Tech Note 671

Wonderware System Platform: Optimizing I/O Performance

Topic#: 002435
OpsManage09 Session#: TS112
Created: November 2009

Introduction

Optimizing communications between controllers and computers is an expected requirement of ever-expanding systems. Further, as automation solutions become more and more complex, software and hardware units require added functionality, diagnostics capabilities, and extensibility.

Typical control systems are no longer isolated or small. Instead, control systems comprise multiple controllers, PLCs, dedicated client and server PCs, and different database management systems for specific roles such as configuration, alarms, history, or integration with MES-type systems.

Distributed systems require data communications across networks. Process data originates in field devices and must be processed at the controller level, then communicated to other nodes. Specialized I/O programs (drivers) communicate with PLCs and controllers and serve this data to the various applications and computers requiring it.

Operator actions are initiated in visualization nodes, processed in dedicated server nodes, and propagated to other nodes requiring it.

The architecture for Wonderware System Platform divides roles and components so that optimization benefits in I/O performance can be attained at different system layers, such as:

- PLC configuration and communications.
- I/O Drivers (OPC and DA Servers) configuration.
- Execution of Device Integration objects and Application Engines that host them.

Product features like **Advanced Communications Management** and **Engine Scheduler** provide flexibility and improved I/O communications. Different design considerations and recommended architecture best practices can contribute to increased performance when deploying and configuring Device Integration objects, OPC Servers, and DAServers.

Application Versions

- Wonderware System Platform 3.1 or later
- InTouch® 10.1 or later

Simplified Introduction to Wonderware System Platform

An automation project based on Wonderware System Platform is called a Galaxy. All Galaxy components are created from a Development

Station that runs the Integrated Development Environment (IDE) and the configuration is stored in the Galaxy Repository (GR) on top of a Micrsosoft SQL Server. In runtime, the key three components ever present in every project based on System Platform are the **I/O Server**, the **Device Integration Object** (DI Object), and the **Application Engine**.

This *Tech Note* focuses on these elements in order to optimize your I/O performance. The basic principles of these components are presented in this section.

I/O Servers: Wonderware DA Servers

Any supervisory, SCADA, or MES system is rendered useless without data from field devices that indicate the status of the physical world. The I/O Server understands the protocol required to connect to the controllers that have the process data, and serves the data to its clients using industry-standard protocols like SuiteLink™ or OPC.

A PC running the I/O Server, OPC Server, or DAServer is the data source for a System Platform solution. This PC is referred to as the I/O Server node. I/O Server applications translate data from protocols like DDE, SuiteLink or OPC, into vendor-specific protocols to communicate with controllers, PLCs, or RTUs. In their basic role, I/O Servers maintain the list of items that client applications request, then poll or handle data received from field devices, and pass it to subscribed clients.

Wonderware's I/O Servers are called **DAServers**. In addition to their basic role to communicate with the field and serve clients with I/O data, DA Servers provide diagnostics information and centralized management capabilities through the System Management Console (SMC) (Figure 1 below).

