

# Simulator Driver

© 2024 PTC Inc. All Rights Reserved.

# Table of Contents

<b>Simulator Driver</b> .....	<b>1</b>
<b>Table of Contents</b> .....	<b>2</b>
Simulator Driver .....	3
Overview .....	3
<b>Setup</b> .....	<b>4</b>
Channel Properties — General .....	5
Tag Counts .....	5
Channel Properties — Write Optimizations .....	6
Channel Properties — Advanced .....	7
Channel Properties — Persistence .....	7
Configuration API Service — Channel Properties .....	8
Device Properties — General .....	9
Operating Mode .....	10
Tag Counts .....	10
Device Properties — Scan Mode .....	10
Device Properties — Tag Generation .....	11
Configuration API Service — Device Properties .....	13
<b>Data Types Description</b> .....	<b>15</b>
<b>Address Descriptions</b> .....	<b>17</b>
8-Bit Device Addresses .....	17
16-Bit Device Addresses .....	18
Simulation Functions .....	18
Ramp Function .....	19
Random Function .....	20
Sine Function .....	20
User Function .....	20
<b>Event Log Messages</b> .....	<b>22</b>
Could not load item state data. Reason: <reason>. .....	22
Could not save item state data. Reason: <reason>. .....	22
<b>Index</b> .....	<b>23</b>

## Simulator Driver

---

### CONTENTS

#### [Overview](#)

What is the Simulator Driver?

#### [Setup](#)

How do I configure a device for use with this driver?

#### [Data Types Description](#)

What data types can be used with a simulated device?

#### [Address Descriptions](#)

How are addresses specified on a simulated device?

#### [Event Log Messages](#)

What error messages does the Simulator Driver produce?

Version 1.047

© 2024 PTC Inc. All Rights Reserved.

### Overview

---

The Simulator Driver provides a reliable way to connect Simulator devices to OPC client applications; including HMI, SCADA, Historian, MES, ERP, and countless custom applications. It is provided for testing the OPC Server software product without requiring an external device.

---

## Setup

---

### Supported Devices

8-Bit  
16-Bit

● **Note:** Each device supports up to 10000 addressable Read / Write register and constant locations, in addition to 1000 variable length, Read / Write string locations. *For more information, refer to [Address Descriptions](#).*

### Channel and Device Limits

The maximum number of channels supported by this driver is 1024. The maximum number of devices supported by this driver is 999 per channel.

### Device ID

Simulator devices can be assigned Device IDs in the range of 1 to 999. When using different model types within the same channel, unique Device IDs are required.

### Live Data Simulation

The driver simulates live data by incrementing register data each time it is read as an integer data type. In addition to simulating the simple register-based memory of a PLC-like device, the Simulator Driver also supports four high-level simulation functions. These new simulation functions include RAMP, SINE, RANDOM, and USER Defined.

Each new simulation tag is structured like a function call would be in a programming language. Using each function requires that the appropriate properties be applied to determine the desired simulation effect. With RAMP, there is the option to set the rate of change, the low limit, the high limit, and the increment value. With SINE, there is the option to set the rate of change, the low limit, the high limit, the frequency, and the phase. With RANDOM, users have the option to set the rate of change, the low limit, and the high limit. The most creative of the new simulation functions, however, is the USER Defined function.

The USER Defined function similarly provides the option to specify a rate of change. Unlike the preset simulation outputs of the RAMP, RANDOM, and SINE functions, the USER Defined function is used to create sequences of data. In the place of the high limits or low limits, the USER Defined function accepts a comma separated list of items. The list of items can be either numeric data or string data. Once a list is established, the USER Defined function cycles through the elements of the list at the rate specified. The USER Defined function can create complex demos that can be used to mirror real world outputs and results.

● *For more information, refer to [Simulation Functions](#).*

## Channel Properties — General

This server supports the use of multiple simultaneous 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.

