which is used primarily by Ethernet based drivers) controls the amount of time required to establish a socket connection to a remote device. The device's connection time often takes longer than normal communications requests to that same device. The valid range is 1 to 30 seconds. The default is typically 3 seconds, but can vary depending on the driver's specific nature. If this setting is not supported by the driver, it is disabled.

**Note**: Due to the nature of UDP connections, the connection timeout setting is not applicable when communicating via UDP.

**Request Timeout**: This property specifies an interval used by all drivers to determine how long the driver waits for a response from the target device to complete. The valid range is 50 to 9,999,999 milliseconds (167.6667 minutes). The default is usually 1000 milliseconds, but can vary depending on the driver. The default timeout for most serial drivers is based on a baud rate of 9600 baud or better. When using a driver at lower baud rates, increase the timeout to compensate for the increased time required to acquire data.

**Attempts Before Timeout**: This property specifies how many times the driver issues a communications request before considering the request to have failed and the device to be in error. The valid range is 1 to 10. The default is typically 3, but can vary depending on the driver's specific nature. The number of attempts configured for an application depends largely on the communications environment. This property applies to both connection attempts and request attempts.

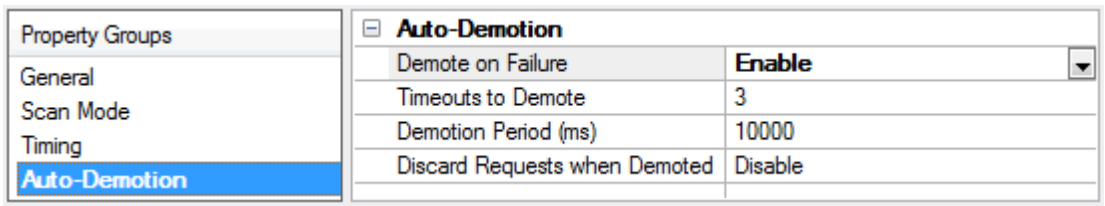
### Timing

**Inter-Request Delay**: This property specifies how long the driver waits before sending the next request to the target device. It overrides the normal polling frequency of tags associated with the device, as well as one-time reads and writes. This delay can be useful when dealing with devices with slow turnaround times and in cases where network load is a concern. Configuring a delay for a device affects communications with all other devices on the channel. It is recommended that users separate any device that requires an inter-request delay to a separate channel if possible. Other communications properties (such as communication serialization) can extend this delay. The valid range is 0 to 300,000 milliseconds; however, some drivers may limit the maximum value due to a function of their particular design. The default is 0, which indicates no delay between requests with the target device.

**Note**: Not all drivers support Inter-Request Delay. This setting does not appear if it is not available.

## Device Properties — Auto-Demotion

The Auto-Demotion properties can temporarily place a device off-scan in the event that a device is not responding. By placing a non-responsive device offline for a specific time period, the driver can continue to optimize its communications with other devices on the same channel. After the time period has been reached, the driver re-attempts to communicate with the non-responsive device. If the device is responsive, the device is placed on-scan; otherwise, it restarts its off-scan time period.



**Demote on Failure**: When enabled, the device is automatically taken off-scan until it is responding again.
**Tip**: Determine when a device is off-scan by monitoring its demoted state using the _AutoDemoted system tag.

**Timeouts to Demote**: Specify how many successive cycles of request timeouts and retries occur before the device is placed off-scan. The valid range is 1 to 30 successive failures. The default is 3.

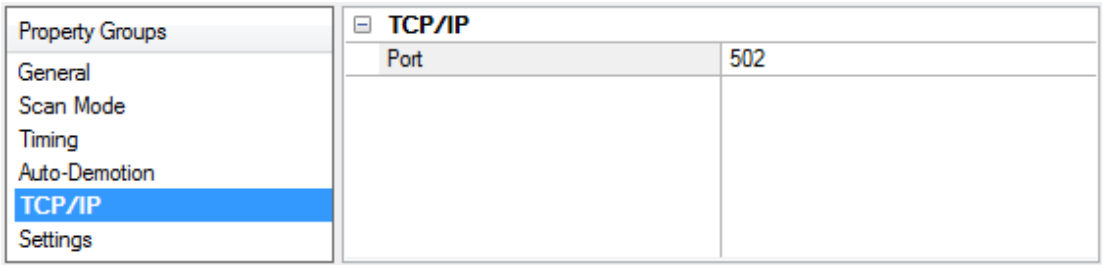
**Demotion Period**: Indicate how long the device should be placed off-scan when the timeouts value is reached. During this period, no read requests are sent to the device and all data associated with the read requests are set to bad quality. When this period expires, the driver places the device on-scan and allows for another attempt at communications. The valid range is 100 to 3600000 milliseconds. The default is 10000 milliseconds.

**Discard Requests when Demoted**: Select whether or not write requests should be attempted during the off-scan period. Disable to always send write requests regardless of the demotion period. Enable to discard writes; the server automatically fails any write request received from a client and does not post a message to the Event Log.

## Device Properties — TCP/IP

In order to use this driver, TCP/IP must be properly installed. For information on TCP/IP setup, refer to the Windows documentation.

### Port




Specify the TCP/IP port number that the remote device is configured to use. The default port number is 502.

## Device Properties — Settings

### First Word Low

Specify if the driver should assume the first word is the low or high word in 32-bit values. If disabled, the driver will use the Honeywell default FP B. If enabled, the FP LB format will be used.

Two consecutive register addresses are used for 32-bit data types (such as Floats). Users can specify whether the driver should treat the contents of the first register as the low or high word in 32-bit values. The HC900 can be configured to use a number of Double Register Formats. The formats are as follows.

| Format | Description | Byte Order |
|---|---|---|
| FP B* | Floating Point Big Endian | 4, 3, 2, 1 |
| FP BB | Floating Point Big Endian with byte-swap | 3, 4, 1, 2 |
| FP L | Floating Point Little Endian | 1, 2, 3, 4 |
| FP LB** | Floating Point Little Endian with byte-swap | 2, 1, 4, 3 |

*Honeywell default.
**Modbus standard.

### Examples of Data in FP B Format

| Value (decimal) | Value (hex) | Register N | | Register N+1 | |
|---|---|---|---|---|---|
| | | High | Low | High | Low |
| 100.0 | 0x42C80000 | 0x42 | 0xC8 | 0x00 | 0x00 |
| 55.32 | 0x425D47AE | 0x42 | 0x5D | 0x47 | 0xAE |
| 2.0 | 0x40000000 | 0x40 | 0x00 | 0x00 | 0x00 |
| 1.0 | 0x3F800000 | 0x3F | 0x80 | 0x00 | 0x00 |
| -1.0 | 0xBF800000 | 0xBF | 0x80 | 0x00 | 0x00 |

**Note**: If this property is disabled, the driver will use the Honeywell default FP B. If enabled, the FP LB format will be used. The driver does not currently support the Honeywell "FP BB" and "FP L" double register formats.