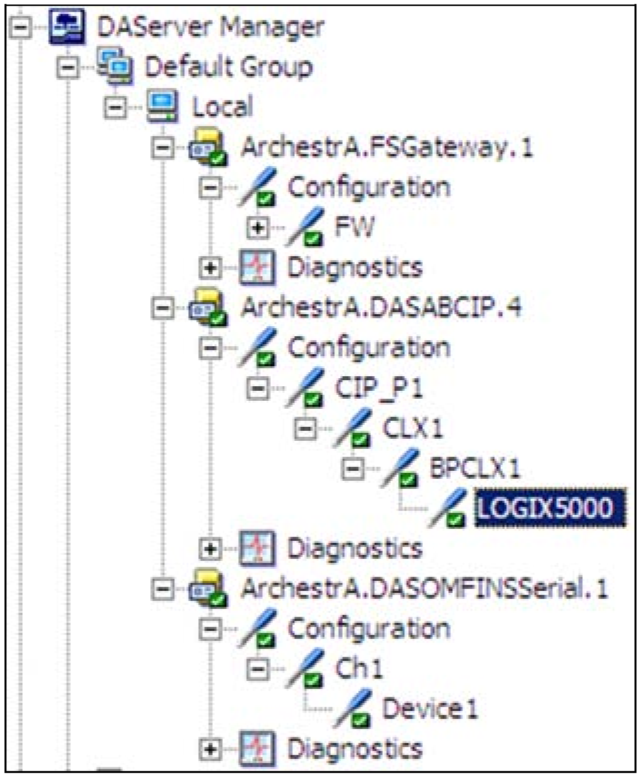


Figure 1: SMC DAServer Manager

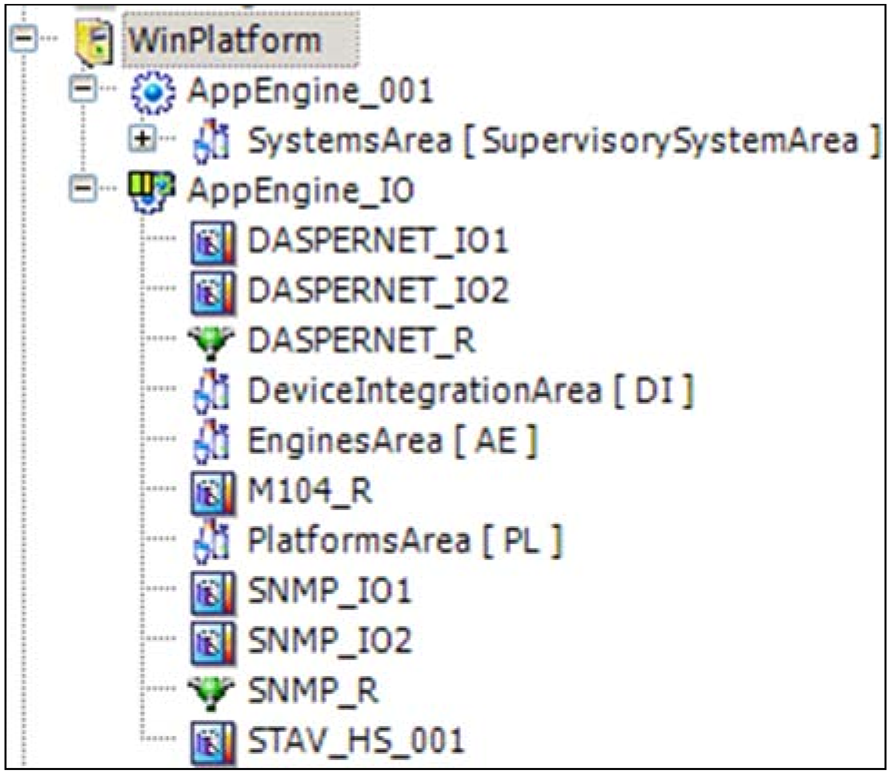
Device Integration Objects (DIO)

In System Platform, **Device Integration** objects are the interfaces between the data sources and the clients that require the data. Data sources are typically I/O Servers, but they can also be HMI packages. Some examples of DI Objects are:

- **$DDESuiteLinkClient**
- **$OPCClient**
- **$InTouchClient** (SuiteLink connection to InTouch node)
- **$iProxy** (connectivity to Intellution's FIX32, iFIX, DMACS).

In addition to the DI Objects, a **$RedundantDIObject** template is available.

Clients are application objects that get their inputs by referencing these DIOs.


```
WinPlatform
  AppEngine_001
    SystemsArea [ SupervisorySystemArea ]
  AppEngine_IO
    DASPERNET_IO1
    DASPERNET_IO2
    DASPERNET_R
    DeviceIntegrationArea [ DI ]
    EnginesArea [ AE ]
    M104_R
    PlatformsArea [ PL ]
    SNMP_IO1
    SNMP_IO2
    SNMP_R
    STAV_HS_001
```
FIGURE 2: DEVICE INTEGRATION OBJECTS

Special Case - Device Integration DA Objects

DAServers can run as standalone applications, or as System Platform Objects that represent the communication with external devices.

For example:

- **DINetwork Object** – Refers to the object that represents the network interface port to the device through the Data Access Server. The object provides diagnostics, and configuration for that specific card.

- **DIDevice Object** – Refers to the object that represents the actual external device (such as a PLC or RTU), which is associated to the DINetwork Object.

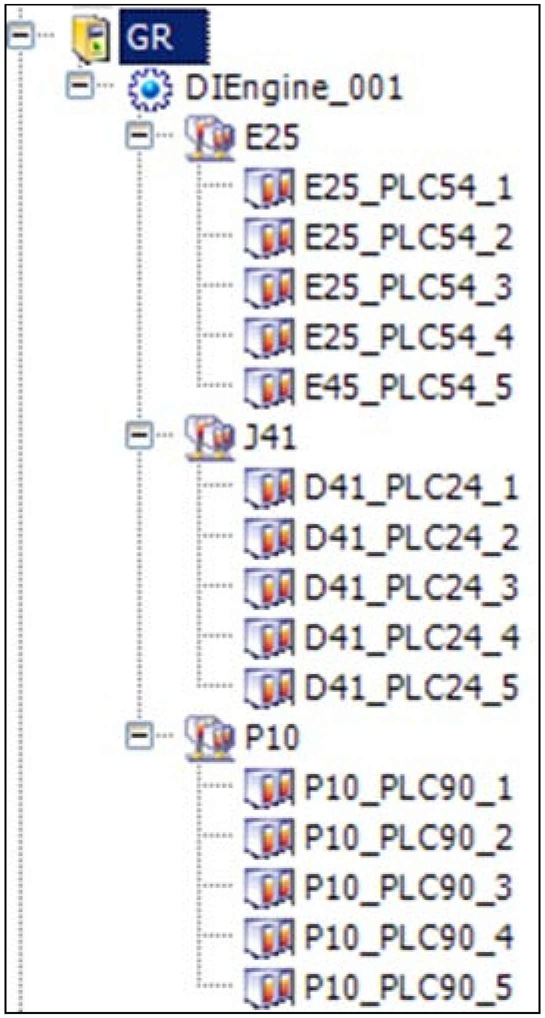


Figure 3: DAS DI Objects

Application Engine

The **AppEngine Object** hosts application objects, device integration objects and areas. Much like a PLC, an Application Engine runs objects cyclically at a specified interval to update inputs from external sources (such as other engines or DI Objects), executes the logic for all application objects, and sends outputs to external sources. Another Engine function is the ability to set up and initialize objects when they are Deployed or Undeployed.

