

# Real-Time Extensions to OPC

The use of OPC (OLE for Process Control) as an industry-standard mechanism for access to hardware I/O devices is likely to become the preferred middleware mechanism for the connection between HMI/SCADA software and its associated hardware system elements. Many of these I/O devices require real-time capabilities that are specific to the hardware and essential to proper operation of the high-level HMI/SCADA application. This paper describes the incorporation of hard real-time concepts into standard OPC servers, as implemented by RadiSys, utilizing their INtime real-time extension for Microsoft Windows NT.

## INTRODUCTION

### OPC means OLE for process control

The OPC Foundation, a non-profit standards group, has established a set of standard OLE/COM interface protocols intended to promote greater interoperability between automation and control applications, I/O devices, and business systems used by the manufacturing and process control industry. The standards developed by the OPC Foundation are built directly on top of Microsoft's OLE/COM technology. OPC servers (as implementations are commonly called) are implementations of OLE/COM servers with an industrial-control-specific interface.

The OPC specification defines a standard set of OLE/COM interfaces to be used for building an OPC server. OPC servers—which are device-specific implementations of standard objects, methods, and properties—are used by OPC clients as sources of real-time information. For example, distributed process control systems, programmable logic controllers, smart field devices, and analyzers all represent possible OPC servers that can communicate device-specific information to OLE/COM-compliant OPC client applications. Human Machine Interface applications (HMI), Supervisory, Control, and Data Acquisition applications (SCADA), OLE-compliant databases, and off-the-shelf spreadsheets all represent candidate OPC clients.

For detailed information regarding the OPC

Foundation's activities and specifications please see reference [1] and the OPC Foundation's web site at <http://www.opcfoundation.org/>.

### OPC and the Enterprise

The following figure is a high-level view of an OPC system that incorporates three basic levels in the manufacturing hierarchy, which OPC unifies: field management, process management, and business management. In order to maintain a competitive position in the marketplace, manufacturers need real-time access to data from their plant floor, which they must then integrate directly into their existing business systems. For efficiency, manufacturers need to utilize off the shelf tools (HMI and SCADA packages, databases, spreadsheets, etc.) in order to assemble a management and control system that meets their needs. The key is an open and effective communication architecture that concentrates on how data is accessed, not on the type of the data.

The devices on the bottom of this figure are examples of dedicated hardware and computers involved in the field management of an OPC system. Intelligent field devices can provide a wealth of information to the manufacturing system. Information regarding the health of an I/O device, its configuration parameters, its current operating state, etc. are all data that must be presented to a user or an application in a consistent manner. OPC data can be used for monitoring and/or directly controlling these elements of the system.