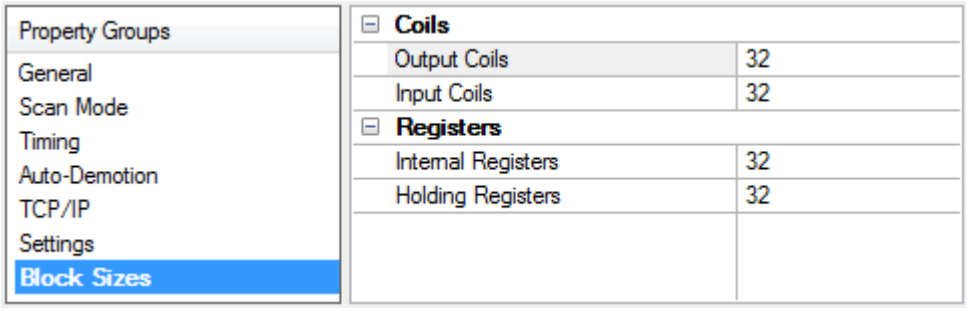
## Device Properties — Block Sizes

### Coils

The **Output Coils** and **Input Coils** properties can be readaccept a value from 8 to 800. This represents the coil points (bits) that can be read at a time. The value must be a multiple of 8 and has a default value of 32.

### Registers

The **Internal Registers** and **Holding Registers** properties can accept a value from 1 to 120. This represents the register locations (words) that can be read at a time. The properties have a default value of 32.




**Note**: Given the overhead involved in sending data via TCP/IP, it is generally advantageous to keep the block size large. Reducing the block size may improve performance, however, if data will be read from non-contiguous locations within the device.

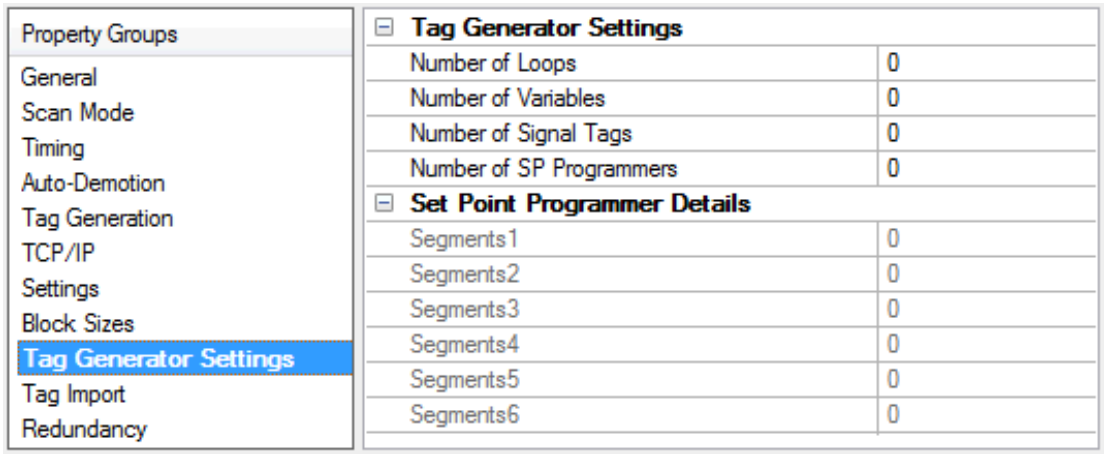
## Device Properties — Tag Generation Settings

The Honeywell HC900 Ethernet Driver has the ability to automatically create tags in the OPC server by using either Tag Generation or Tag Import. Tag Generation creates a set of tags that is based on a general description of the controller's configuration. Tag Import involves importing tags that have been defined in a Hybrid Control Designer application (version 2.1 or later). For more information, refer to **Tag Import**.

**Note**: The driver will generate tags assuming the Universal Modbus Map is in use in the controller. Newer versions of the HC900 may use the Custom Modbus Map, where various control parameters can be mapped to user-defined Modbus Partitions. The driver assumes the Universal Modbus Map is in use by default.

Therefore, if the controller uses the Custom Modbus Map, it is recommended that Tag Import be utilized instead.

The Tag Generation Device Settings of the Tag Generation property group is used to specify how many loops, variables, signal tags and Set Point Programmers are used in the project. The Set Point Programmer Details section of the Tag Generation property group is used to specify the number of segments for each set point programmer. The driver creates a set of tags based on these settings. After all parameters have been specified, click "Apply" or "OK" to begin the tag generation process.




## Number of Loops

This property specifies the number of control loops used in the controller. This property accepts a value of 0 - 32. Tags for the most important loop parameters are created for each loop. The default parameter names are similar to those listed in the HC900 Communications Manual. Loops are listed according to their loop number (e.g., Loop_01) in the order entered in the controller configuration. For information on how to associate the control loop number with the tag name assigned in the controller configuration, refer to **Tag Generation Notes**.

## Number of Variables

This property specifies the number of Read / Write variables used in the controller configuration. This property accepts a value of 0 - 600. Variables are listed by number in the same sequence as in the controller configuration. Each variable will have an analog (Float) and a digital (Boolean) parameter listing. For information on how to select the appropriate type (as well as how to associate the variable number with that assigned in the controller configuration) refer to **Tag Generation Notes**.

## Number of Signal Tags

This property specifies the number of Read Only signal tags used in the controller configuration. This property accepts a value of 0 - 4000. Signal tags are listed by number in the same sequence as in the controller configuration. Each signal tag will have an analog (Float) and a digital (Boolean) parameter listing. For information on how to select the appropriate type (as well as how to associate the signal tag number with that assigned in the controller configuration) refer to **Tag Generation Notes**.

### Number of SP Programmers

This property specifies the number of SP programmers used in the controller. This property accepts a value of 0 - 8. For information on how to associate the set point programmer number with the tag name assigned in the controller configuration, refer to **Tag Generation Notes**. The value specified for this property enables a corresponding number of **Segment** properties in the **Set Point Programmer Details** property section.

**Set Point Programmer Details...**
This property section allows you to specify additional tag generation settings that are specific to each set point programmer. There is a one to one correspondence to the **Number of SP Programmers** specified in the **Tag Generation** property section and enabled Segment properties in the **Set Point Programmer Details** property section. For example, if the **Number of SP Programmers** property is set to 0, no **Segment** properties are enabled. If the **Number of SP Programmers** property is set to 1, property **Segment1** is enabled. If the **Number of SP Programmers** property is set to 2, properties **Segment1 – Segment2** are enabled, etc. An enabled **Segment** property accepts a value of 0 – 50.

Tags generated for each set point programmer are categorized into three tag groups: Parameters, Additional Parameters and Segment-Specific Parameters. The number of Segment-Specific tag groups corresponds to the value specified in each enabled **Segment** property. Descriptions are as follows:

- **Parameters:** This group contains the current programmer status parameters including those specific to the current segment and programmer start, hold, advance and reset.
- **Additional Parameters:** This group contains other setup parameters plus the current program number, the program save request that assigns a set of parameters previously written to the programmer block to a stored profile number and the auxiliary output status (if used).
- **Segment-Specific Parameters:** These groups are specific to each segment for profile setup.

**Note**: For more information on using these parameters for program monitoring and profile setup, refer to the HC900 Communications User Manual.