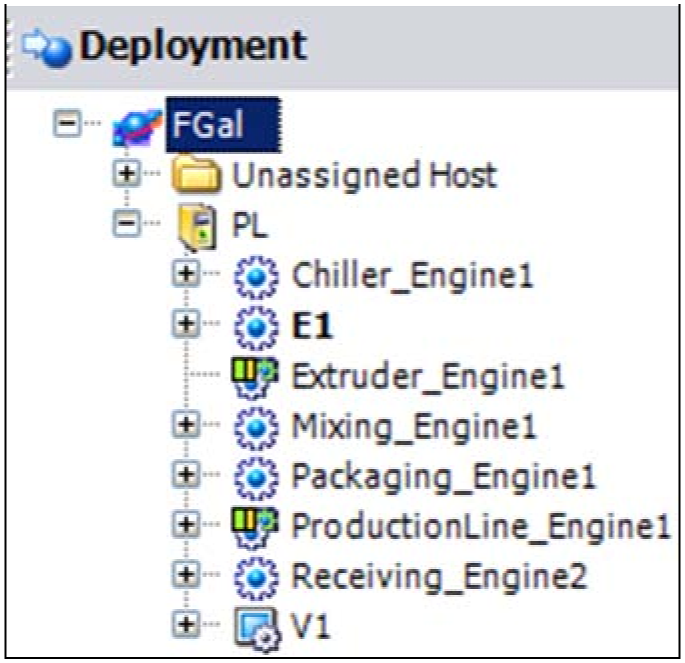


FIGURE 4: APPLICATION ENGINES

Special Case – Platform Object

The **Platform** object is a specialized Engine that represents a computer. The computer is part of the control system using Wonderware System Platform. Only one Platform can be deployed per machine, and these platforms are required by *every* PC running Application Engines or visualization clients (InTouch). Also, a unique Platform must be deployed on the Galaxy Repository (GR) node prior to deploying any other object in the Galaxy.

Optimizing I/O Performance

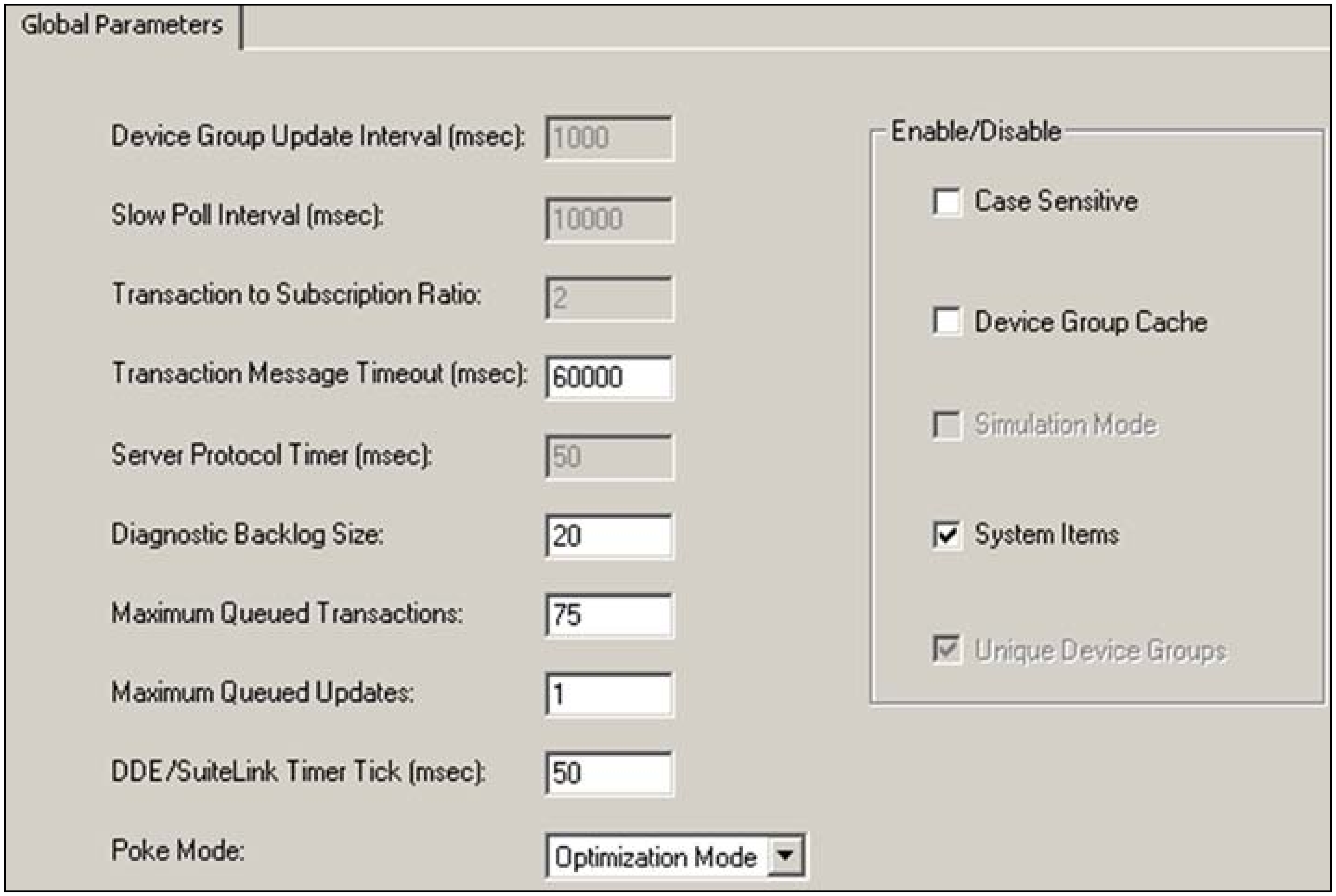
You could implement your automation project with the fastest PLCs, the most powerful servers, and you could spend all your time designing the best-performing solution. However, time and money constraints are always present. The flexibility of System Platform allows running systems to be changed with minimal costs and downtime. This section explains making those changes in order to optimize a system.