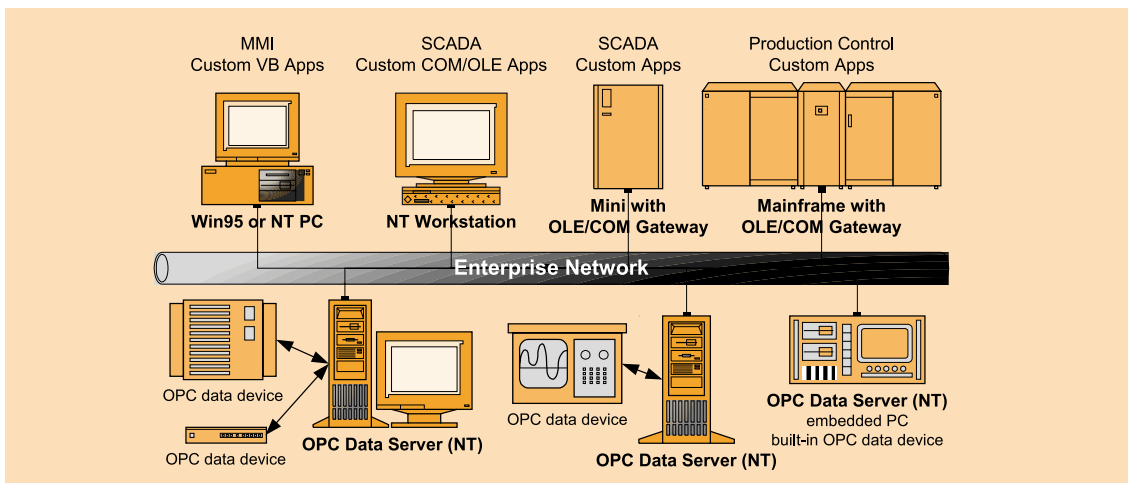


Figure 1. OPC System Overview

Ad MOTOROLA CG

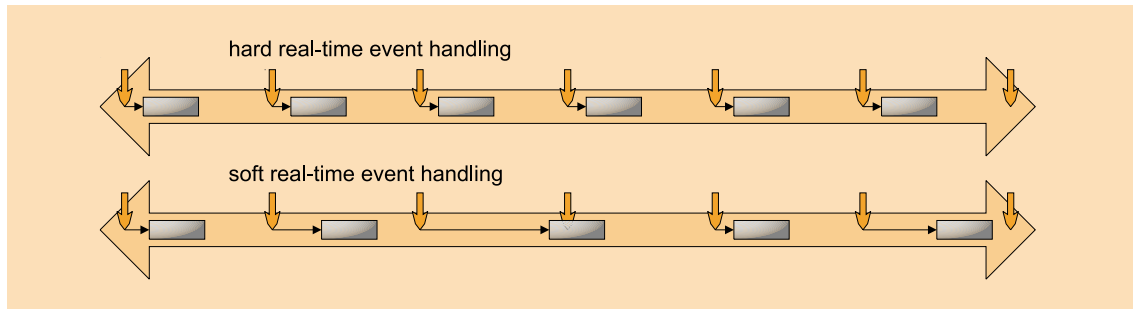


Figure 2. Soft Real-Time vs. Hard Real-Time Event Handling for a Periodic Task

The two Win32 computing nodes in the upper left of the figure represent process management nodes in an OPC-based manufacturing system. The installation of Distributed Control Systems (DCS) and Supervisory, Control, and Data Acquisition Systems (SCADA) to monitor and control manufacturing processes makes the need for supplying data electronically, where previously it was gathered manually, increasingly important for implementation of efficient manufacturing systems.

Legacy software systems, such as those represented by the large compute nodes in the upper right corner of this figure, are located at the business management level of an OPC-based manufacturing system. The benefits gained by incorporating OPC into this level of the manufacturing system are significant, especially where timeliness of the data is an important factor in the overall management process. Utilizing OPC at this level allows one to integrate information collected from the process and field management systems directly into the business management system, resulting in real-time data being used to monitor and control the financial aspects of the manufacturing process.

Providing real-time data in a consistent manner to client applications and users, regardless of that data's location in the manufacturing hierarchy, minimizes the effort required to build a completely integrated system. This summarizes the goal of OPC, to manage the efficient flow of automation and control information, regardless of its source or destination, throughout the manufacturing system.

## REAL-TIME OPC SERVERS

OPC servers, by definition, are producers and consumers of real-time data. Which means they generate data for use by OPC clients or employ data created by OPC clients. The exact nature of the data is dependent on the OPC server, its associated I/O device(s), and the function of the OPC client application. Data is transferred between an OPC server and client via an OLE "interface" defined by the OPC specification.

### Definition of real-time

Real-time systems can be generalized into two fundamental categories: hard real-time and soft real-time. A hard real-time system must, without fail, operate on its data and events within a predetermined time window. In contrast, a soft real-time system will, on average, operate on its data or events in a timely fashion but will not be considered to fail if it misses some deadlines. Based on these definitions, Windows NT is not a hard real-time operating system. [2]

The following figure illustrates the difference between a hard and soft real-time system. This figure assumes the real-time system utilizes a regular, periodic event (i.e., a timer event or a timer interrupt). An example of such a system is a digital PID controller. The vertical arrows represent the regular, periodic timer events, which are typical of digital PID controllers; it is the control-loop's desired sampling instant. The small horizontal bars represent the processing time associated with the PID task, or real-time task. The horizontal arrows that lead from the sampling instant to the real-time task are the event latency, or the delay from the time the task should begin execution to the time it actually begins execution.

