Graco PD2K Dual Panel Rockwell SDK

Release 0.3.1

Graco, Inc.

Nov 03, 2022

CONTENTS

CHAPTER

ONE

OVERVIEW

The PD2K Dual Panel Rockwell Software Development Kit (SDK) is a collection of sample code for easily controlling a PD2K proportioner through a Rockwell PLC. The code is designed to simplify many common integration challenges, allowing you to build a working system very quickly.

1.1 About the PD Platform

The ProMix PD Platform is a family of electric dosing pump systems utilizing positive displacement (PD) technology. More information can be found on the Graco product page.

This SDK is specifically designed for the ProMix PD2K Dual Panel Proportioner for Automatic Spray Applications, which includes two independent 2-component mix units controlled from a Communication Gateway Module (CGM). SDKs for the other ProMix PD products can be found at help.graco.com.

1.1.1 Related Manuals

These manuals are available for download at www.graco.com.

Manual number	Description
3A4486	ProMix PD2K Dual Fluid Panel Electric Proportioner for Automatic Spray Applications, Operation
332458	ProMix PD2K Proportioner for Automatic Spray Applications, Installation
334494	ProMix PD2K CGM Installation Kit, Instructions/Parts

1.2 Compatibility

The examples in this SDK are built using Rockwell Studio 5000 Logix Designer® v32.04. The GracoPd2kDual_Example.ACD project is built for a 5069-L306ER CompactLogix™ Controller using firmware v32. If using a different controller, the model may be changed using Logix software.

All code is compatible with CompactLogix™ and ControlLogix™ controllers firmware v32 and later. Backwards compatibility with earlier firmware versions may be possible, but is not tested and not guaranteed.

All code is written in Ladder language. The main reason for this is portability - Rockwell's licensing structure supports Ladder at the base level, but requires an upgraded license to work with other PLC languages. Writing all code in Ladder lets any PLC programmer view the code and modify it if needed, regardless of which license they use.

CHAPTER

TWO

DESIGN

This chapter discusses various decisions in designing the SDK.

2.1 Command model standard

Each of the AOIs in this SDK follow a convention called the "Command model standard". This convention is loosely based on the PLCopen design recommendations for motion control blocks.

Generally speaking, every AOI is designed to perform a particular command or set of commands. Their interfaces all follow the same conventions, so all blocks of a given model type are called in the same way.

There are three different command models:

- Function model
- Execute model
- Enable model

The function model is for simple stateless operations, while the execute and enable models run over time and may depend on their internal memory. These models will be covered in more detail below.

2.1.1 The state output parameter

Each command can be thought of as a state machine, where each instance of the command exists in exactly one state at any point in time. These states are represented by an output parameter `state`. The values range between 16#0000 and 16#FFFF (i.e., only the lower 16-bits are used, the rest are ignored).

The state values are standardized such that the most-significant byte corresponds to a particular category. These categories are defined in the following table:

State category	State value (hex)	Description
IDLE	16#0xxx	The command is not running (enable model only)
BUSY	16#1xxx	The command is running (execute model only)
DONE	16#2xxx	The command completed successfully (execute/function models only)
VALID	16#3xxx	The command is running without issues/errors (enable model only)
ABORTED	16#4xxx	The command was aborted (either locally or by a higher-priority command)
ERROR	16#8xxx	An error occurred while processing the command
ERROR_BUSY	16#9xxx	An error occurred, but the command is busy attempting to recover (enable model only)

There can be multiple independent states within each category - for example, states 16#8000, 16#8001, and 16#8002 would represent distinct error conditions for that command.

The command's state is also expressed in the form of boolean output parameters, each corresponding to the bits in `state`'s most significant byte. The flags are defined in the following table:

State flag	bit number	Description
BUSY	0 (2#0001)	The command is running
DONE	1 (2#0010)	The command completed successfully (execute/function models only)
VALID	1 (2#0010)	The command is running without issues/error (enable model only)
ABORTED	2 (2#0100)	The command was aborted (either locally or by a higher-priority command)
ERROR	3 (2#1000)	An error occurred while processing the command