The goal for optimization is to maximize your software solution by improving its usability.

You can realize optimization in a project based on System Platform at different levels, and at different stages in the projects' life. You must consider some factors early in the Project life, when the basic architecture is being designed. Other factors can be changed anytime and with little effort. The key three areas for optimization are the **DA Server**, the **Device Integration Object**, and the **Application Engine**. Further, best practices architecting a solution based on System Platform will yield the best I/O performance.

General Settings for DAServer Optimization

Tech Note 424 Working with DA Servers explains global parameters for DAServers and makes recommendations for best performance.


Global Parameters

Device Group Update Interval (msec):	1000
Slow Poll Interval (msec):	10000
Transaction to Subscription Ratio:	2
Transaction Message Timeout (msec):	60000
Server Protocol Timer (msec):	50
Diagnostic Backlog Size:	20
Maximum Queued Transactions:	75
Maximum Queued Updates:	1
DDE/SuiteLink Timer Tick (msec):	50
Poke Mode:	Optimization Mode

Enable/Disable

- [ ] Case Sensitive
- [ ] Device Group Cache
- [ ] Simulation Mode
- [✓] System Items
- [✓] Unique Device Groups

Figure 5: DAServer Manager – Global Parameters

Scan Groups

Creating multiple scan groups-per-PLC is a recommended practice. For example, create at least two scan groups for those items that require a fast update and one for items that are not that critical to monitor, or have infrequent changes. When multiple PLCs and thousands of items must be scanned, it is important to monitor system resources – CPU utilization most likely is a constraint here. Should this be the case, splitting the CPU load across multiple I/O Server nodes is the recommended solution.

Another common practice is to stagger the multiple scan groups that have the same polling rate and use instead prime numbers around the desired polling rate.

Unsolicited Messages

For exceptional conditions in the process or in the PLC, it is also possible for some PLCs to send broadcast or unsolicited messages with data, *without* the I/O Server polling for them. For example, **Tech Note 595** shows how to setup Wonderware Application Server to receive unsolicited messages from Allen Bradley Controllogix PLCs.

DI Object Optimization and DI Object Type Usage: DDE/SuiteLink vs. OPC

- For local communications DDE is the oldest technology available, but it has low overhead and it can be faster than SuiteLink in some cases with low topic count. On the other hand, DDE is *not* recommended in Vista and Windows 2008, and NET DDE is not supported in the Vista and later OSs.
- SuiteLink includes some limitations that are not present when using OPC. For example, SuiteLink supports 4 data types (Integer, float, boolean, string) but not long integer or long float data types.
- OPC is the industry standard, but setting up communications over network may present a challenge for security and DCOM configuration.

DI Objects vs Standalone DAServers

DI Objects are deployed, as opposed to DAServers that must be installed – deployment translates into flexibility and easier change management.

Tuning the ApplicationEngine

You can optimize Engine performance using the following guidelines.

Adjust the Scan Period

Engine.Scheduler.ScanPeriod is the parameter in the Application Engine object that sets the execution speed (in mSec); The faster the period, the faster objects will execute and update. However, the scan loads the CPU and takes time that other windows processes compete for, such as the visualization clients or I/O Servers. You must consider this setting carefully, and take into account a steady-state situation after all objects have been deployed.

Best Practice

The goal is to maximize engine performance during steady state (not in deployment time) with the system resources available.