In hard real-time systems every event is serviced, and the task associated with that event is started and finished within a bounded period of time. In a soft real-time system some events may be dropped (i.e., never serviced) and the time required to service the event is not guaranteed to be bounded. A PID controller is an example of a real-time task that depends directly on hard real-time conditions being met to insure that the quality of its control is maintained. Without a guarantee of a hard real-time environment such a controller may become either unstable or exhibit an unacceptable quality of service.

### Hard real-time OPC servers

The OPC interface is intended to hide the intricacies of an abstract functional unit (e.g., a device or real-time process). By combining OPC with a hard real-time extension, such as RadiSys' INtime extension for Windows NT, one can create an abstract interface for real-time processes, like the PID controller example, without having to understand the details of moving data between the Windows NT soft real-time environment and the INtime hard real-time environment. Further, direct deterministic control of I/O devices can be implemented in the hard real-time environment without sacrificing the requirement of using a standard interface to access that real-time device's data and control parameters. In other words, adding an OPC interface to your hard real-time processes means you can share real-time data with third-party applications that might otherwise have required intimate knowledge of the data structures and algorithms inside in your real-time process.

A real-time data source is a hard real-time process running on the INtime kernel which produces data and exports it to an OPC client as an OPC server, by presenting the data as an OPC "item." Likewise, a real-time

data sink is an INtime process, which accepts data from an OPC client by receiving data via an OPC "item." The INtime environment supports the development of a special class of OPC-friendly real-time processes, called real-time OPC agents, which can act as real-time data sinks and sources. This concept of a real-time OPC agent is illustrated in the following figure.

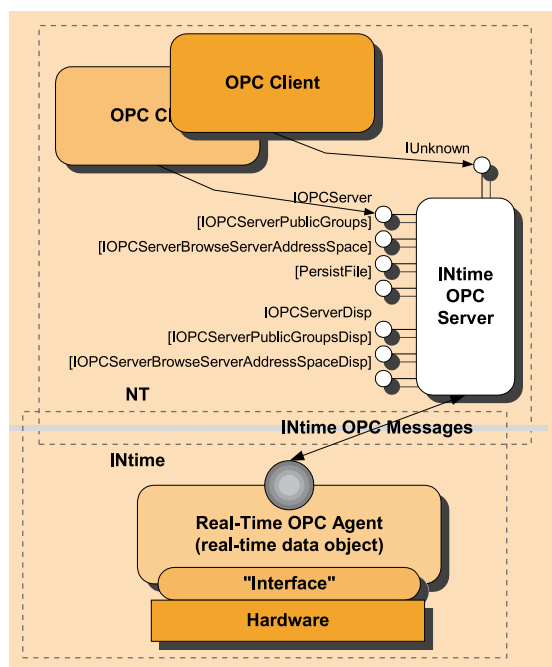


Figure 3. Real-Time OPC Server Block Diagram

A special OPC proxy, supplied by RadiSys for use with real-time INtime OPC agents, transfers messages between standard OPC clients and real-time OPC agents, the real-time component of an INtime OPC server. Developers of real-time OPC agents are not bothered with the complexities of building a COM or OLE object, since the OPC proxy provides that framework. Developers need only concentrate on implementing their real-time task as real-time agents, which are simply special versions of real-time C programs.

### Hard real-time enables OPC server determinism

Consider again the example of a PID controller implemented as an OPC server object in the hard real-time environment. The real-time agent which implements a PID controller might be designed to accept five input parameters, or OPC items, which represent  $k_p$ ,  $k_i$ ,  $k_d$ , (the proportional, integral, and derivative gains) a setpoint, and the PID loop's sampling rate ( $T_s$ ). This OPC-based PID controller might also generate data, as OPC items, which can be consumed by an OPC client. For example the data generated by such an OPC server could be the measured value of the controlled variable and the controller's error (the difference between the desired value of the controlled variable, the setpoint, and the measured value of the controlled variable).