Property Groups	[-] <b>Identification</b>	
<b>General</b>	Name	
Write Optimizations	Description	
Advanced	Driver	
	[-] <b>Diagnostics</b>	
	Diagnostics Capture	Disable
	[-] <b>Tag Counts</b>	
	Static Tags	10

### Identification

**Name:** Specify the 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. The property is required for creating a channel.

• For information on reserved characters, refer to "How To... Properly Name a Channel, Device, Tag, and Tag Group" in the server help.

**Description:** Specify user-defined information about this channel.

• Many of these properties, including Description, have an associated system tag.

**Driver:** Specify the protocol / driver for this channel. Specify the device driver that was selected during channel creation. It is a disabled setting in the channel properties. The property is required for creating a channel.

• **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. Changes to the properties should not be made once a large client application has been developed. Utilize proper user role and privilege management to prevent operators from changing properties or accessing 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 not available if the driver does not support diagnostics.

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

### Tag Counts

**Static Tags:** Provides the total number of defined static tags at this level (device or channel). This information can be helpful in troubleshooting and load balancing.

## Channel Properties — Write Optimizations

The server must ensure that the data written from the client application gets to the device on time. Given this goal, the server provides optimization properties to meet specific needs or improve application responsiveness.

Property Groups	<input type="checkbox"/> <b>Write Optimizations</b>	
General	Optimization Method	Write Only Latest Value for All Tags
<b>Write Optimizations</b>	Duty Cycle	10

### 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.

Property Groups	<input type="checkbox"/> <b>Non-Normalized Float Handling</b>	
General	Floating-Point Values	Replace with Zero
Write Optimizations	<input type="checkbox"/> <b>Inter-Device Delay</b>	
<b>Advanced</b>	Inter-Device Delay (ms)	0

**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.

## Channel Properties — Persistence

Property Groups	<input type="checkbox"/> <b>Persistence</b>	
General	Item Persistence	Disable
Write Optimizations	Data File	C:\ProgramData\...
Advanced		
<b>Persistence</b>		

**Item Persistence:** Select Enable to configure the device to retain its values of K (constant), R (register), and S (string) addresses between runs. Item persistence for simulation functions is not currently supported. When enabled, the values stored in all K, R, and S addresses of each device configured on the channel are saved to file when the server project is closed (or the server is shut down). These values are restored from file the next time the server project is opened. A separate file is used for each channel using this feature. The default setting is disabled.

● *For more information, refer to [Simulation Functions](#).*

**Data File:** Specify where the data should be stored for devices on this channel. A fully qualified path and file name is required. The driver creates the file and folders in its path, but users must use this feature to specify a unique file for each Simulator channel.

---

## Configuration API Service — Channel Properties

---

The following properties define a channel using the Configuration API service.

### General Properties

`common.ALLTYPES_NAME` \* Required parameter

● **Note:** Changing this property causes the API endpoint URL to change.

`common.ALLTYPES_DESCRIPTION`

`servermain.MULTIPLE_TYPES_DEVICE_DRIVER` \* Required parameter

`servermain.CHANNEL_DIAGNOSTICS_CAPTURE`

### Ethernet Communication Properties

`servermain.CHANNEL_ETHERNET_COMMUNICATIONS_NETWORK_ADAPTER_STRING`

### Advanced Properties

`servermain.CHANNEL_NON_NORMALIZED_FLOATING_POINT_HANDLING`

### Write Optimizations

`servermain.CHANNEL_WRITE_OPTIMIZATIONS_METHOD`

`servermain.CHANNEL_WRITE_OPTIMIZATIONS_DUTY_CYCLE`

● **See Also:** *The server help system Configuration API Service section.*



## 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.

Property Groups	Identification	
General	Name	
Scan Mode	Description	
	Channel Assignment	
	Driver	
	Model	
	ID Format	Decimal
	ID	2

### Identification

**Name:** Specify 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:** Specify the user-defined information about this device.

● Many of these properties, including Description, have an associated system tag.

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

**Driver:** Selected protocol driver for this device.

**Model:** Specify the type of device that is associated with this ID. The contents of the drop-down menu depend 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 documentation.