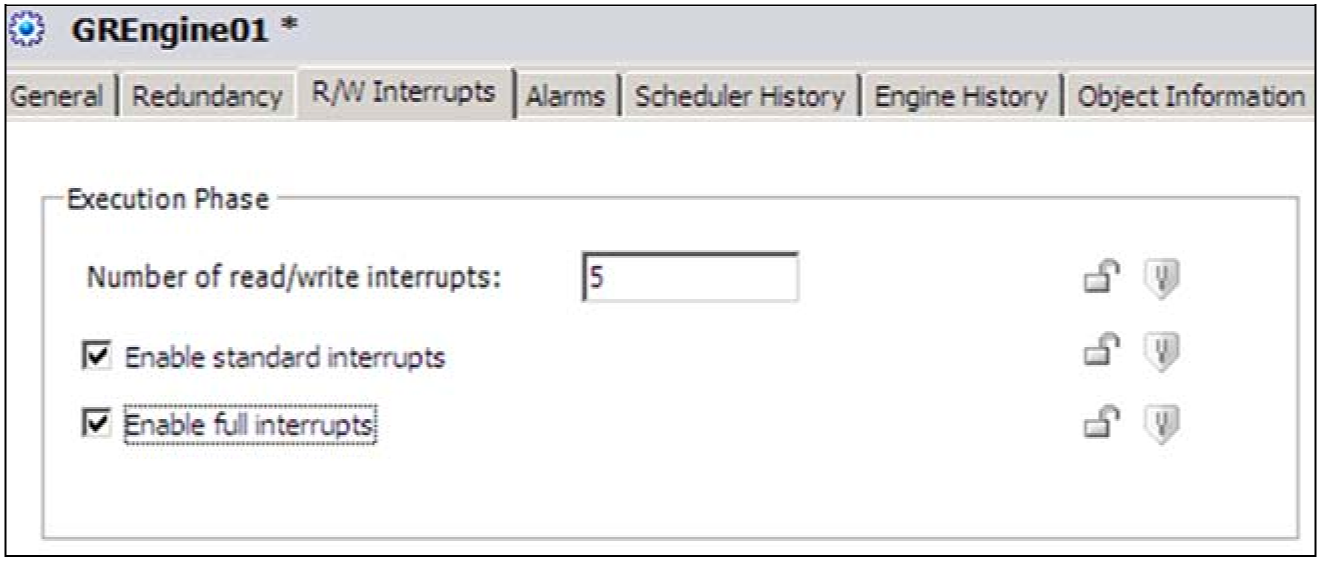
- Overall average CPU utilization for the computer should not exceed 30 to 40 % in a system with load balancing or 50 to 60 % without load balancing
- **Scheduler.ExecutionTimeAvg** should not exceed 30 to 40 % of the Engines Scan period; Time idle average at around 50% of the engine scan period
  - Scheduler.ScanOverrunsCnt may occur during initialization or deployment but should be a rare event in steady state, this count should never exceed .05% of Scheduler.ScanCyclesCnt
  - Consecutive scan overruns zero, excluding deployment overruns

Optimizing the Engine Scheduler

The scheduler is one of the building blocks of an Application Engine or Platform Engine. It controls timing, object execution, and external communications. The tasks performed by the scheduler follow a specific pattern that is similar to what a PLC does; read inputs, execute objects alphabetically and according to their area assignment, write outputs (external data), and save state (checkpointing). By default, the engine scheduler updates DI Objects multiple times during the scan cycle. This is in itself an enhancement in System Platform introduced in version 3.0; previous releases would scan DI objects only once, like all other objects hosted by the engine. A Read/Write interrupt occurs when the application engine updates DI Objects during a scan cycle.

The scheduler can be configured for standard interrupts (enabled by default) or full interrupts. Interrupts are nothing but DI object updates that take place during the execution of application objects. As interrupts take place, DI Objects send any new received write requests to the I/O server, read any new updates from the I/O server, and publish all new data to visualization engines. Standard interrupts occur during while all objects in the engine are being executed, and the number is configurable.

Full interrupts extend the functionality of standard interrupts.


GREngine01 *

General | Redundancy | R/W Interrupts | Alarms | Scheduler History | Engine History | Object Information

Execution Phase

Number of read/write interrupts: 5

☑ Enable standard interrupts

☑ Enable full interrupts

FIGURE 6: ENGINE SCHEDULER AND CONFIGURATION FOR READ / WRITE INTERRUPTS

Standard Interrupts

You can enable/disable Standard interrupts per Application Engine. This parameter can also be set during runtime from Object Viewer or an InTouch client. The attribute is called **EngineName.Scheduler.ExecInterruptsStandardEnable**
and the number of interrupts during the Execute Phase can be also be configured in runtime by changing the **EngineName.Scheduler.ExecInterrupts** attribute.

The valid range **0 to 256** with a default value of 5.

Full Interrupts

You can enable this mode per Application Engine *only* if Standard Interrupts have been enabled. This parameter can also be set during runtime from Object Viewer or an InTouch client. The attribute is called **EngineName.Scheduler.ExecInterruptsFullEnable**.

These three attributes can be used to adjust the frequency and behavior of RW Interrupts that occur during an Engine's Execute phase, and they can be adjusted both at Configuration time and Runtime.

Best Practice

Standard Interrupts are enabled by default when a new ApplicationEngine is created. Migration from Industrial Application Server 2.1 keeps interrupts disabled in order to maintain existing behavior. After migration of earlier galaxies in version 2.x, review application needs and determine whether this option should be enabled to accelerate. In most cases Standard Mode will make improvements, since these settings only really matter when the ApplicationEngine is at least moderately loaded.

1. **EngineName.Scheduler.ExecInterrupts** – defaults to 5; do not set higher than 20 since too high settings will not appreciably help performance and may cost CPU.
2. **EngineName.Scheduler.ExecInterruptsStandardEnable** – defaults to true; should be set to true for most applications; will improve HMI response times
3. **EngineName.Scheduler.ExecInterruptsFullEnable** – defaults to false; will perform slightly better than Standard Interrupts but usually not noticeable; should be set to false for most applications.

