Allen-Bradley

Using Event Tasks with Logix5000™ Controllers

Event-based tasks give Logix5000™ controllers a more effective way of gaining high-speed processing without compromising CPU performance.

Event Tasks Take Controllers to the Next Level

Whether it is material handling, packaging, or a myriad of other manufacturing operations, routine and non-routine tasks are a common occurrence. Handling them appropriately within the control system means the difference between smooth, error-free production and unnecessary downtime and waste. For example, a shipping company moves thousands of packages along a maze of conveyors toward their destination. Executing a task at the wrong time instantly sends a package down a wrong path and delays the delivery of the package.

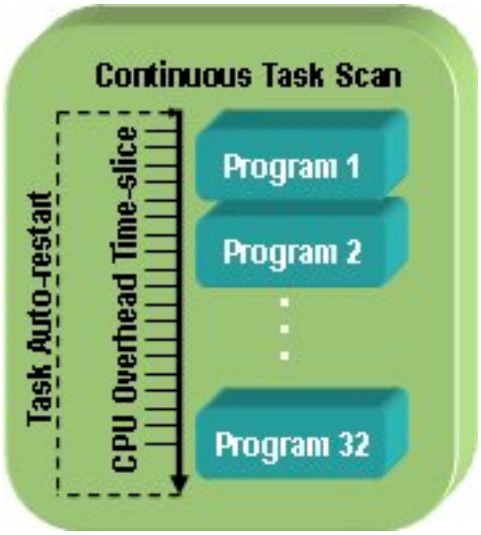
One step toward smooth, error-free production is to execute code when needed without burdening the controller with unnecessary execution. With event tasks, a new feature of Allen-Bradley Logix5000™ controllers, this has become much easier.

Logix5000 Controllers Support Multiple Tasks

In a Logix5000 controller, a task defines how and when the controller executes the various sections of application code. With support for 32 different tasks, a Logix5000 controller provides flexibility when constructing your application. It also lets you prioritize the task so that you can tune the controller's execution to meet the problem at hand. Before event tasks, Logix5000 users relied on continuous and periodic tasks.

Using the Continuous Task

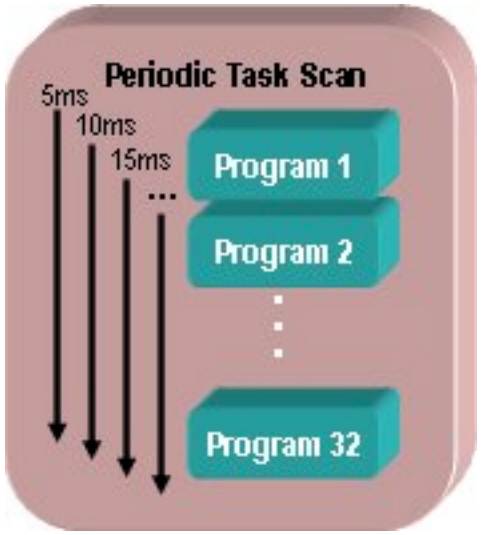
A continuous task is what normally comes to mind when you think about how a programmable controller traditionally operates: The controller scans the code from top to bottom. When it completes, it goes back to the top and restarts the process.



In a Logix5000 controller, the continuous task operates as a background task at the lowest priority. It uses any CPU time that remains after the controller executes the other tasks.

Using a Periodic Task

Like the continuous task, a periodic task executes all of its code from top to bottom. However, when a periodic task completes its scan, it waits a pre-configured time interval before restarting.



Because you can have multiple periodic tasks that can be triggered at the same time, the controller needs to determine which tasks it can or cannot interrupt. This is accomplished by way of a priority setting for each period task.

- Tasks with higher priority cause lower priority tasks (and the continuous task) to be suspended so that they can take control and perform their operation.
- Tasks with lower priority wait for higher priority tasks to complete before being given an opportunity to execute.
- Tasks configured at the same priority level are automatically time-sliced back and forth on a 1millesecond basis till one of the tasks completes.

Because the scan time required to execute a continuous task can be quite lengthy, periodic tasks let you move a small amount of code into a separate execution thread and trigger its execution at a higher rate.

- In the past the function of a periodic task was accomplished by placing multiple copies of some select code within the continuous task. The code was placed at appropriate intervals so that the input would be detected and outputs controlled.
- The periodic task simplifies the creation and management of this high-speed code. This lets you control a process that might occur at a significantly faster rate than the normal scan of the continuous task can attain.