## Tag Generation Notes

1. Depending on the device's configuration, variables and signal tags can be used for either analog or digital data. In either case, the actual data stored in the device will be in IEEE Floating point format. For digital data, TRUE/ON will be stored as 1.0 and FALSE/OFF will be stored as 0.0. Users may read and write to any of these locations using tags with a data type of Float; however, many client applications will not be able to handle "digital" values represented as Floats. This is because TRUE is typically coerced into a Float value of –1.0. The driver will automatically do the required conversion to or from a true Boolean if the tag is created with the Boolean data type.
2. Each variable or signal tag is created with two listings, with different tag names ending in B (for Boolean) or F (for Float); thus, representing the digital and analog data types. Users do not need to specify ahead of time what type of data each tag will be accessing. Instead, they only need to configure the client application to use the appropriate version of the tag. Users may also delete any unused tags from the server: they are of no consequence as long as the client does not use them.
3. All tags generated by this driver are given generic names. To help associate the tags generated by this driver with the tags configured in the controller, it is recommended that two reports be printed from the Honeywell Hybrid Control Designer application relative to the controller configuration. To do so, click **File** | **Print Report Preview** and then select **FBDs**.
    - For information on variables and signal tags, select **Tag Information**. The Tag Generation Wizard will list the variables and signal tags in the same numerical order as documented.
    - For information about loop order and SP programmer order, select the **Block Modbus Addresses** report. The Tag Generation Wizard will list the loops and SP programmers in the same numerical order as documented.

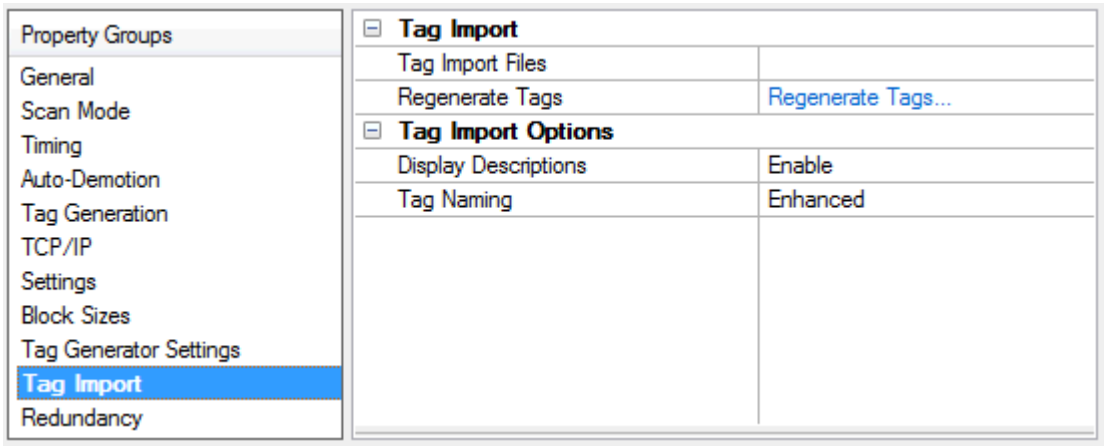
    **Note**: Users should record the associated tag names for these blocks.

4. If Hybrid Control Designer version 2.1 or later is being used, the Tag Import method may be preferable because it only imports the tags defined in the project. Additionally, the imported tags will have the same (or similar) names as the tags in the controller.

**See Also:** Tag Import

## Device Properties — Tag Import

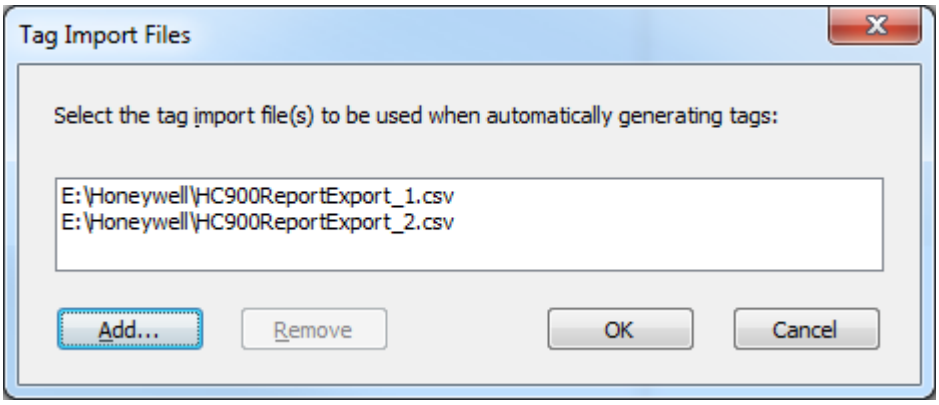
The Honeywell HC900 Ethernet Driver can automatically create tags in the OPC server through either Tag Import or Tag Generation. Tag Import involves importing tags that have been defined in a Hybrid Control Designer application (version 2.1 or later). It is the preferred method because it only imports the tags defined in the controller and no others. Tag Generation creates a set of tags based on a general description of the controller's configuration. For more information, refer to Tag Generation.




### Tag Import Files

This property specifies a pipe delimited (|) list of file paths to be imported. Applying changes to this property will regenerate tags.

- This property's file list can be manually entered.
- Alternatively, the property's browse button can be clicked to launch the Tag Import Files dialog.




- **Add...** Launches a file browser to select and add a tag import file to the list.
- **Remove** Removes a selected tag import file from the list. This button is only enabled if a file is selected. This button is used to remove the selected tag import file from the list.

- **Regenerate** Regenerates all tags as specified by the Tag Import Files and Tag Generation property groups. Any property changes will be applied before tag regeneration.
- **Display Descriptions** Specify Enabled to display descriptions defined in the Hybrid Control Designer as the tag descriptions in the OPC server. Descriptions for OPC server tags are limited to 64 characters and will be truncated if necessary.
- **Tag Naming** This property is used to specify the tag naming option (which will specify the default for all new devices added thereafter). Options include Enhanced or Legacy. Descriptions are as follows:
  - **Enhanced:** This option has fewer naming constraints and is consistent with the naming requirements of the current OPC server. Tag names cannot have a period, double quotes or start with an underscore.
  - **Legacy:** This option enforces the stricter naming requirements of previous versions of this driver. Tag names must start with a letter and the name must consist of letters and digits only.

**Note**: For existing projects, the default option of this property is Legacy.

## Tag Import Notes

The Hybrid Control Designer can generate many different types of export files, which are only used by this driver and are given as Hybrid Control Designer menu options. For more information, refer to **Creating Tag Import Files**.

1. FBD | Modbus Register Map | Detailed Function Block Report.
2. FBD | Modbus Register Map | User-Defined Signals and Variables.
3. FBD | Tag Information | Signal Tags.
4. FBD | Tag Information | Variables.
5. FBD | Tag Information | Signal Tags and Variables.
6. FBD | Modbus Partitions | (Selected Partition Name)(Used for Custom Map).
7. FBD | All Modbus Registers.
8. FBD | All Modbus Partitions (Used for Custom Map).

## Additional Notes

1. Using options 3 + 5 or 4 + 5, users can create files with duplicate information. If used in the driver's import tag file list, this will result in the creation of duplicate tags.
2. Option 6 refers to the selection of an individual named partition (from a list of all named partitions), while option 8 refers to the selection of all named partitions.
3. Using 7 + any other option will result in duplicate tags.
4. The preferred method for importing all tags is a single file. Use option 6 for applications utilizing the Fixed Modbus Map (default for HC900). If the Custom Modbus Map (for Modbus Partitions) is utilized in HC Designer, use option 7.
5. Tags generated from an import operation will be placed in groups. For file type 2, the driver will create groups called "UserDefinedVariables" and "UserDefinedSignalTags." For file types 3, 4, and 5, the driver will create groups called "TagInfoVariables" and "TagInfoSignalTags." For file type 1, the driver will create groups based on the block names given in the file. For Custom Map partition import, the Partition name will be used as a primary group name.