**ID:** Specify the device's driver-specific station or node. The type of ID entered depends on the communications driver being used. For many communication 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 format is set by the driver by default. Options include Decimal, Octal, and Hexadecimal.

● **Note:** 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

Property Groups	+ Identification	
General	- Operating Mode	
Scan Mode	Data Collection	Enable
	Simulated	No

**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:** Place the device into or out of 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 data is not saved if the server removes the item (such as when the server is reinitialized). The default is No.

**Notes:**

1. Updates are not applied until clients disconnect and reconnect.
2. The 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.
3. 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.
4. When a device is simulated, updates may not appear faster than one (1) second in the client.

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

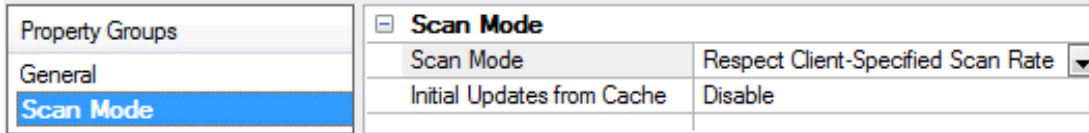
## Tag Counts

Property Groups	- Identification	
General	- Operating Mode	
	- Tag Counts	
	Static Tags	130

**Static Tags:** Provides the total number of defined static tags at this level (device or channel). This information can be helpful in troubleshooting and load balancing.

## 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:** Specify how tags in the device are scanned for updates sent to subscribing 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 value set as the maximum scan rate. 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 OPC 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.

• *Not all devices and drivers support full automatic tag database generation and not all support the same data types. Consult the data types descriptions or the supported data type lists for each driver for specifics.*

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.*

Property Groups	<input checked="" type="checkbox"/> <b>Tag Generation</b>	
General	On Device Startup	Do Not Generate on Startup
Scan Mode	On Duplicate Tag	Delete on Create
Timing	Parent Group	
Auto-Demotion	Allow Automatically Generated Subgroups	Enable
<b>Tag Generation</b>	Create	Create tags
Communications		
Redundancy		

**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:** Specify 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.

## Configuration API Service — Device Properties

---

The following properties define a device using the Configuration API service.

### General Properties

`common.ALLTYPES_NAME` \* Required parameter

`common.ALLTYPES_DESCRIPTION`

`servermain.DEVICE_CHANNEL_ASSIGNMENT`

`servermain.MULTIPLE_TYPES_DEVICE_DRIVER` \* Required parameter

`servermain.DEVICE_MODEL` \* Not required, but verify the default is acceptable

`servermain.DEVICE_ID_STRING` \* Required parameter

`servermain.DEVICE_DATA_COLLECTION`

`servermain.DEVICE_SIMULATED`

### Scan Mode

`servermain.DEVICE_SCAN_MODE`

`servermain.DEVICE_SCAN_MODE_RATE_MS`

servermain.DEVICE\_SCAN\_MODE\_RATE\_MS

servermain.DEVICE\_SCAN\_MODE\_PROVIDE\_INITIAL\_UPDATES\_FROM\_CACHE

## Auto Demotion

servermain.DEVICE\_AUTO\_DEMOTION\_ENABLE\_ON\_COMMUNICATIONS\_FAILURES

servermain.DEVICE\_AUTO\_DEMOTION\_DEMOTE\_AFTER\_SUCESSIVE\_TIMEOUTS

servermain.DEVICE\_AUTO\_DEMOTION\_PERIOD\_MS

servermain.DEVICE\_AUTO\_DEMOTION\_DISCARD\_WRITES


## Tag Generation

servermain.DEVICE\_TAG\_GENERATION\_ON\_STARTUP

servermain.DEVICE\_TAG\_GENERATION\_DUPLICATE\_HANDLING

servermain.DEVICE\_TAG\_GENERATION\_GROUP

servermain.DEVICE\_TAG\_GENERATION\_ALLOW\_SUB\_GROUPS

 **Tip:** To Invoke Automatic Tag Generation, send a PUT with an empty body to the TagGeneration service endpoint on the device.