The effects of these interrupts disappear on an engine that has greater than 90% idle time; if engine is 30% or more busy (ExecutionTimeAvg/ScanPeriod), then these settings will make a difference.

For Supervisory Control applications in which synchronization of data “snapshots” is important, Full Mode is not recommended. For example, different ApplicationObjects on same engine may see different value updates (non-synchronized) from the field or from other engines during a single execution phase.

Interrupts mean more work during middle of Execution phase, thus they may slow the ExecutionPhase. It is not recommended to set ExecInterrupts too high (e.g. > 100); monitor the scan overruns attribute as interrupts are adjusted. In addition to higher CPU utilization, there is the potential for more network traffic as DI objects may send smaller packets to servers, resulting in DA and IO Servers sending smaller write packets to PLCs; this is particularly important when polling analog values that could change very frequently in the field. Also, visualization nodes may get smaller packets more frequently.

Runtime ApplicationEngine Tuning Tips

- Tune one ApplicationEngine at a time.
- Make sure your engine is fully loaded, as it will be in your production environment. Tuning is useless on an unloaded engine.
- You have three "knobs" that you can adjust per Engine. These knobs are the 3 attributes in the Engine Scheduler. Place each on the ObjectViewer.
  - ExecInterrupts
  - ExecInterruptsStandardEnable
  - ExecInterruptsFullEnable
- In addition, monitor the following attributes within the Engine to determine the impact the above settings are having on engine performance. The goal is to keep these two parameters at a minimum while achieving the required RW performance from the HMI.
  - Scheduler.ExecutionTimeAvg
  - Scheduler.ExecInterruptsTimeAvg
- Start out at the default settings of **ExecInterruptsStandardEnable = true** and **ExecInterruptsFullEnable = false**.
- Measure performance of InTouch updates and writes using a test point that writes to the controller. Include both write-side and read-side timing capability. Usually, measuring the combined time of a complete round-trip is most effective.

Adjust ExecInterrupts to various settings and measure average write and read time. Suggested settings for ExecInterrupts are **5**, **10**, **15**, and **20**.

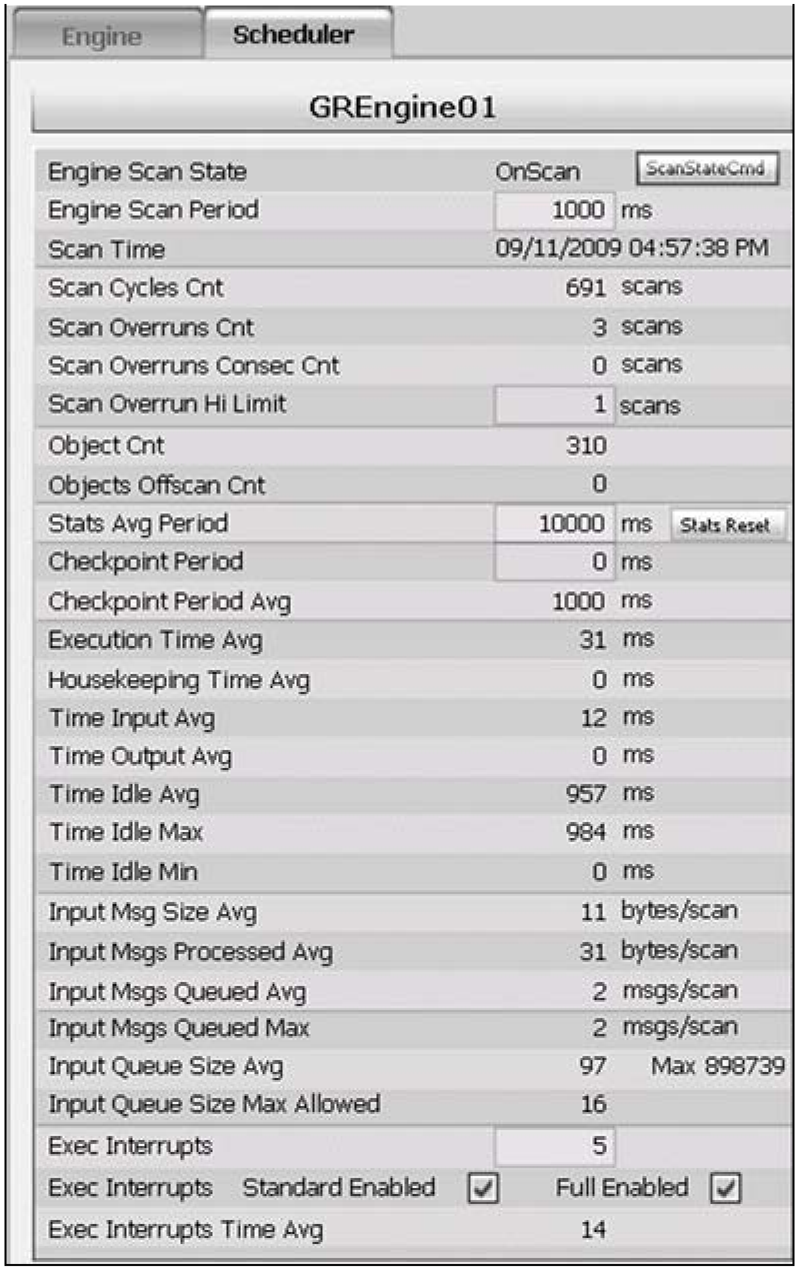
- Select an optimal setting for **ExecInterrupts** that provides the best combination of I/O performance and engine loading (ExecutionTimeAvg).
- Follow by enabling Full interrupts with **ExecInterruptsFullEnable**.