Such an OPC-based PID controller could also be implemented within the standard Windows NT environment, but the quality and consistency of the con-

troller's performance will be subject to the soft real-time nature of the Windows NT operating system, resulting in, at best, imprecise operation. For some PID applications the imprecise operation of a soft real-time controller might be sufficient. For many systems, however, the precision associated with a hard real-time environment that can only be obtained with a real-time extension to Windows NT is required. By utilizing OPC to build the interface to the PID controller one can develop high-level supervisory software that remains unchanged, regardless of whether it uses an imprecise Windows NT-only PID controller or a precise hard real-time-enabled PID controller. In either case the high-level supervisory OPC client software uses only one interface to exchange data with the low-level OPC server-based PID controller.

### INSIDE A REAL-TIME OPC SERVER

A comparison of the data obtained by a hard real-time OPC sampler and that obtained by a soft real-time OPC sampler is shown in the following figures (these are simplified, theoretical servers for the purpose of discussion). For this example assume that an OPC server is being utilized to monitor a chemical reaction. Data will be collected via an OPC server and later analyzed by inspecting the rate of change of the reaction curve. In this case, the reaction curve is expected to follow a standard exponential decay curve.

Figure 4 overlays two sets of data sampled by two OPC servers, one hard real-time and one soft real-time. The hard real-time OPC server took the "precise" set of data points. Its samples are aligned nearly perfectly with each desired sampling instant. The "imprecise" set of data points are not aligned perfectly with the desired sampling time, they are delayed from the desired sampling instant by a variable amount, which is a function of the sum of all activities on the computer at the time the sampling instant was registered. In this figure the x-axis represents the actual moment in time at which a data sample was taken and the y-axis represents the value of the exponential decay at that moment in time. Note that the "imprecisely sampled data" does represent valid points along the exponential curve, they simply are not taken at exactly the desired moments in time, due to soft real-time delays.

Figure 5 is a plot of the sampled data from figure 4. It illustrates what happens to the OPC client software's perception of the reaction curve when it performs its analysis. The shape of the reaction curve "changes" due to the imprecise sampling of the soft real-time OPC server. The precise reaction curve is identical to that in figure 4 because the values obtained by the hard real-time OPC server were taken at exactly the expected moments in time. The "change" in the shape of the curve results because we have plotted the data obtained by the imprecise sampler at the expected sampling times.

Performing a least squares fit on the precise and imprecise data results, in order to obtain the exponential decay constant, results in the curves shown in figure 6. The slope of the precise curve represents the actual decay constant for this particular reaction. The

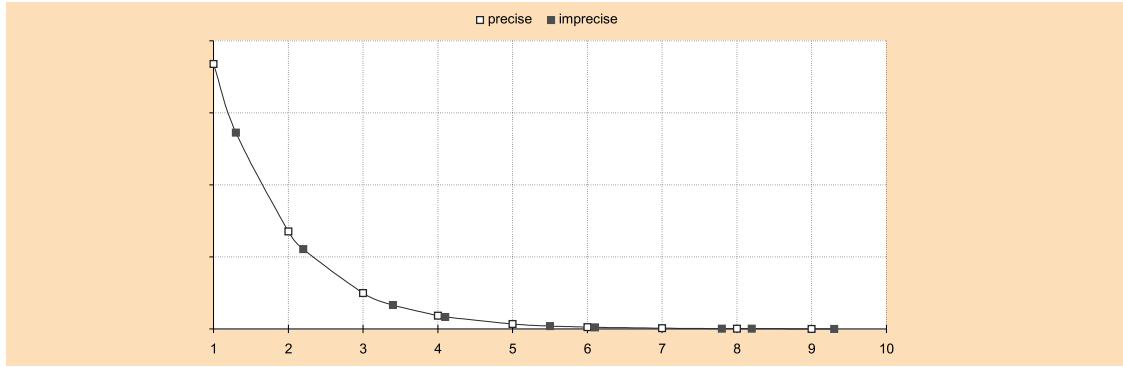


Figure 4. Precise and Imprecise Sampled Data Points

imprecise data, which resulted because of the variable delays relative to each sampling instant, has the wrong decay constant.

Obviously, the impact such an error has on the system being monitored or controlled depends on the sensitivity of that system to the error in the calculation that resulted from variations (delays) from the desired sampling instants. Those systems, which cannot withstand the errors that result from a variation in the timing inherent in many soft real-time systems, are better suited to using hard real-time OPC servers.