 **See Also:** For more information, see *Services help*.


## Timing

servermain.DEVICE\_CONNECTION\_TIMEOUT\_SECONDS

servermain.DEVICE\_REQUEST\_TIMEOUT\_MILLISECONDS

servermain.DEVICE\_RETRY\_ATTEMPTS

servermain.DEVICE\_INTER\_REQUEST\_DELAY\_MILLISECONDS

 **See Also:** The server help system *Configuration API Service* section.

## Data Types Description

Each address that can be accessed must be assigned a data type. Simulator devices support the following data types.

Data Type	Description
BCD	Two-byte packed BCD Value range is 0-9999. Behavior is undefined for values beyond this range.
Boolean	Single bit
Byte	Unsigned 8-bit value bit 0 is the low bit bit 7 is the high bit
Char	Signed 8-bit value bit 0 is the low bit bit 6 is the high bit bit 7 is the sign bit
Date	Floating-point OLE automation date (maps to VARIANT VT_DATE data type).
Double*	64-bit floating point value The driver interprets four consecutive registers as a double precision value by making the last two registers the high DWord and the first two registers the low DWord.
Double example	If register K0001 is specified as a double, bit 0 of register K0001 would be bit 0 of the 64-bit data type and bit 15 of register K0004 would be bit 63 of the 64-bit data type.
DWord	Unsigned 32-bit value bit 0 is the low bit bit 31 is the high bit
Float*	32-bit floating point value The driver interprets two consecutive registers as a single precision value by making the last register the high word and the first register the low word.
Float example	If register K0001 is specified as a float, bit 0 of register K0001 would be bit 0 of the 32-bit data type and bit 15 of register K0002 would be bit 31 of the 32-bit data type.
LBCD	Four byte packed BCD Value range is 0-99999999. Behavior is undefined for values beyond this range.
Long	Signed 32-bit value bit 0 is the low bit bit 30 is the high bit bit 31 is the sign bit
LLong	Signed 64-bit value bit 0 is the low bit bit 62 is the high bit bit 63 is the sign bit
QWord	Unsigned 64-bit value bit 0 is the low bit bit 63 is the high bit
Short	Signed 16-bit value bit 0 is the low bit bit 14 is the high bit bit 15 is the sign bit

<b>Data Type</b>	<b>Description</b>
String	Null-terminated ASCII character array
Word	Unsigned 16-bit value bit 0 is the low bit bit 15 is the high bit

\*The descriptions assume the default; that is, first DWord low data handling of 64-bit data types and first word low data handling of 32-bit data types.



## Address Descriptions

The Simulator Driver supports four types of addresses: B registers, R registers, K registers, and S registers. The R and K registers are numeric data. S registers are variable length string data locations.

The B Registers simulate coils and are true Boolean types. Boolean registers do not automatically change values on scan. The R registers simulate changing data by incrementing by one on each read when referenced as type Char, Byte, Word, Short, BCD, Long, DWord, LLong, QWord, or LBCD. Arrays of these types increment by one on each read. When R registers are referenced as type Float or Double, the value is incremented by 1.25 on each read. Arrays of type Float or Double increment by 1.25 on each read. Each type has a range over which the incrementing occurs. For type Float, the range is 0 to 32767. For type Double, the range is 0 to 65535.

The K and R registers have an initial value of zero. S registers have an initial value of 'String data Sn' where *n* is the register number.

Address range and data type specifications vary depending on the model in use. The Simulator Driver also supports new simulation functions, which include RAMP, SINE, RANDOM and USER Defined. For more information, select a link from the list below.

### [8-Bit Device Addresses](#)

### [16-Bit Device Addresses](#)

## 8-Bit Device Addresses

The memory configuration for the 8-bit device is simulated as a block of byte locations numbered from 0 to 9999. Each byte can be addressed as an offset from the start of the block. The default data types for each format are shown in **bold**.