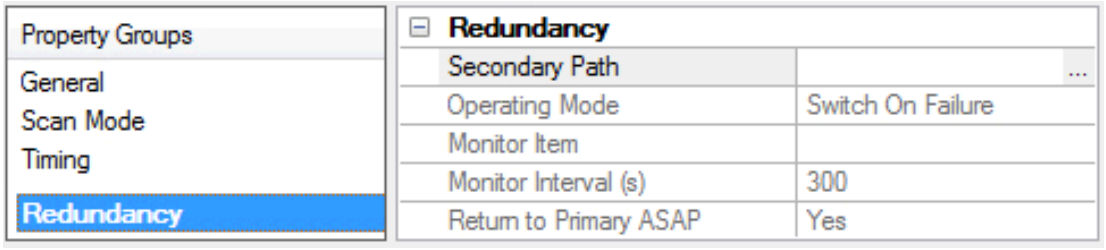
6. A leading underscore in group names will be replaced with an "A."
7. Tag names may be modified slightly to create valid OPC server tag names. Digits may be appended to the end of a tag name to make the name unique. Any name changes will be reported in the server's event log. A leading underscore in tag names will be replaced with 0 (zero).

### OPC Server Requirements for Valid Tag Names

The following is required by the OPC server for a valid tag name.

- The tag name cannot start with an underscore_.
- Periods, double quotes and back slashes are not allowed.
- The tag name must be no longer than 256 characters.
- Tags may have the same name, as long as they are in different groups.

## Device Properties — Redundancy




Redundancy is available with the Media-Level Redundancy Plug-In.
*Consult the website, a sales representative, or the user manual for more information.*

## Data Types Description

| Data Type | Description |
|---|---|
| Boolean | Single bit |
| Word | Unsigned 16-bit value<br><br>bit 0 is the low bit<br>bit 15 is the high bit |
| Short | Signed 16-bit value<br><br>bit 0 is the low bit<br>bit 14 is the high bit<br>bit 15 is the sign bit |
| DWord | Unsigned 32-bit value<br><br>bit 0 is the low bit<br>bit 31 is the high bit |
| Long | Signed 32-bit value<br><br>bit 0 is the low bit<br>bit 30 is the high bit<br>bit 31 is the sign bit |
| BCD | Two byte packed BCD<br><br>Value range is 0-9999. Behavior is undefined for values beyond this range. |
| LBCD | Four byte packed BCD<br><br>Value range is 0-99999999. Behavior is undefined for values beyond this range. |
| Float | 32-bit Floating point value.<br><br>The driver interprets two consecutive registers as a Floating-point value by making the second register the high word and the first register the low word. |
| Float | If register 40001 is specified as a Float, bit 0 of register 40001 would be bit 0 of the 32-bit word, and bit 15 of register 40002 would be bit 31 of the 32-bit word. |

## Address Descriptions

Address specifications vary depending on the model in use. Select a link from the following list to obtain specific address information for the model of interest.

Output Coils
Input Coils
Internal Registers
Holding Registers

## Output Coils

The default data type is shown in **bold**.

### Decimal Addressing

| Address | Range | Data Type | Access |
|---|---|---|---|
| 0xxxxx | 1-65536 | **Boolean** | Read/Write |

### Hexadecimal Addressing

| Address | Range | Data Type | Access |
|---|---|---|---|
| H0yyyyy | 1-10000 | **Boolean** | Read/Write |

**Note**: Honeywell documentation often gives the location of parameters in device memory as a 1-based coil or register number in decimal and a 0-based Modbus address in hex. The coil or register number is the Modbus address plus 1. For example, Honeywell documentation may discuss "Variable #1" at "holding register 6337 with Modbus address 0x18C0." This may be confusing. Users should make sure that the coil or register number is specified when creating tags with this driver, instead of the given Modbus address. This number may be expressed in decimal (as in the Honeywell documentation) or in hex.

**Important:** In hex, the coil or register number is not the same value as the Modbus address given in the Honeywell documentation. It is one greater.

**Example:**
The 255'th output coil would be addressed as '0255' (using decimal addressing) or 'H0FF' (using hexadecimal addressing).

## Input Coils

### Decimal Addressing

| Address | Range | Data Type | Access |
|---|---|---|---|
| 1xxxxx | 1-65536 | Boolean | Read Only |

### Hexadecimal Addressing

| Address | Range | Data Type | Access |
|---|---|---|---|
| H1yyyyy | 1-10000 | Boolean | Read Only |

**Note**: Honeywell documentation often gives the location of parameters in device memory as a 1-based coil or register number in decimal and a 0-based Modbus address in hex. The coil or register number is the

Modbus address plus 1. For example, Honeywell documentation may discuss "Variable #1" at "holding register 6337 with Modbus address 0x18C0." This may be confusing. Users should make sure that the coil or register number is specified when creating tags with this driver, instead of the given Modbus address. This number may be expressed in decimal (as in the Honeywell documentation) or in hex.

**Important:** In hex, the coil or register number is not the same value as the Modbus address given in the Honeywell documentation. It is one greater.

**Example:**
The 127'Th input coil would be addressed as '1127' (using decimal addressing) or 'H17F' (using hexadecimal addressing).

## Internal Registers

The default data types are shown in **bold**.

### Decimal Addressing

| Address | Range | Data Type | Access |
|---|---|---|---|
| 3xxxxx | 1-65536 | **Word**, Short, BCD | Read Only |
| 3xxxxx.bb | xxxxx.0-xxxxx.15 | **Boolean** | Read Only |
| 3xxxxx | 1-65535 | Float, DWord, Long, LBCD | Read Only |

### Hexadecimal Addressing

| Address | Range | Data Type | Access |
|---|---|---|---|
| H3yyyyy | 1-10000 | **Word**, Short, BCD | Read Only |
| H3yyyyy.c | yyyyy.0-yyyyy.F | **Boolean** | Read Only |
| H3yyyyy | 1-FFFF | Float, DWord, Long, LBCD | Read Only |

**Note**: Honeywell documentation often gives the location of parameters in device memory as a 1-based coil or register number in decimal and a 0-based Modbus address in hex. The coil or register number is the Modbus address plus 1. For example, Honeywell documentation may discuss "Variable #1" at "holding register 6337 with Modbus address 0x18C0." This may be confusing. Users should make sure that the coil or register number is specified when creating tags with this driver, instead of the given Modbus address. This number may be expressed in decimal (as in the Honeywell documentation) or in hex.

**Important:** In hex, the coil or register number is not the same value as the Modbus address given in the Honeywell documentation. It is one greater.

### Arrays

Arrays are also supported for the holding register addresses. The syntax for declaring an array using decimal addressing is as follows:

3xxxx[cols]
with assumed row count of 1 and 3xxxx[rows][cols].

For Word, Short and BCD arrays, the base address + (rows * cols)-1 cannot exceed 65536. For Float, DWord, Long and Long BCD arrays, the base address + (rows * cols * 2)-1 cannot exceed 65535. For all arrays, the total number of registers being requested cannot exceed the internal register block size that was specified for the device.

## Holding Registers

The default data types are shown in **bold**.