Using an Event Task

One downside of a periodic task is that it always executes regardless of whether or not the input that it is monitoring changes state. This over-execution burdens the controller and lowers the performance of the remaining tasks. To alleviate this situation, event tasks provide a more effective way of balancing the need for high-speed processing with CPU performance.

Initially supported by ControlLogix® and SoftLogix5800™ controllers, event tasks let you execute a piece of code based on the detection of some incident that occurs in the control system.

The types of incidents (triggers) that can initiate the event task include:

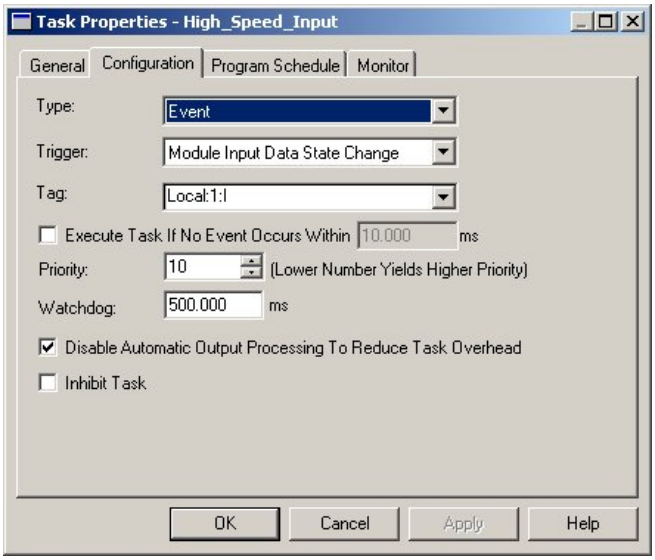
- traditional input point trigger
- receipt of produced/consumed tags
- certain motion operations
- instruction in another task on the controller

Similar to periodic tasks, you assign a priority to each event task. This lets the event task either immediately take control at a high priority or wait at a low priority for other tasks to complete. Because event tasks only execute when their trigger is initiated, they reduce the amount of code a controller must scan on a regular basis. This frees up additional controller bandwidth to perform other operations or improve the overall performance of the application. This additional performance coupled with ability of an event task to respond to high speed inputs gives the control system extra capacity to increase production output.

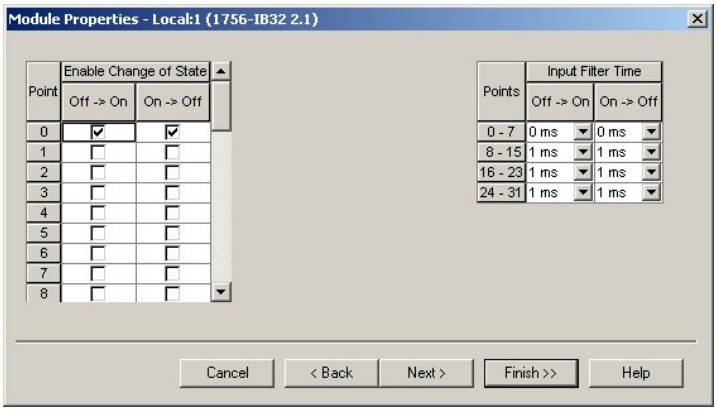
Applying an Event Task

ControlLogix and SoftLogix controllers can use event tasks in a variety of scenarios. Event tasks are valuable for applications such as packaging and material handling, where detection of a package by an input sensor must be processed quickly to drive an output solenoid to manipulate or divert product.

With Logix5000 controllers, much of the burden for event detection is offloaded from the controller to the input module.



ControlLogix input modules let you configure “Change-Of-State” detection on the module.



When configured, the module continually monitors the state of the inputs, looking for state changes.

- You have full control over which bits and even which transition (off-to-on, or on-to-off) will be used to initiate the event task.
- Based on this configuration, the ControlLogix module detects and communicates the occurrence to the controller.

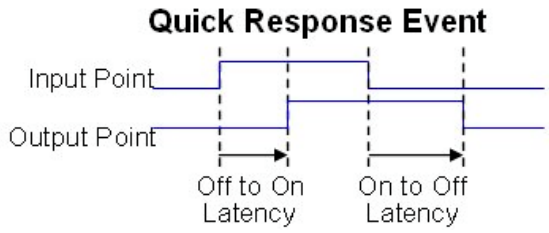
Upon receipt of the information from the module, the controller initiates the event task and executes the application code configured in the programs and routines within the task. This saves valuable execution time by avoiding the need for the controller to poll the inputs and perform change-of-state detection. An added benefit is that you can easily capture and process multiple events without additional CPU overhead.