Device Type	Range	Data Type	Access
Registers	R0000-R9999 R0000-R9998 R0000-R9996 R0000-R9992	<b>Byte</b> , Char Word, Short, BCD, DWord, Long, LBCD, Float, LLong, QWord, Double, Date, Boolean	Read/Write
Constants	K0000-K9999 K0000-K9998 K0000-K9996 K0000-K9992	<b>Byte</b> , Char Word, Short, BCD DWord, Long, LBCD, Float LLong, QWord, Double, Date, Boolean	Read/Write
Bits	R0000.0-R9999.7 K0000.0-K9999.7	<b>Boolean</b>	Read/Write
Boolean	B0000-B9999	<b>Boolean</b>	Read/Write
Strings	S000-S999	<b>String</b>	Read/Write

### Notes:

1. All data types except bit-level Boolean support arrays by appending the [r] or [r][c] notation to the address.

2. The maximum array size is based on the upper range of the address type. For example, an array of R registers with a byte data type would have a maximum size of 1000 elements. Example: R0000[1000], maximum size for an array of words, R0000[500].
3. The address specified for a data type must allow for the full size of the data type. This means that users cannot write past the end of the data range.

● **See Also:** [Simulation Functions](#)

## 16-Bit Device Addresses

The memory configuration for the 16-Bit Device is simulated as a block of word locations numbered from 0 to 9999. Each word can be addressed as an offset from the start of the block. The default data types for each format are shown in **bold**.

Device Type	Range	Data Type	Access
Registers	R0000-R9999 R0000-R9998 R0000-R9996	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float LLong, QWord, Double, Date, Boolean	Read/Write
Constants	K0000-K9999 K0000-K9998 K0000-K9996	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float LLong, QWord, Double, Date, Boolean	Read/Write
Bits	R0000.00-R9999.15 K0000.00-K9999.15	<b>Boolean</b>	Read/Write
Boolean	B0000-B9999	<b>Boolean</b>	Read/Write
Strings	S000-S999	<b>String</b>	Read/Write

### ● Notes:

1. All data types except bit-level Boolean support arrays by appending the [r] or [r][c] notation to the address.
2. The maximum array size is based on the upper range of the address type. For example, an array of R registers with a Word data type would have a maximum size of 1000 elements. Example: R0000 [1000], maximum size for an array of words, R0000[500].
3. The address specified for a data type must allow for the full size of the data type. This means that users cannot write past the end of the data range.

● **See Also:** [Simulation Functions](#)

## Simulation Functions

Simulation functions can be used to create OPC items that mimic many real world data sources. While each of the simulation functions provides different outputs, they have many common properties such as **Rate**, **Low Limit**, and **High Limit**. The Rate (which is also called Rate of Change) is used to specify how often the simulation value changes states. The Rate value is entered as a count of milliseconds with the value range being 10 to 3600000. This rate of change is completely independent of how often the client application may be reading data. Like a PLC, the simulation functions are always running in the background.

## The Low Limit and High Limit

The Low Limit and High Limit properties can be used to specify the range or span of data generated by the simulation function. By using the Low Limit and High Limit property, users can produce simulation values that are offset by adjusting the span of the data. For the simulation functions that support limits, the format in which the range is entered is used to determine the data type of the simulation value. If, for example, a RAMP tag is entered with a Low Limit of 75 and a High Limit of 3000 the resulting OPC item is considered to be a Long data type. If the same RAMP is entered with a Low Limit of 75.123 and a High Limit of 3000.567 the resulting OPC item is considered to be Float data type. In these two examples, the default data format was either Float or Long, but any of the available data types can be chosen as the output format for any simulation function. The range specified by the Low and High Limits, however, must fit within the desired data type selection.

Unlike the register-based address above, there is no limit to the number of simulation functions that can be entered. Each simulation function that has unique properties is considered a new simulation value. Thus, when creating addressing simulation functions with the intent of reading the same value in multiple locations in the client application, it is important that the simulation function is entered the same way in each case.