### Decimal Addressing

| Address | Range | Data Type | Access |
|---|---|---|---|
| **4xxxxx** | 1-65536 | **Word**, Short, BCD | Read/Write |
| 4xxxxx.bb | xxxxx.0-xxxxx.15 | **Boolean** | Read/Write |
| 4xxxxx:DF | 1-65536 | Boolean (Float)* | Read Only |
| 4xxxxx:DW | 1-65536 | Boolean (Word)* | Read Only |
| 4xxxxx:DDW | 1-65536 | Boolean (DWord)* | Read Only |
| 4xxxxx | 1-65535 | Float, DWord, Long, LBCD | Read/Write |

**For more information, refer to **Variables and Signal Tags**.*

### Hexadecimal Addressing

| Address | Range | Data Type | Access |
|---|---|---|---|
| H4yyyyy | 1-10000 | **Word**, Short, BCD | Read/Write |
| H4yyyyy.c | yyyyy.0-yyyyy.F | **Boolean** | Read/Write |
| H4yyyyy:DF | 1-10000 | Boolean (Float)* | Read Only |
| H4yyyyy:DW | 1-10000 | Boolean (Word)* | Read Only |
| H4yyyyy:DDW | 1-10000 | Boolean (DWord)* | Read Only |
| H4yyyyy | 1-FFFF | Float, DWord, Long, LBCD | Read/Write |

**For more information, refer to **Variables and Signal Tags**.*

**Note**: Honeywell documentation often gives the location of parameters in device memory as a 1-based coil or register number in decimal and a 0-based Modbus address in hex. The coil or register number is the Modbus address plus 1. For example, Honeywell documentation may discuss "Variable #1" at "holding register 6337 with Modbus address 0x18C0." This may be confusing. Users should make sure that the coil or register number is specified when creating tags with this driver, instead of the given Modbus address. This number may be expressed in decimal (as in the Honeywell documentation) or in hex.

**Important:** In hex, the coil or register number is not the same value as the Modbus address given in the Honeywell documentation. It is one greater.

### Tag Addressing and Data Type Assignment

Since this driver is derived from the Modbus Ethernet driver, it allows tremendous flexibility in addressing and data type assignment. No effort is made to restrict address and data type options to reflect the complexities of the data mapping in Honeywell HC900 devices. Thus, it is possible to configure tags that will not access data stored in the device correctly. For example, users could create a tag addressing register 6337 with a data type of word. Such a tag would not be useful here since this is the starting address of "Variable #1" and will always contain Floating point data. Likewise, a tag addressing register 6338 as a Float would not be useful either, since variables are mapped to registers 6337, 6339, 6341 and etc. These are starting addresses. Each value uses two registers. For information on the correct starting address and data type for each parameter, refer to the device documentation.

## Variables and Signal Tags

Depending on the device configuration, variables and signal tags can be used for either analog or digital data. If the data type is a digital Float, the actual data stored in the device will be in IEEE Floating point format. TRUE/ON will be stored as 1.0 and FALSE/OFF will be stored as 0.0. Users may read and write to any of these locations using tags with a data type of Float. Many client applications, however, will not be able to handle digital values represented as Floats because TRUE is typically coerced into a Float value of -1.0. In this case, tags that address digital variables and signals with a data type of Boolean should be created. The driver will automatically convert these values to and from true Booleans.

Likewise, digital Words and DWords are stored in the device as Words and DWords respectively, although they may be addressed with a data type of Boolean. TRUE/ON will be stored as 1 and FALSE/OFF will be stored as 0. In this case, tags that address digital variables and signals with a data type of Boolean should be created. The driver will automatically convert these values to and from true Booleans. For more information on Boolean-to-data type conversion, refer to the table below.

| Data Type in Driver | Data Type on Controller | Address |
|---|---|---|
| Boolean | Float | 4xxxx1 |
| Boolean | Float | 4xxxx1:DF |
| Boolean | Word | 4xxxx1:DW |
| Boolean | DWord | 4xxxx1:DDW |

**Note:** These are legacy projects.

## Arrays

Arrays are also supported for the holding register addresses. The syntax for declaring an array (using decimal addressing) is as follows:

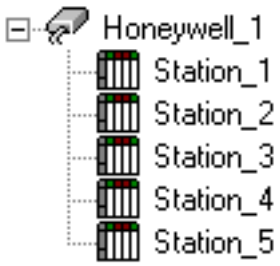
4xxxx[cols]
with assumed row count of 1 and 4xxxx[rows][cols].

For Word, Short and BCD arrays, the base address + (rows * cols) - 1 cannot exceed 65536. For Float, DWord, Long and Long BCD arrays, the base address + (rows * cols * 2) - 1 cannot exceed 65535. For all arrays, the total number of registers being requested cannot exceed the holding register block size that was specified for this device.

## Optimizing Communications

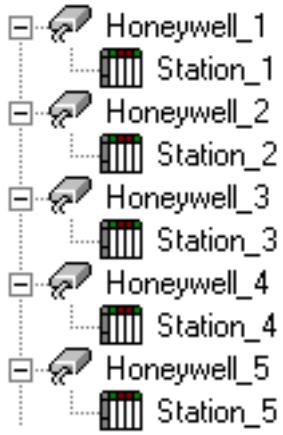
The Honeywell HC900 Ethernet Driver has been designed to provide the best performance with the least amount of impact on the system's overall performance. While the Honeywell HC900 Ethernet Driver is fast, there are a couple of guidelines that can be used in order to control and optimize the application and gain maximum performance.

Our server refers to communications protocols like Honeywell HC900 Ethernet as a channel. Each channel defined in the application represents a separate path of execution in the server. Once a channel has been defined, a series of devices must then be defined under that channel. Each of these devices represents a single Honeywell HC900 controller from which data will be collected. While this approach to defining the application will provide a high level of performance, it won't take full advantage of the Honeywell HC900 Ethernet Driver or the network. An example of how the application may appear when configured using a single channel is shown below.




Each device appears under a single Honeywell HC900 Ethernet channel. In this configuration, the driver must move from one device to the next as quickly as possible in order to gather information at an effective rate. As more devices are added or more information is requested from a single device, the overall update rate begins to suffer.

If the Honeywell HC900 Ethernet Driver could only define one single channel, then the example shown above would be the only option available; however, the Honeywell HC900 Ethernet Driver can define up to 16 channels. Using multiple channels distributes the data collection workload by simultaneously issuing multiple requests to the network. An example of how the same application may appear when configured using multiple channels to improve performance is shown below.




Each device has now been defined under its own channel. In this new configuration, a single path of execution is dedicated to the task of gathering data from each device. If the application has 16 or fewer devices, it can be optimized exactly how it is shown here.

The performance will improve even if the application has more than 16 devices. While 16 or fewer devices may be ideal, the application will still benefit from additional channels. Although by spreading the device load across all channels will cause the server to move from device to device again, it can now do so with far less devices to process on a single channel.

Block Size, which is available on each defined device, can also affect the Honeywell HC900 Ethernet Driver performance. Block Size refers to the number of bytes that may be requested from a device at one time. To refine the performance of this driver, may be configured to 1 to 120 registers and 8 to 800 bits.

## Error Descriptions

The following messages may be generated. Click on the link for a description of the message.