You can apply the basic execution model of an event task to a variety of scenarios, depending on what you are controlling and how you have to handle it. For example, use ControlLogix input event tasks for:

- quick-response events
- short-duration input events
- synchronized-execution events.

Quick Response Events

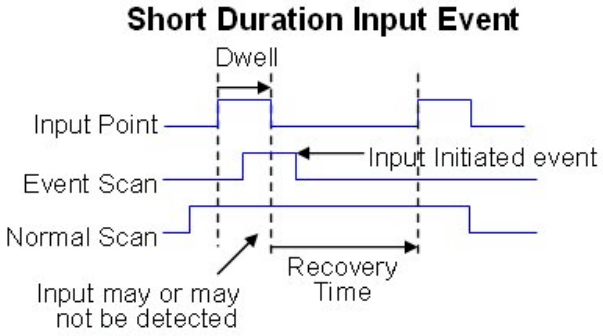
In a quick-response event, you want to minimize the latency (delay) between the input changing state and the control of an output.



An application that might use this is a packaging line where each time the product is detected you want to apply a label. Because of the speed with which product moves by the labeler, an event task is the only way to guarantee that you detect and label every load.

Short-Duration Input Events

A short-duration input event is an input whose duration is shorter than the normal scan of the controller but the event does not occur very often. In this case the input could change state (on/off) and then change back (off/on) faster than the normal scan of the continuous or periodic task.

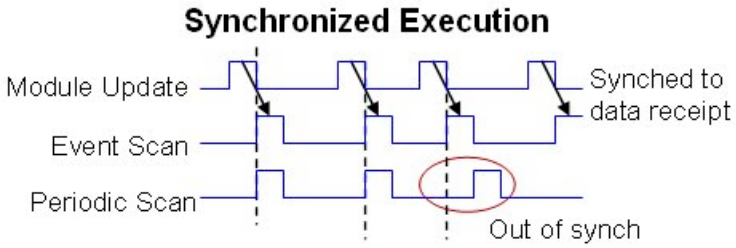


Once the module's Change-Of-State detection captures the input, the module sends this information to the controller. The controller responds and executes the event task even though the input may have turned back off by the time the code in the event task executes.

An example of this type of scenario is a bottling line where a sensor detects an improperly capped bottle that must be rejected. Because of the high speed of the line, an event task is the only way to detect the reject and respond to it.

Synchronized-Execution Events

A synchronized-execution event is typically used with analog data, where the arrival of new data triggers an operation.



Because each analog module has its own internal sample period for converting analog input signals to digital floating values, an asynchronous loop can exist between the receipt of new input values and the execution of the code for the loop. This discrepancy means that PID calculations execute with stale data, which could result in calculation errors. With an event task, you can easily synchronize the PID calculation to the arrival of flow transmitter data and avoid this potential error.

Local and Remote I/O Locations

Logix5000 controllers let you use both local and remote locations for the input module that triggers an event task.

- Because of application performance constraints, input modules being used to initiate and event task will generally be located in the same chassis as the controller that needs to respond to the event.
- You can also place ControlLogix input modules in a remote chassis connected via a ControlNet™ or EtherNet/IP network. Each network adds some additional delays but for many applications the benefit of reducing the CPU loading overrides the need for speed.

Using the SoftLogix5800 Virtual Backplane

The SoftLogix5800 controller lets you initiate events from its virtual backplane. A toolkit is available to configure a specialized software process such as a barcode ID search algorithm into the virtual backplane. This lets you trigger code to execute within the SoftLogix5800 application

Getting Started with Input-Driven Events

Before applying input-driven event tasks, there are some key design aspects to consider. Here are some tips to help ease the configuration process and ensure the best performance:

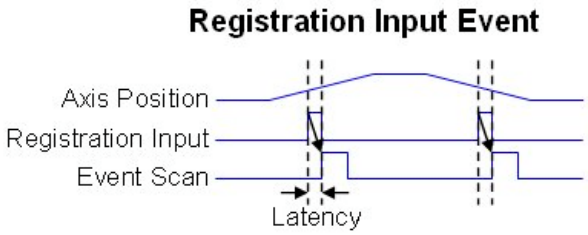
- Keep in mind that the throughput of the operation depends on a number of factors, including module type, input voltage, module temperature, module filter and response time settings, backplane size and loading, and processor type.
- Place the module that triggers an event in the same chassis as the Logix5000 controller. Placing it in a remote chassis adds to response times and requires additional communication and processing.
- Limit the number of modules that are in the chassis with the event module and controller. Additional modules increase the potential for backplane delays.
- For digital inputs, restrict the triggering input to a single point on the module. When you enable change-of-state, all inputs on a module trigger a single event. Multiple points increase the chance of a task overlap. (Task is triggered while it is already executing, resulting in a minor fault.)
- Set the priority of your event task as the highest priority task on the controller. Selecting a lower priority forces the event to wait for the periodic task to complete its execution.
- Limit the number of event tasks. A high number of event tasks reduces the available CPU bandwidth and increases the chance of a task overlap.
- Choose modules with the best response times.
- If you perform motion control with the controller, keep in mind that the motion planner may slow down the execution of event and periodic tasks.

Motion Events

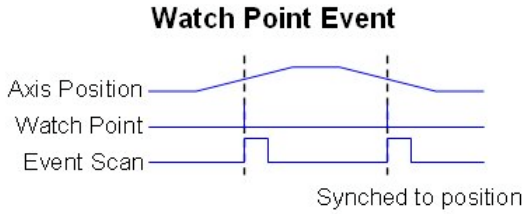
Logix5000 controllers provide a highly integrated motion solution capable of controlling multiple servo axes in addition to conventional control. Event tasks extend this capability by letting you initiate an event task based on various motion-related triggers, including:

- registration input
- watch point
- completion of the motion planner

The registration input lets you execute specific application code when a motion axis reaches a specific location. A physical device determines the location and triggers the task.

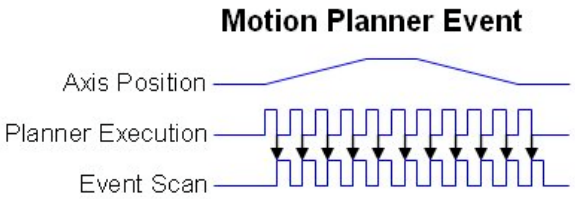


A watch point also lets you execute specific application code when a motion axis reaches a specific location. However, a watch point is a software-based position that serves as the trigger for the event.



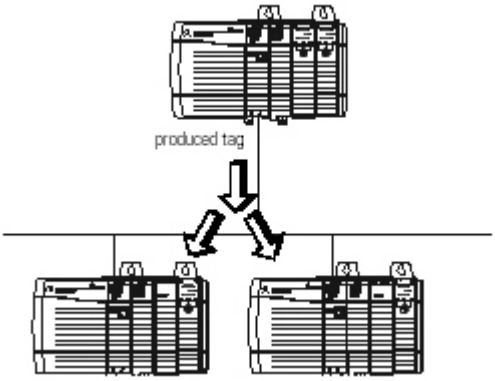
Both registration inputs and watch points are valuable for packing applications where you want to synchronize an operation to the position of some servo axis as it moves product through a machine.

Finally, the completion of the motion planner can trigger an event task. This lets you use specialized control algorithms to override the normal motion loop processing.



Produced/Consumed Tag Events

The produced and consumed tag capability of Logix5000 controllers provides an easy way to exchange data between controllers with minimal programming.



Because a controller produces (transmits) and consumes (receives) data asynchronous to the execution of its control program, the program may begin processing with a mixture of old and new data. Event tasks provide an easy way synchronize the data exchange to ensure that a consuming controller receives all of the data before processing begins.

- After the producing controller loads the data into its produced tag, the controller initiates an immediate output (IOT) instruction.
- The consuming controller simply configures an event task to look for the trigger via the consumed tag.

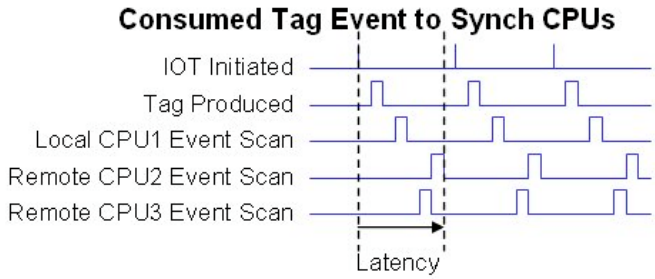
This technique provides the following advantages:

- automates the data detection processing on the consumer
- eliminates complex handshaking code
- improves the transfer rate

Synchronize Controllers

In addition to exchanging data between controllers, produced and consumed tags let you use event tasks to coordinate the operations of multiple controllers in a distributed system.