## Simulation Functions are Read-Only Objects

Simulation functions are Read Only objects. Once a client application begins reading data from a simulation function, the function continues to generate data until the item is deleted by a client application. When entering floating point properties, the simulation functions do not accept the entry of numeric values in exponential form.

## Functions Definitions

[Ramp Function](#)

[Random Function](#)

[Sine Function](#)

[User Function](#)

## Ramp Function

---

### RAMP(Rate, Low Limit, High Limit, Increment)

The RAMP function can be used to create a value that increments or decrements through a numeric range. The low limit and high limit should be used to set the desired range. The low or high limits can be adjusted to apply an offset to the data generated. The increment value can be either a positive or negative value. If the increment value is positive, the value generated ramps from the low limit to the high limit at the desired rate. If the increment value is negative, the value generated ramps from the high limit to the low limit at the desired rate. The values of low limit, high limit, and increment can be entered either as whole numbers or in floating-point format.

## Supported Data Types

Byte, Char, Word, QWord, DWord, Short, **Long**, LLong, Float, Double

## Examples

**RAMP(120, 35, 100, 4)**

This creates a value that ramps up from 35 to 100 incremented by 4 every 120 milliseconds.

**RAMP(300, 150.75, 200.50, -0.25)**

This creates a value that ramps down from 200.50 to 150.75 decremented by 0.25 every 300 milliseconds.

---

## Random Function

### **RANDOM(Rate, Low Limit, High Limit)**

The RANDOM function can be used to create an item that changes randomly within a specific numeric range. The Low Limit and High Limit should be used to set the desired range. The Low or High limits can be adjusted to apply an offset to the data generated.

### **Supported Data Types**

Byte, Char, Short, Word, DWord, QWord, **Long**, LLong, Float, Double

### **Example**

#### **RANDOM(30, -20, 75)**

This creates a value that randomly changes within the range of -20 to 75 at a rate of 30 milliseconds.

---

## Sine Function

### **SINE(Rate, Low Limit, High Limit, Frequency, Phase)**

SINE function can be used to create an item that follows a sinusoidal change of value. The Low Limit and High Limit should be used to set the desired range. The Low or High limits can be adjusted to apply an offset to the data generated. The Frequency property can be used to specify the desired waveform in Hertz. The maximum effective frequency is about 5 Hertz. The valid range for the Frequency property is 0.001 to 5 Hertz. The Phase property can be used to offset the sine wave generated by a specific angle. Phase should be entered with a range of 0.0 to 360.0. The Rate property in this case plays a key role in how this simulation function operates. To get a good sinusoidal output from this function, the Rate must at least twice as fast as the desired Frequency. For example, if users desire a sine wave of 5 Hertz which changes at about a 200 millisecond rate, the Rate property should be set to 100 milliseconds at maximum. For the best Sine wave results setting the Rate to either 10 or 20 milliseconds is recommended. The valid range of Rate for a SINE function is 10-1000 milliseconds.

### **Supported Data Types**

Float, Double

### **Example**

#### **SINE(10, -40, 40, 2, 0)**

This creates a sinusoidal value with a frequency of 2 Hertz that ranges from -40 to 40 with no phase shift.

---

## User Function

### **USER(Rate, User Value1, User Value2, User Value3, ...User ValueN)**

The USER function provides the most flexibility in defining what type of data the simulation function returns. Unlike the other functions that operate over a specified range, the USER function can be used to specify a set of numeric or string values to be cycled through at the specified rate. The values entered are used to determine data type of this item. For example, if a value of 100.45 is entered as one of the user values, the output of the simulation object would be considered to be floating point data. If one of the user values entered was "Hello World" the output of the simulation object would be considered to be string data. These default selections can be overridden by specifying the desired data type when the item is defined.

● **Note:** When entering user values as strings the comma can be entered within a string value by first prefixing it with the backslash "\,".