Fortunately, for those who require a hard real-time OPC server it is not necessary to understand how to build an OPC server, only how to write a hard real-time program in the C or C++ language.

### Building a real-time OPC agent

Figure 3 is a schematic representation of how one builds a hard real-time OPC server (a.k.a. an RT-OPC server) using the RadiSys INtime real-time extension for Windows NT. The yellow box named "Real-Time OPC Agent" is a real-time C language program that must be written to implement those actions associated with the OPC server's data items. In other words, the real-time code in the real-time OPC agent component is the actualization of the methods behind the OPC data items.

That part of the OPC server which sits above the "OPC Messages" line in figure 3 is pre-built, generic code that is supplied with the RT-OPC development tool. It can be thought of as a RT-OPC proxy because it presents a standard COM/OLE interface to the OPC client program on behalf of the real-time OPC agent, which

is a real-time C, or C++ program. The RT-OPC proxy generates a standard set of messages, which are implemented as well-defined methods in the real-time environment. The names of the OPC data items are defined by the real-time developer and are "discovered" by the RT-OPC proxy, also using a standard set of messages that correspond to a set of well-defined methods.

Thus, the process of implementing a real-time OPC server consists of two basic steps: define the data items to be presented by the RT-OPC server and write the methods that correspond to the messages that can be generated by the RT-OPC proxy, which represent actions or operations on those data items.

### Defining Real-Time OPC Data Items

If we take the data sampling OPC server example from the previous section and define some data items for it we might come up with the following list: `sampler.A`, `sampler.B`, `sampler.C`, `sampler.START`, `sampler.STOP`, and `sampler.RATE`. Data items `sampler.A`, `B`, and `C` are read-only arrays that represent three analog to digital converters, which we can read; we used only one in our previous example. The other three data items, `sampler.START`, `STOP`, and `RATE`, do not represent hardware but the sampling rate and time span during which our data sampling OPC server actually reads the analog to digital converters.

This design requires that the OPC client first set a start time, stop time, and sample rate (by writing to the `sampler.START`, `STOP`, and `RATE` data items) and then read the results as an array. In order to insure proper timing of the sampled data the real-time OPC server will read