- You can use the produced/consumed tag as a proxy to initiate code in the distributed controllers (consumers) based on something occurring in a master controller (producer).
- Multiple controllers can simultaneously consume the same produced tag.



This technique lets you use smaller, more modular controllers in flexible manufacturing cells. You can reconfigure the cells while avoiding the interlock-wiring issues that would otherwise exist.

Estimating Event Throughput

To estimate the throughput time from input to output (screw to screw), use the following worksheet:

Consideration:	Value:
1. What is the input filter time of the module that triggers the event task? This is typically shown in milliseconds. Convert it to microseconds (µs).	µs
2. What is the hardware response time for the input module that triggers the event task? Make sure you use the appropriate type of transition (Off → On or On → Off).	µs
3. What is the backplane communication time?	µs
4. What is the total execution time of the programs of the event task?	µs
5. What is the backplane communication time? (Same value as step 3.)	µs
6. What is the hardware response time of the output module.	µs
7. Add steps 1 through 6. This is the minimum estimated throughput, where execution of the motion planner or other tasks *do not* delay or interrupt the event task.	µs
8. What is the scan time of the motion group?	µs
9. What is the total scan time of the tasks that have a higher priority than this event task (if any)?	µs
10. Add steps 7 through 9. This is the nominal estimated throughput, where execution of the motion planner or other tasks delay or interrupt the event task.	µs

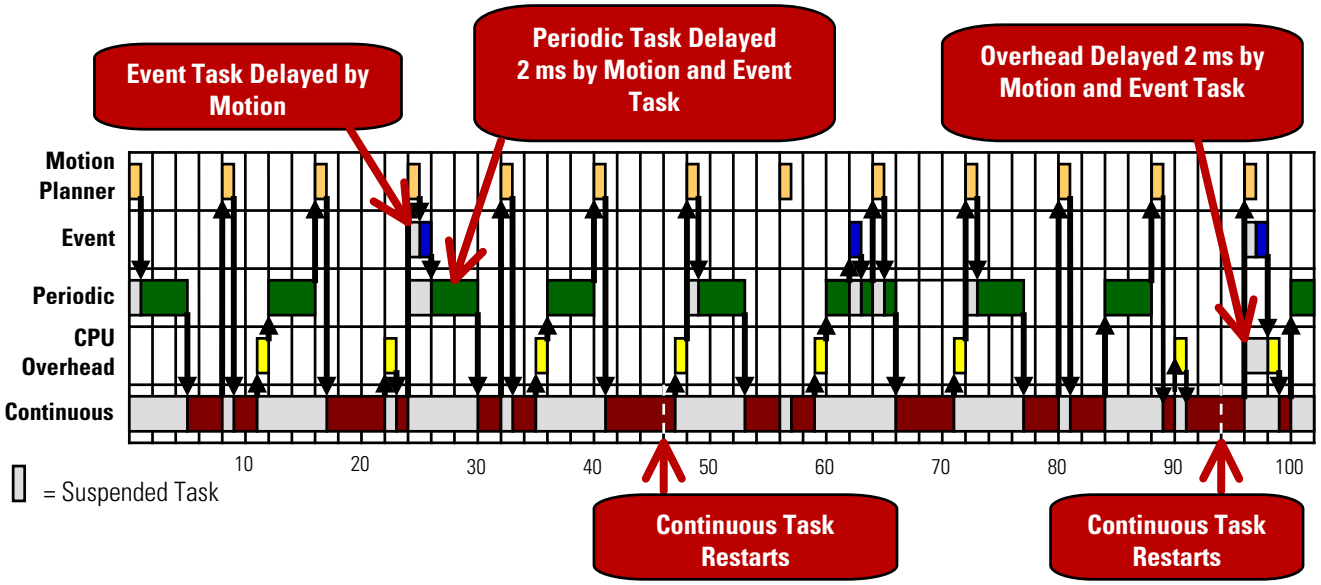
If the chassis size is:	Use this value (worst case):
4 slot	13 µs
7 slot	22 µs
10 slot	32 µs
13 slot	42 µs
17 slot	54 µs

Additional Design Considerations

As you design an event task, consider the following factors that also effect the speed at which the task can respond to the input signal.