  Note that for *some* applications, this may not be an acceptable setting, since it means that a single I/O point could have different values in a scan.

  - Repeat tests and see if it makes a measurable difference.
  - If no measurable difference, then disable Full interrupts.

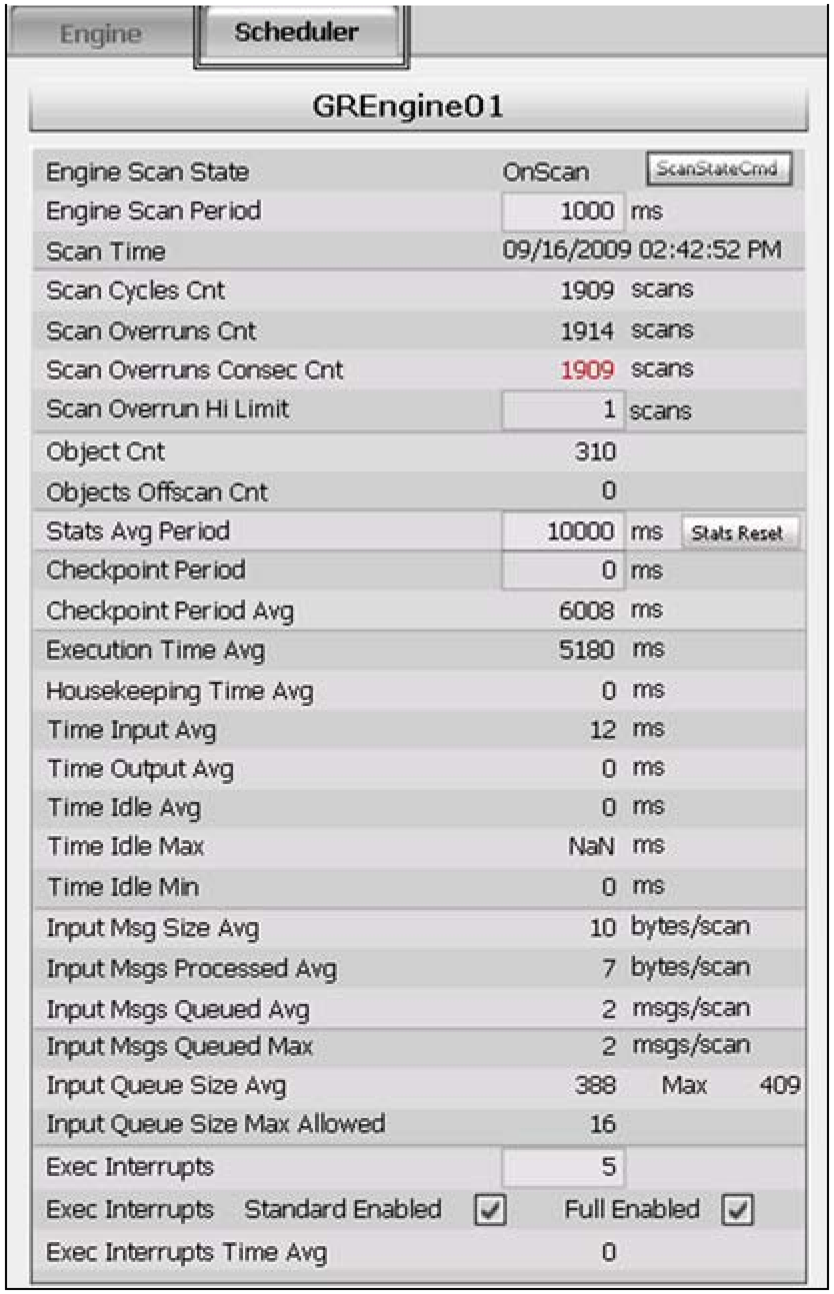
Monitoring the Engine Status

System Platform includes multiple attributes you can use to report the health status for an Application Engine.



GREngine01

Engine Scan State	OnScan	ScanStateCmd	
Engine Scan Period	1000	ms	
Scan Time	09/11/2009 04:57:38 PM		
Scan Cycles Cnt	691	scans	
Scan Overruns Cnt	3	scans	
Scan Overruns Consec Cnt	0	scans	
Scan Overrun Hi Limit	1	scans	
Object Cnt	310		
Objects Offscan Cnt	0		
Stats Avg Period	10000	ms	Stats Reset
Checkpoint Period	0	ms	
Checkpoint Period Avg	1000	ms	
Execution Time Avg	31	ms	
Housekeeping Time Avg	0	ms	
Time Input Avg	12	ms	
Time Output Avg	0	ms	
Time Idle Avg	957	ms	
Time Idle Max	984	ms	
Time Idle Min	0	ms	
Input Msg Size Avg	11	bytes/scan	
Input Msgs Processed Avg	31	bytes/scan	
Input Msgs Queued Avg	2	msgs/scan	
Input Msgs Queued Max	2	msgs/scan	
Input Queue Size Avg	97	Max 898739	
Input Queue Size Max Allowed	16		
Exec Interrupts	5		
Exec Interrupts	Standard Enabled ✓	Full Enabled ✓	
Exec Interrupts Time Avg	14		