## Supported Data Types

Bool, Byte, Char, Short, Word, DWord, QWord, Long, LLong, Float, Double

● **Note:** The values entered in the USER simulation function are used to determine the default data type.

## Examples

### **USER(250, Hello, World, this, is, a, test)**

This creates a value of data type string that changes from one text word in the sequence to the next at a rate of 250 milliseconds.

### **USER(20, 1.25, 100.56, 200.11,75.1)**

This creates a value of data type float that changes from one floating-point value in the sequence to the next at a rate of 20 milliseconds.

### **USER(50, 1,1,0,1,0,1,0,0,1,1,1,0,0,0)**

This generates a value of type Boolean that changes from one Boolean state in the sequence to the next at a rate of 50 milliseconds. This can be used to create very complex bit patterns.

### **USER(1000, In this case\ , I needed to use a \, in , my text)**

The "\", " in these text fragments were needed to allow a comma to be placed within a text value. Additionally the text for each value can be a sentence or sentence fragment if needed.

# Event Log Messages

The following information concerns messages posted to the Event Log pane in the main user interface. Consult the OPC server help on filtering and sorting the Event Log detail view. Server help contains many common messages, so should also be searched. Generally, the type of message (informational, warning) and troubleshooting information is provided whenever possible.

---

## Could not load item state data. Reason: <reason>.

### Error Type:

Warning

### Possible Cause:

1. The driver could not load the item state data for the specified reason.
2. Corrupt data files.
3. Inadequate disk space.
4. Invalid drive in path.
5. Deleted or renamed data files.

### Possible Solution:

Solution depends upon the reason given in the error message. In the case of file corruption or deletion, previous state data is lost.

---

## Could not save item state data. Reason: <reason>.

### Error Type:

Warning

### Possible Cause:

1. The driver could not save the item state data for the specified reason.
2. Corrupt data files.
3. Inadequate disk space.
4. Invalid drive in path.
5. Deleted or renamed data files.

### Possible Solution:

Solution depends upon the reason given in the error message. In the case of file corruption or deletion, previous state data is lost.

# Index

## 1

16-Bit Device Addresses 18

## 8

8-Bit Device Addresses 17

## A

Address Descriptions 17

Allow Sub Groups 13

## B

BCD 15

Boolean 15

Byte 15

## C

Channel Assignment 9

Channel Properties — Advanced 7

Channel Properties — General 5

Channel Properties — Write Optimizations 6

Char 15

Could not load item state data. Reason  
<reason>. 22

Could not save item state data. Reason  
<reason>. 22

Create 13

## D

Data Collection 10

Data Types Description 15

Date 15  
Delete 12  
Device Properties — General 9  
Device Properties — Tag Generation 11  
Diagnostics 5  
Do Not Scan, Demand Poll Only 11  
Double 15  
Driver 9  
Duty Cycle 6  
DWord 15

## **E**

Event Log Messages 22

## **F**

Float 15

## **G**

General 9  
Generate 12

## **H**

Help Contents 3

## **I**

ID 9  
Identification 5, 9  
Initial Updates from Cache 11  
Inter-Device Delay 7

## **L**

LBCD 15



LLong 15

Long 15

## **M**

Model 9

## **N**

Name 9

Non-Normalized Float Handling 7

## **O**

On Device Startup 12

On Duplicate Tag 12

On Property Change 12

Operating Mode 10

Optimization Method 6

Overview 3

Overwrite 12

## **P**

Parent Group 13

Persistence 7

## **Q**

QWord 15

## **R**

Ramp Function 19

Random Function 20

Registers 17

Replace with Zero 7

Respect Tag-Specified Scan Rate 11

**S**

Scan Mode 11

Setup 4

Short 15

Simulated 10

Simulation Functions 18

Sine Function 20

String 16

**T**

Tag Counts 5, 10

Tag Generation 11

**U**

Unmodified 7

User Function 20

**W**

Word 16

Write All Values for All Tags 6

Write Only Latest Value for All Tags 6

Write Only Latest Value for Non-Boolean Tags 6