### Address Validation

Missing address
Device address '<address>' contains a syntax error
Address '<address>' is out of range for the specified device or register
Device address '<address>' is not supported by model '<model name>'
Data Type '<type>' is not valid for device address '<address>'
Device address '<address>' is Read Only
Array size is out of range for address '<address>'
Array support is not available for the specified address: '<address>'

### Device Status Messages

Device '<device name>' is not responding
Unable to write to '<address>' on device '<device name>'

### Device Specific Messages

Failure to initiate 'winsock.dll'
Bad address in block [x to y] on device '<device name>'
Bad received length [x to y] on device '<device name>'

### Tag Import Specific Messages

Could not read record <record>-Buffer length exceeded
No tags imported-Unsupported file format
Could not parse expected data (field <field>, record <record>)
Invalid decimal address (field <field>, record <record>)
Invalid tag name '<name>' (field <field>, record <record>) could not be coerced into valid name
Invalid tag name '<old name>' (field <field>, record <record>) changed to '<new name>'
Invalid datatype (field <field>, record <record>)
Invalid access (field <field>, record <record>)
Invalid type (field <field>, record <record>)
Invalid tag type (field <field>, record <record>)
Could not import tag in record <record>-unknown block name
Invalid block name '<name>' (field <field>, record <record>) could not be coerced into valid group name
Invalid block name '<old name>' (field <field>, record <record>) changed to '<new name>'

## Missing address

### Error Type:

Warning

### Possible Cause:

A tag address that has been specified statically has no length.

### Solution:
Re-enter the address in the client application.

## Device address '<address>' contains a syntax error

### Error Type:
Warning

### Possible Cause:
An invalid tag address has been specified in a dynamic request.

### Solution:
Re-enter the address in the client application.

## Address '<address>' is out of range for the specified device or register

### Error Type:
Warning

### Possible Cause:
A tag address that has been specified statically references a location that is beyond the range of supported locations for the device.

### Solution:
Verify that the address is correct; if it is not, re-enter it in the client application.

## Device address '<address>' is not supported by model '<model name>'

### Error Type:
Warning

### Possible Cause:
A tag address that has been specified statically references a location that is valid for the communications protocol but not supported by the target device.

### Solution:
Verify that the address is correct; if it is not, re-enter it in the client application. Also verify that the selected model name for the device is correct.

## Data Type '<type>' is not valid for device address '<address>'

### Error Type:
Warning

### Possible Cause:
A tag address that has been specified statically has been assigned an invalid data type.

### Solution:
Modify the requested data type in the client application.

## Device address '<address>' is Read Only

### Error Type:
Warning

### Possible Cause:
A tag address that has been specified statically has a requested access mode that is not compatible with what the device supports for that address.

### Solution:
Change the access mode in the client application.

## Array size is out of range for address '<address>'

### Error Type:
Warning

### Possible Cause:
A tag address that has been specified statically is requesting an array size that is too large for the address type or block size of the driver.

### Solution:
Re-enter the address in the client application to specify a smaller value for the array or a different starting point.

## Array support is not available for the specified address: '<address>'

### Error Type:
Warning

### Possible Cause:
A tag address that has been specified statically contains an array reference for an address type that doesn't support arrays.

### Solution:
Re-enter the address in the client application to remove the array reference or correct the address type.

## Device '<device name>' is not responding

### Error Type:
Serious

### Possible Cause:

1. The connection between the device and the Host PC is broken.
2. The communication parameters for the connection are incorrect.
3. The named device may have been assigned an incorrect Network ID.

4. The response from the device took longer to receive than the amount of time specified in the "Request Timeout" device setting.

### Solution:

1. Verify the cabling between the PC and the device.
2. Verify that the specified communication parameters match those of the device.
3. Verify that the Network ID given to the named device matches that of the actual device.
4. Increase the Request Timeout setting so that the entire response can be handled.

## Unable to write to '<address>' on device '<device name>'

### Error Type:
Serious

### Possible Cause:

1. The named device may not be connected to the network.
2. The named device may have been assigned an incorrect Network ID.
3. The named device is not responding to write requests.
4. The address does not exist in the PLC.

### Solution:

1. Check the PLC network connections.
2. Verify that the Network ID given to the named device matches that of the actual device.

## Failure to initiate 'winsock.dll'

### Error Type:
Fatal

### Possible Cause:
Could not negotiate with the operating systems winsock 1.1 functionality.

### Solution:
Verify that the winsock.dll is properly installed on the system.

## Bad address in block [x to y] on device '<device name>'

### Error Type:
Fatal addresses falling in this block.

### Cause:
This error is reported when the driver attempts to read a location in a PLC that does not exist. For example, in a PLC that only has holding registers 40001 to 41400, requesting address 41405 would generate this

error. Once this error is generated, the driver will not request the specified block of data from the PLC again. Any other addresses being requested that are in this same block will also go invalid.

### Solution:
The client application should be modified to ask for addresses within the range of the device.

## Bad received length [x to y] on device '<device name>'

### Error Type:
Fatal addresses falling in this block.

### Cause:
The driver attempted to read a block of memory in the PLC. The PLC responded with no error, but did not provide the driver with the requested block size of data.

### Solution:
Ensure that the range of memory exists for the PLC.

## Could not read record <record>-Buffer length exceeded

### Error Type:
Warning

### Possible Cause:

1. A record exceeded the maximum allowed length of 1024 characters.
2. File may be corrupted.

### Solution:

1. Edit tag name and description data to reduce length.
2. Regenerate file.

## No tags imported-Unsupported file format

### Error Type:
Warning

### Possible Cause:
The import file is not one of the types the driver is able to read.

### Solution:
Change the import file type to one that is supported.

### See Also:
Creating Tag Import Files

## Could not parse expected data (field <field>, record <record>)

### Error Type:

Warning

### Possible Cause:

1. The field exceed the maximum allowed length of 256 characters.
2. The field delimiter (comma) is missing, possibly due to a file editing error.
3. The file is corrupted.

### Solution:

1. Edit the specified field to reduce length if possible.
2. Edit the file and replace the delimiter.
3. Regenerate the file.

## Invalid decimal address (field <field>, record <record>)

### Error Type:
Warning

### Possible Cause:

1. The data in the specified field is not a decimal value, possibly due to a file editing error.
2. The file is not in one of the supported formats.
3. The file is corrupted.

### Solution:

1. Edit the file and correct the specified field.
2. Regenerate the file.

### See Also:
Creating Tag Import Files

## Invalid tag name '<name>' (field <field>, record <record>) could not be coerced into valid name

### Error Type:
Warning

### Possible Cause:
The tag name specified in the file is not a valid OPC server tag name, and the driver's tag name modification mechanism was unable to create a unique and valid name based on the field.

### Solution:
Manually create the tag.

## Invalid tag name '<old name>' (field <field>, record <record>) changed to '<new name>'

### Error Type:
Information

### Possible Cause:
This is not truly an error. Tag names that are valid in the Hybrid Control Designer are not necessary valid in the OPC server. The driver created a valid tag name based on the name read from the file.

### Solution:
N/A

## Invalid datatype (field <field>, record <record>)

### Error Type:
Warning

### Possible Cause:

1. The datatype in the specified field is not one of the types supported by the driver, possibly due to a file editing error.
2. The file is not in one of the supported formats.
3. The file is corrupted.

### Solution:

1. Edit the file and change the specified field to a supported datatype. Supported types are: "unsigned 16", "signed 16", "unsigned 32", "signed 32", and "Float 32".
2. Regenerate the file.

### See Also:
Creating Tag Import Files

## Invalid access (field <field>, record <record>)

### Error Type:
Warning

### Possible Cause:

1. The access type in the specified field is not one of the types supported by the driver, possible due to a file editing error.
2. The file is not in one of the supported formats.
3. The file is corrupted.

### Solution:

1. Edit the file and change the specified field to a supported access types. Supported types are: "R", "W", and "R/W".
2. Regenerate the file.

**See Also:**
Creating Tag Import Files

## Invalid type (field <field>, record <record>)

### Error Type:
Warning

### Possible Cause:

1. The type in the specified field is not one of the types expected by the driver, possible due to a file editing error.
2. The file is not in one of the supported formats.
3. The file is corrupted.