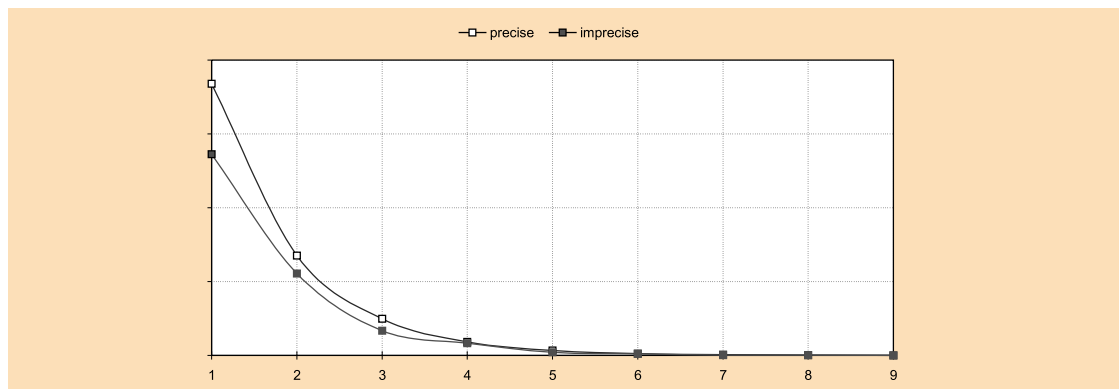


Figure 5. Perceived Reaction Curve due to Imprecise Sampling

Ad CETIA

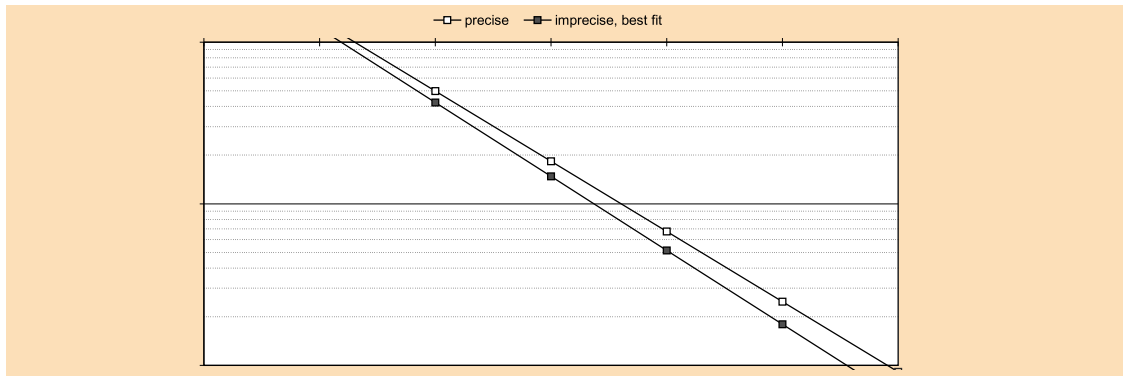


Figure 6. Variation of Calculated Curve Parameters due to Imperfect Sampling

the analog channels at the rate specified and buffer that data for the final read from the OPC client. For this simple example it may be necessary to return vendor-specific error codes to indicate when a read is attempted before all the data has been collected.

### Implementing Real-Time OPC Methods

The most basic actions to be performed on a set of OPC data items are simply to read and write to them. However, in our example, writing to one data item can result in affecting the behavior of other data items. In this example, one must write to the sampler.START, STOP, and RATE data items as an initialization sequence before reading the sampled data via the sampler.A, B, and C data items.

The basic code one must write, the C/C++ real-time code, is typically a switch statement inside a well-defined function. In the following example the handle that was assigned to each data item is used to switch to the appropriate internal variable for a read.

```
#1: OPCRead( pdata_item_handle ) {
#2:   switch( *pdata_item_handle ) {
#3:     case ITEM_A_HANDLE :
#4:       // report sampler.A
       array buffer
#5:       break ;
#6:     case ITEM_B_HANDLE :
#7:       // report sampler.B
       array buffer
#8:       break ;
#9:     case ITEM_C_HANDLE :
#10:      // report sampler.C
       array buffer
#11:      break ;
#12:     default :
#13:       // return an error
       code
#14:   }
#15: }
```

Similar functions are defined for writes to data items and for browsing the list of available data items for presentation to the OPC client.

In a more complex example, such as a PID controller, additional real-time code would be written that actually implements the PID control loop. These well-defined functions would still be required as the interface into

that PID controller, and would reference internal variables in the PID controller, transforming them into a format that is presentable to the OPC interface.

## CONCLUSION

By combining the industry-standard OPC interface with a hard real-time extension for Windows NT, such as RadiSys' INtime product, it is possible to create deterministic OPC objects that communicate directly with off-the-shelf, OPC-compliant automation and control software, like HMI and SCADA applications. OPC servers built from C language real-time agents are able to interact intimately with monitoring and control applications on the same Windows NT system, or via a network to a remote Win32 system. Uniting OPC and hard real-time enables one to build deterministic data and control objects, meaning it is possible to build reliable, hard real-time data servers for high-quality control without the requirement that third-party applications must understand the details of the underlying low-level data structures and control algorithms. ■

## REFERENCES

- [1] The OPC Foundation, OLE for Process Control Data Access Standard (Updated), Version 1.0A, September 11, 1997, <http://www.opcfoundation.org/>
- [2] Microsoft Corporation, Real-Time Systems and Microsoft Windows NT, Microsoft Developer Network, June 29, 1995, <http://premium.microsoft.com/msdn/library/bkgmd/html/realtime.htm>

*Paul Fischer joined the RadiSys software group in August of 1997 to help design and market their INtime real-time extension for Microsoft Windows NT. Previously, Fischer spent seven years at Force Computers in various engineering and marketing roles related to the application of software to embedded systems and, most recently, as the chair and editor of the BusNet VMEbus Backplane Protocol committee sponsored by the VITA Standards Organization (VSO). He has more than fifteen years experience with real-time embedded systems serving in a variety of engineering and marketing roles for Parker-Compumotor, Seagate Technology, and several Silicon Valley start-ups. Fischer has an MSE from UC Berkeley and a BSME from the University of Minnesota.*