<table>
<tr><th>Consideration:</th><th>Description:</th></tr>
<tr><td>motion planner</td><td>The motion planner interrupts all your tasks, regardless of their priority.
The number of axes and coarse update period for the motion group effects how long and how often the motion planner executes.
If the motion planner is executing when a task is triggered, the task waits until the motion planner is done.
If the coarse update period occurs while a task is executing, the task pauses to let the motion planner execute.</td></tr>
<tr><td>trends</td><td>Each active trend (histogram) interrupts all your tasks, regardless of their priority.
The trend interrupts your tasks at the sample period that you define for the trend.
You can have up to 32 trends active at one time.
The worst case is when all trends capture data at the same sample period.
To estimate the length of time that trends interrupt your tasks, use the following formula:<table><tr><th>If the controller type is:</th><th>Then the interrupt is:</th></tr><tr><td>1756-L55Mxx</td><td>(100 µs / trend) + (20 µs / tag)</td></tr><tr><td>1756-L63</td><td>(75 µs / trend) + (15 µs / tag)</td></tr></table></td></tr>
<tr><td>amount of code in the event task</td><td>Each logic element (rung, instruction, structured text construct, etc…) adds scan time to the task.</td></tr>
<tr><td>task priority</td><td>If the event task is not the highest priority task, a higher priority task may delay or interrupt the execution of the event task.</td></tr>
<tr><td>CPS and UID instructions</td><td>If one of these instructions is active, the event task cannot interrupt the currently executing task. (The task with the CPS or UID.)</td></tr>
<tr><td>communication interrupts</td><td>The following actions of the controller interrupt a task, regardless of the priority of the task
communication with I/O modules
Modules that have large data packets have a greater impact, such as the 1756-DNB module.
serial port communication</td></tr>
</table>

The following example shows the timing relationship between three user tasks, the motion planner, and system overhead.



Task	Configuration	Execution Time	Duration
Motion Planner	8 ms	assumes 1 ms	1 ms
Event	Trigger = Input Priority = 1	1 ms	1 to 2 ms
Periodic	Trigger = 12 ms period Priority = 2	4 ms	4 to 6 ms
CPU Overhead	20% of continuous task (i.e., 1 ms out of every 5 ms)	assumes 1 ms worst case	1 to 7 ms
Continuous	n/a	20 ms	≈ 46 to 48 ms

Reaping the Benefits

By incorporating the suggestions that we have outlined, you can use event tasks to get the following significant benefits:

- Faster performance and reduced costs by executing tasks only when needed. This frees up CPU time for other operations and improves the efficiency of the controller.
- Increase in output due to improved loop closure times.
- Reduction in development and maintenance costs because programs are more modular.

For More Information

For detailed, step-by-step information on how to configure and program an event task, see *Logix5000™ Controllers Common Procedures*, publication 1756-PM001.

www.rockwellautomation.com

Corporate Headquarters
Rockwell Automation, 777 East Wisconsin Avenue, Suite 1400, Milwaukee, WI, 53202-5302 USA, Tel: (1) 414.212.5200, Fax: (1) 414.212.5201

Headquarters for Allen-Bradley Products, Rockwell Software Products and Global Manufacturing Solutions
Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444
Europe/Middle East/Africa: Rockwell Automation SA/NV, Vorstlaan/Boulevard du Souverain 36, 1170 Brussels, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640
Asia Pacific: Rockwell Automation, 27/F Citicorp Centre, 18 Whitfield Road, Causeway Bay, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Headquarters for Dodge and Reliance Electric Products
Americas: Rockwell Automation, 6040 Ponders Court, Greenville, SC 29615-4617 USA, Tel: (1) 864.297.4800, Fax: (1) 864.281.2433
Europe/Middle East/Africa: Rockwell Automation, Brühlstraße 22, D-74834 Elztal-Dallau, Germany, Tel: (49) 6261 9410, Fax: (49) 6261 17741
Asia Pacific: Rockwell Automation, 55 Newton Road, #11-01/02 Revenue House, Singapore 307987, Tel: (65) 6356-9077, Fax: (65) 6356-9011

Publication LOGIX-WP003A-EN-P — June, 2003