### Solution:

1. Edit the file and change the specified field to a supported type. Expected types are "Variable" and "Signal Tag". There are many other types used for function block data, but the driver should not be trying to process these when importing a function block file.
2. Regenerate the file.

**See Also:**
Creating Tag Import Files

## Invalid tag type (field <field>, record <record>)

### Error Type:
Warning

### Possible Cause:

1. The tag type in the specified field is not one of the types expected by the driver, possible due to a file editing error.
2. The file is not in one of the expected formats.
3. The file is corrupted.

### Solution:

1. Edit the file and change the specified field to a supported tag type. Expected tag types are "Digital" and "Analog".

2. Regenerate the file.

**See Also:**
Creating Tag Import Files

## Could not import tag in record <record>-unknown block name

### Error Type:
Warning

### Possible Cause:
The driver read a header that indicated that the file was a Detailed Function Block Report. The driver did not find an expected block sub-header record.

### Solution:
Verify that the file is a Detailed Function Block Report, and that the block sub-headers are present. Regenerate the file if needed.

## Invalid block name '<name>' (field <field>, record <record>) could not be coerced

### Error Type:
Warning

### Possible Cause:
The tag name specified in the file is not a valid OPC server group name, and the driver's tag name modification mechanism was unable to create a unique and valid name based on field.

### Solution:
Edit the file and modify the tag name in the block subheader.

## Invalid block name '<old name>' (field <field>, record <record>) changed to '<new>'

### Error Type:
Information

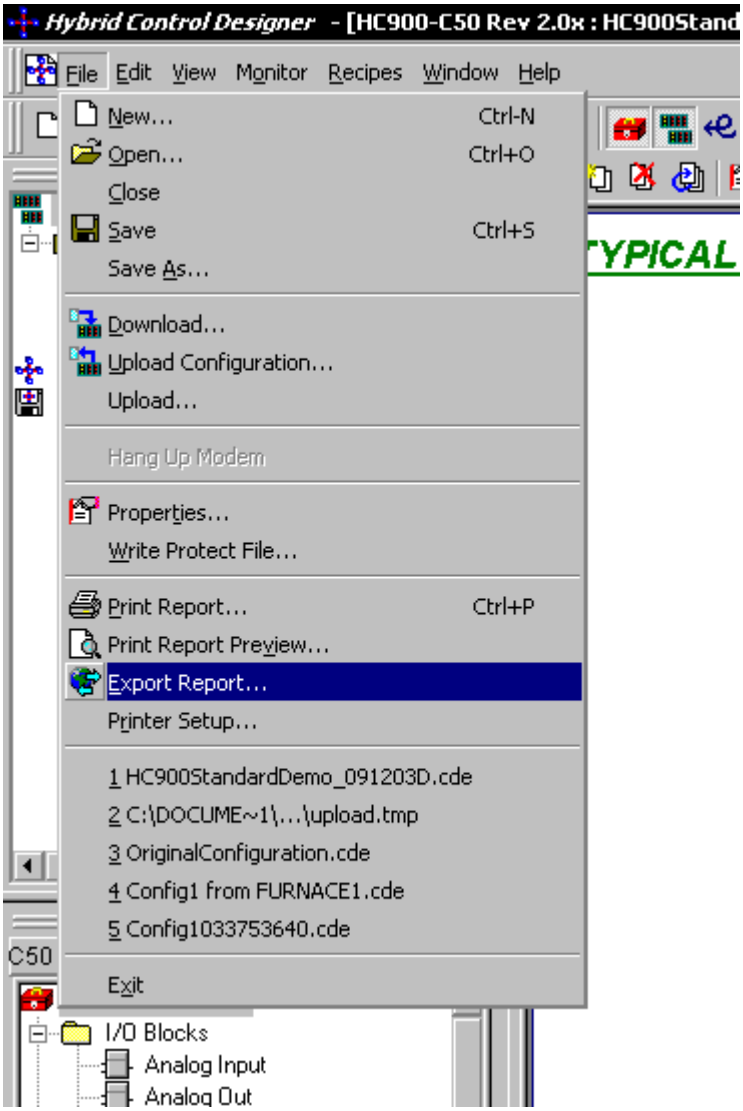
### Possible Cause:
This is not truly an error. Tag names that are valid in the Hybrid Control Designer are not necessary valid in the OPC server. The driver created a valid group name based on the name read from the file.

### Solution:
N/A

## Appendix: Creating Tag Import Files

The Honeywell Hybrid Control Designer version 2.1 and later is used to export controller configuration data to Comma Separated Variable (CSV) format files. The files that contain tag information can in turn be imported by this driver. For more information on driver configuration, refer to **Tag Import**. For instructions on how to create a tag import file, refer to the information below.

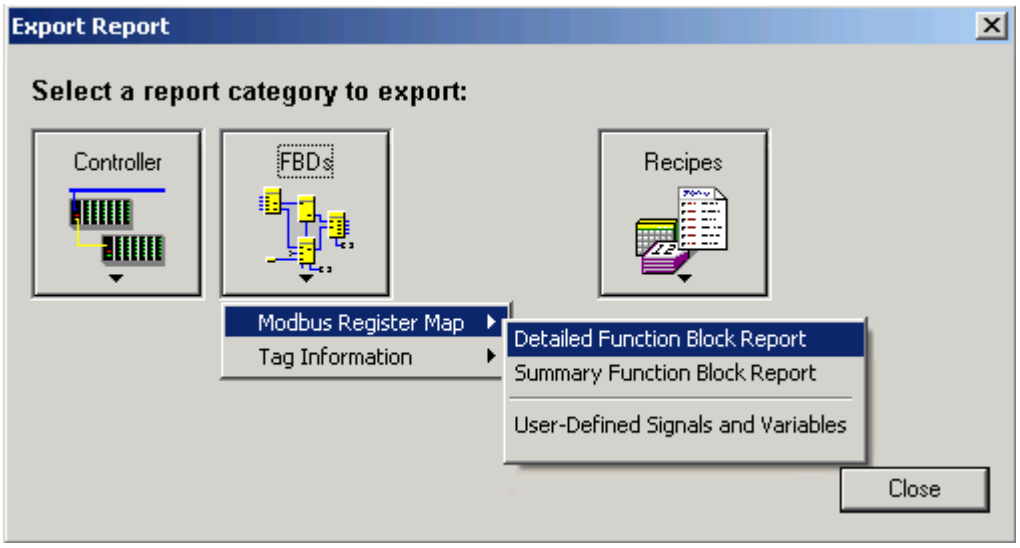
1. To create an export file, start the Hybrid Control Designer and then open a configuration file. Alternatively, upload a controller configuration.
2. Next, click **File** | **Export Report...**.



