



KEPServerEX Tips and Tricks

Document 001-0707

Many of the optimization tips in this document deal with ways to improve the Poll Cycle. The Poll Cycle is the amount of time it takes to request all the items in a device. The main factors which affect Poll Cycle are: how long it takes the device to respond to a request and how many requests are required to obtain all of the data.

1. Optimizing the Device Memory Map

Block Reads

Most device protocols allow for the reading of multiple points of contiguous data in one request. This is referred to as the *blocking* of data. A data block is a range of addresses handles as one unit. To ensure that the server has to perform the smallest number of reads as possible, you should try to design the PLC project so that the items to be read by the server are in contiguous addresses. In the server, data blocks are calculated from the first address.

For example, let's assume that you are using a device that allows a block read of 20 registers and your project has 40 registers. If this data is spread out over a range of 1000 addresses, the server would have to send 40 requests to get all the data. If the data was in 40 contiguous addresses, it would take only 2 reads.

Segregating Data

If only a small portion of the data is required at a fast rate, you can improve performance by placing that data in its own block and using the Group Update Rate in your OPC client configuration to poll that data at a different rate.

2. Optimizing Communications with Devices

Auto Demotion & Multi-dropped Networks

When connecting to a multi-dropped serial network (i.e., RS485 network with multiple devices) you should configure your KEPServerEX project to have multiple devices on a single channel. Note that this is true even when you connect to a multi-dropped serial network via Terminal / Device servers. This means that each device will add to the total poll cycle time on that server channel. Also, if a device becomes unresponsive, the server has to time out on that device during every poll cycle before moving on to the next device.

To improve performance, enable Auto Demotion on each device in the server. This allows the server to poll a device from the poll cycle. The server will then check to see if the device is available before adding it back to the poll cycle.

Note: Each device you create in the server will have a subfolder named `_System`(browse from our free Quick Client to see these `_System` folders). In this folder you will find tags that are related to device settings. To monitor the device status you can watch `_Error` which shows the communication state of the device. A value of 0 is good communications and a value of 1 is an error or bad state. Similarly the `_AutoDemoted` tag can be used to monitor the demotion state. For a complete list of the `_System` tags available in the server see the [KEPServerEX Help file](#).

Multi-threading Scenario 1 (more than 1 PLC)

Since the poll cycle is the amount of time it takes to request a cycle of data from all the devices on one channel, separating the devices so that each one is on its own channel will improve performance. Most drivers in KEPServerEX can support 100 or more channels.

For serial devices you either have to have a serial port for each channel or your devices need to be on a serial-to-Ethernet converter (see “Ethernet Encapsulation” in the KEPServerEX help file).

For Ethernet devices and serial drivers using Ethernet Encapsulation, you will need only your local Ethernet card.

Multi-threading Scenario 2 (1 PLC)

Most modern industrial devices with native Ethernet communications allow multiple Masters to connect to them and poll for data. One way you can often improve communications performance to Ethernet devices is by making several connections to them. In your KEPServerEX project you would define multiple channels, each with a single device connecting to the same physical controller. This means each device in the server project has the same IP address.

Be mindful that if you add too many device connections, you will reach a point where you degrade the performance of your application. The controller you are connected to may have a limited amount of processing time allocated to communications. The more requests the controller has to handle at a single time, the greater the total response turnaround time will be for all connections. Each controller type will be different so it is a trial and error process to see what combination will give you the best results.

3. Optimizing Client Connectivity

Synchronous vs. Asynchronous Operations

When an OPC client application uses Synchronous operations it waits for the return response from the OPC server before performing any other transactions. With Asynchronous operations the request is sent to the server and when completed the server will send a complete callback to the client application. Meanwhile the client application continues performing other transactions. Both call types return all requested items to the client application regardless of whether the data has changed or not.

Most HMI, SCADA, Historian, MES, and ERP applications use Asynchronous operations as the default OPC connectivity method. So typically users of off-the-shelf OPC client-enabled applications do not need to be concerned with choosing between Synchronous and Asynchronous read / write operations. Users who develop custom applications in a programming environment like Visual Basic are typically the ones who should be concerned with this choice. With initial development and with very small or well contained applications Synchronous operations can be easier to implement and commission. Many times users implement Synchronous operations initially, only to “outgrow” them as their applications gets larger and more involved. Redesigning an application to use Asynchronous operations instead of Synchronous can be quite involved for a large application.

Exception-based operations (Unsolicited Asynchronous)

Also referred to as report by exception, callback events, or data change events

With Unsolicited Asynchronous operations, an OPC Client connects to a server and subscribes to data. The server polls the devices at the rate set by the client. If and only if data changes between polls the server sends an event to the client with the new data. For most applications, Unsolicited Asynchronous is the optimal way for OPC clients and servers to connect.

OPC Groups (in OPC clients)

OPC groups are configured in the OPC client. An OPC group is a collection or list of subscribed data. Update rates for the data are set in the OPC group and the group is also where read/write operations are controlled. Each group consumes 2 OS process threads. If you have more than one OPC group in your client application, then only the OPC groups that are using Synchronous operations are **blocked**. For these reasons, client application performance can be improved by designing projects with multiple OPC groups.

4. Configuring DCOM

By default, DCOM Security will be set on any PC running an OPC application. When both the OPC client and server reside on the same PC desktop they will usually connect with no problem. However, when they reside in separate login/user accounts or on different PCs, you will need to modify DCOM security settings to establish OPC client-to-server connectivity.

See Kepware’s DCOM Configuration Guide for instructions on how to set up DCOM Security (the guide is available as a PDF at Kepware.com).

Note: typically DCOM Security is set at the default level which applies it to all applications running on the PC. To apply DCOM Security settings on an individual application basis, users may choose to set DCOM Security at the application level.

5. Running as a Service

Running as an NT Service allows the server to run in the background. This means you do not need to be logged into the PC for remote clients to access the server. We recommend making sure the server is completely configured and all OPC client-to-server connections and basic operations are evaluated prior to running as a Service.

6. Write Optimization

Channel Properties

In KEPServerEX, writes always take priority over read operations, however because writes are performed one at a time, they may need to be queued by the server. To ensure that a large number of writes sent to the server at one time does not adversely impact performance, KEPServerEX sets write performance properties at the Channel level in a project. The default is “Write only last value for all items” with a Duty Cycle of 10 writes per read.

Write only last value tells the server to optimize operations by discarding the previous write in the queue if a new one is received from the client. If you are using momentary buttons to write to Booleans where an On and Off can be sent back-to-back, you will want to change this setting to “Write only last value for Non-Booleans.”

The other default, a duty cycle of 10 writes per read, means that if more than 10 items are queued to be written, then 10 writes will be performed, then a read, then a write again. To modify these defaults, disconnect all OPC client applications from KEPServerEX and then edit the appropriate Channel Properties.

Note: KEPServerEX does not change the value that we are reporting to the client until it can be verified that the data has been changed with a read. The server will promote the written item to be read as soon as possible after the write completes.

Writes for Recipes or Batch processing

If your project requires that you write large amounts of data to the PLC at one instance, you can optimize performance by creating 2 connections to the device (when possible). Then reserve one of the device connections for recipe or batch processing and the other remaining KEPServerEX device for your normal read write operations of your application.