GREngine01

Engine Scan State	OnScan	ScanStateCmd	
Engine Scan Period	1000	ms	
Scan Time	09/16/2009 02:42:52 PM		
Scan Cycles Cnt	1909	scans	
Scan Overruns Cnt	1914	scans	
Scan Overruns Consec Cnt	1909	scans	
Scan Overrun Hi Limit	1	scans	
Object Cnt	310		
Objects Offscan Cnt	0		
Stats Avg Period	10000	ms	Stats Reset
Checkpoint Period	0	ms	
Checkpoint Period Avg	6008	ms	
Execution Time Avg	5180	ms	
Housekeeping Time Avg	0	ms	
Time Input Avg	12	ms	
Time Output Avg	0	ms	
Time Idle Avg	0	ms	
Time Idle Max	NaN	ms	
Time Idle Min	0	ms	
Input Msg Size Avg	10	bytes/scan	
Input Msgs Processed Avg	7	bytes/scan	
Input Msgs Queued Avg	2	msgs/scan	
Input Msgs Queued Max	2	msgs/scan	
Input Queue Size Avg	388	Max	409
Input Queue Size Max Allowed	16		
Exec Interrupts	5		
Exec Interrupts	Standard Enabled ✓	Full Enabled ✓	
Exec Interrupts Time Avg	0		

Figure 7: Faceplates Showing a Normal and Overloaded Engine

System-Wide Optimizations

This section includes guidelines for system-wide Advanced Communications management.

Summary

Client applications subscribe to DA Servers via Device Integration objects and request data from the field devices. The initial release of Industrial Application Server had a different way to handle I/O Data than the current System Platform version. In that release all I/O references in the Galaxy would be activated and subscribed to by the corresponding I/O Servers that served these data.

System Platform 3.1 introduced the **Advanced Communications** functionality, which is the possibility to activate I/O references on demand only. This feature reduces network traffic, reduces system utilization on the I/O Server node, and increases performance on the node running application objects as data is only delivered if subscribed.

System Platform 3.1 Feature Summary

Scan On Demand is a feature where an attribute whose value is not currently being accessed by any client is "suspended." A suspended attribute is a dynamic attribute on a DI object. When it is not scanned by its DAServer, CPU usage in the DAServer is reduced. Changes in suspended attributes' values are not published, thus reducing network traffic. Note that a Historized or Alarmed attribute is *never* suspended.

Tech Note 628 Advanced Communication Management for Wonderware Application Server provides details on Advanced Communications and Scan On Demand. Advanced Communications greatly benefits users of System Platform on Supervisory or SCADA-based projects over wide-area networks, and to a less extent applies to large Galaxy users with high numbers of Historizated and Alarm attributes.

Summary Architecture for Best Performance

Among the most important considerations in the design of a project with Wonderware System Platform is the I/O Count and the number of PCs (platforms). These numbers determine the size of the license that you need to run the project.

There usually not much flexibility in reducing the number of computers and I/O count, but you can use Wonderware Application Server to design a dynamic and flexible system in regards to the number of Application Engines, the number of objects-per-engine, the I/O attribute count-per-application object, and the distribution of Device Integration objects.

Use Application Objects to model your physical production environment. For example, multiple Inputs and Outputs are part of a compressor or a mixer, and the **$Compressor** template object should reflect the actual signals from that equipment. Therefore, the best object design practice aggregates Inputs and Outputs under the same template.

Example

Assuming a hypothetical Galaxy with 3000 I/O points and 200 Application Objects with 15 I/O signals each (5 discrete, 5 Integer, 5 float), let's compare different architectures in terms of performance for I/O.

Number of Engines

Componentization and multithreading are part of the design in System Platform, which provides better performance in terms of CPU utilization. Every Application Engine runs as a distinct process (separate executable) called **aaEngine.exe**. Multiple engine processes will take advantage of machines running on multi-core processors. Performance tests conducted with a single CPU with two engines hosting 100 objects each run at 43% CPU utilization. The same Galaxy on a dual CPU dropped the average CPU utilization to just 7%.

R/W Interrupts

Enabling R/W interrupts speeds up data collection as previously explained in detail.

Advanced Communications

With Advanced Communications and Scan On Demand enabled, the CPU usage is reduced and optimized, since only I/O points that are needed for alarming, scripting, historization, or visualization can be activated while all others remain inactive (suspended). Please refer to **Tech Note 628** for details.

Location of DIOs Relative to Application Objects

Device Integration objects can be hosted by the same engine that hosts Application Objects, or they can run on a separate engine. R/W Interrupts optimize communications in both cases, as discussed in the Scheduler section. Keeping DI Objects on their own engine is good practice, in case the Engine must be stopped/Restarted or Undeployed/Redeployed.

F. Gonzalez

Tech Notes are published occasionally by Wonderware Technical Support. Publisher: Invensys Systems, Inc., 26561 Rancho Parkway South, Lake Forest, CA 92630. There is also technical information on our software products at **Wonderware Technical Support.**

For technical support questions, send an e-mail to **support@wonderware.com**.

Back to top