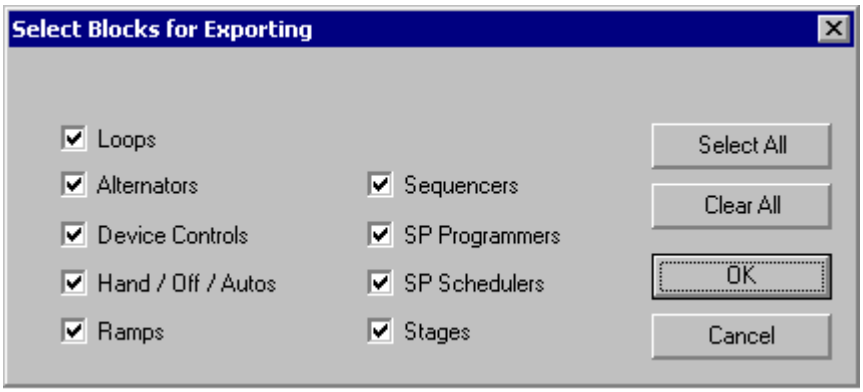

**Note:** Users will be given the option to export Controller, FBDs or Recipes data. Controller and Recipes data will not be of use to the driver. Of the FBDs options, only the Summary Function Block Report will not be of use to the driver. One or more of the files generated by any of the other FBDs options may be imported.

### Example One

1. To create a file describing the tags used by the various function blocks defined in the controller, click **Modbus Register Map** | **Detailed Function Block Report**.




2. Next, specify the types of function blocks that will be included in the export file.

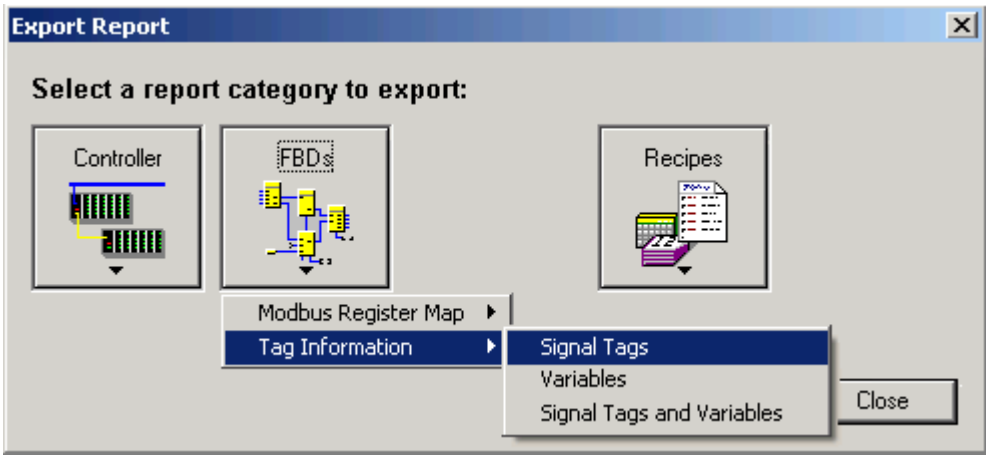



3. Click **OK** and then specify a file name.

**Note:** The file created can now be used directly by the driver's tag import feature.

## Example Two

1. To create a file describing all of the Signal Tags in use by the controller, click **Tag Information** | **Signal Tags**.

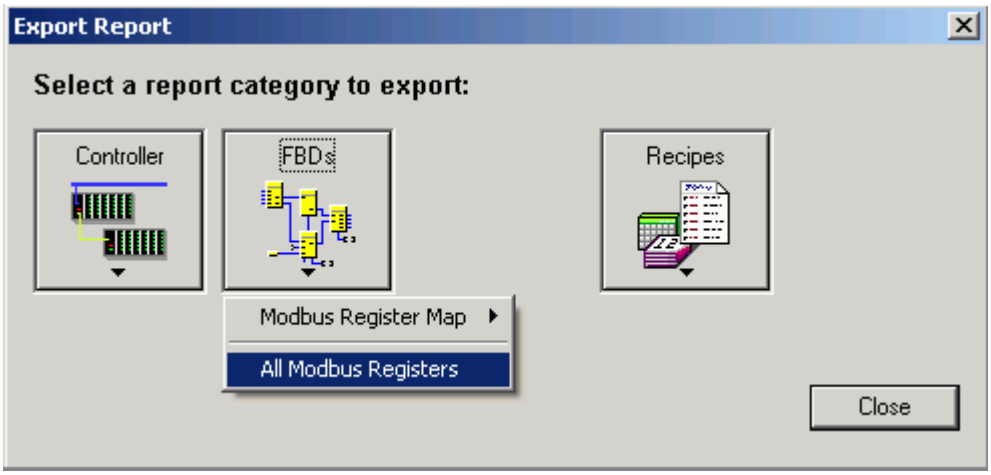



2. After all selections have been made, users will be prompted to specify a file name.

**Note**: The file created can now be used directly by the driver's tag import feature.

## Example Three

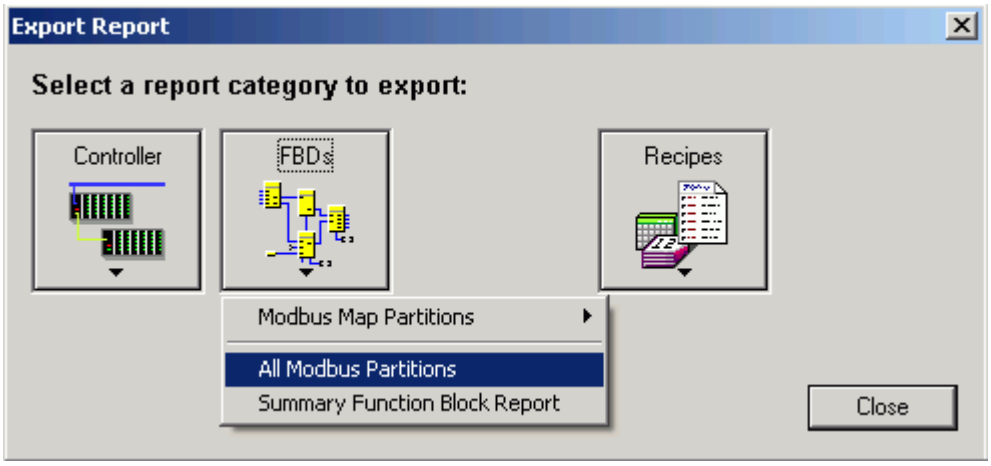
1. To create a file describing all of the tags for an HC900 configuration using the Fixed Modbus Map (the default), select **All Modbus Registers**.




2. Next, specify the file name for import.

## Example Four

1. To create a file for all named Modbus partitions using a Custom Modbus Map (HC900 ver. 4.0 firmware and later) select **All Modbus Partitions**.




2. Next, define the file name for import. Users may choose to delete partitions that do not contain pertinent tags or that are empty. They may also select individual partitions for import.

**Note:** Signal tag and variable data may be interpreted analog values or digital values. For example, for digital Float value, 0.0 for FALSE/OFF or 1.0 for TRUE/ON; for digital Word or DWord, 0 for FALSE/OFF or 1 for TRUE/ON. The driver will automatically assign the appropriate data type of Float for analog values and Boolean for digital values using the data in the Tag Information export file(s). It will also perform any necessary data type conversions during Runtime. Boolean-to-data type conversions are not performed for user-defined signal tags and variables. For more information, refer to **Holding Registers**.

**Important:** The driver uses the header records in each import file to determine its content. Do not modify these headers.

# Resources

In addition to this user manual, there are a variety of resources available to assist customers, answer questions, provide more detail about specific implementations, or help with troubleshooting specific issues.

**Knowledge Base**
**Whitepapers**
**Connectivity Guides**
**Technical Notes**
**Training Programs**
**Training Videos**
**Kepware Technical Support**
**PTC Technical Support**

# Index

## A



## B



## C

## D



## E



## F



## G

## H



## I



## L



## M



## N

## O



## P



## R



## S

## T



## U



## W