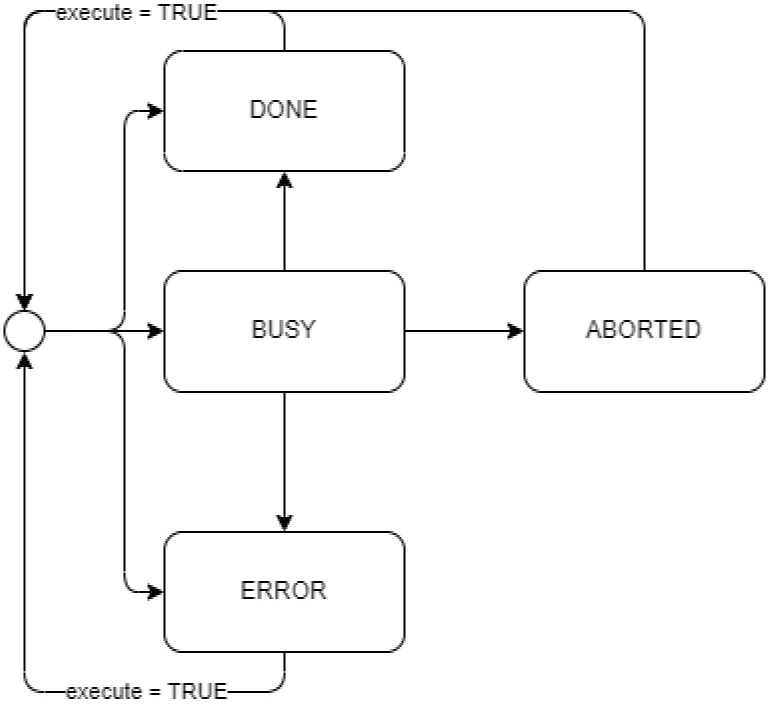
For example, if `state` is set to 16#8000, the ERROR flag would be on and the rest would be off. In many cases, using the flags is more convenient than the state code value. More details regarding the state categories and flags for each of the command models will be covered in the following sections.

2.1.2 Function Model

The function model represents a simple, stateless operation that processes immediately. An example is DecodeDateTime - this command reads the source input parameters and computes a native datetime value.

Function model commands typically use only DONE and ERROR states. If something goes wrong during execution (e.g. a parameter is out of range), an error code can be returned to notify the caller. If a command has no error conditions, then the state code can be omitted altogether.

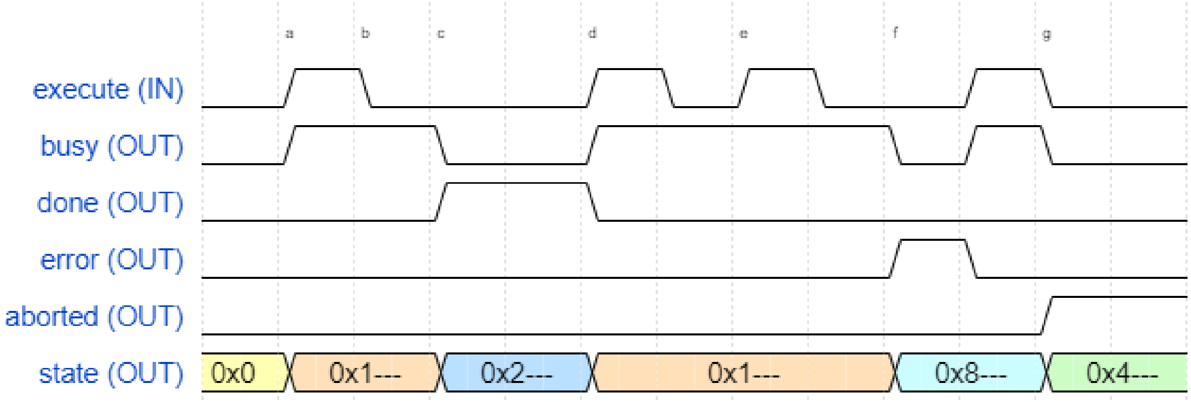
2.1.3 Execute model



The execute model is used for commands that run from start to finish. Once started, the command runs until it either completes the operation (DONE), experiences an error (ERROR), or is aborted (ABORTED). This model is stateful; its behavior depends on previous state and can change over time.

Execute model commands are triggered using an `execute` boolean input parameter. Each rising edge on this parameter executes the command once. The command only re-triggers once the command is in a "terminal" state - i.e., DONE, ERROR, or ABORTED. Triggering the command while already running does nothing.

Here is a timing diagram for the execute model:

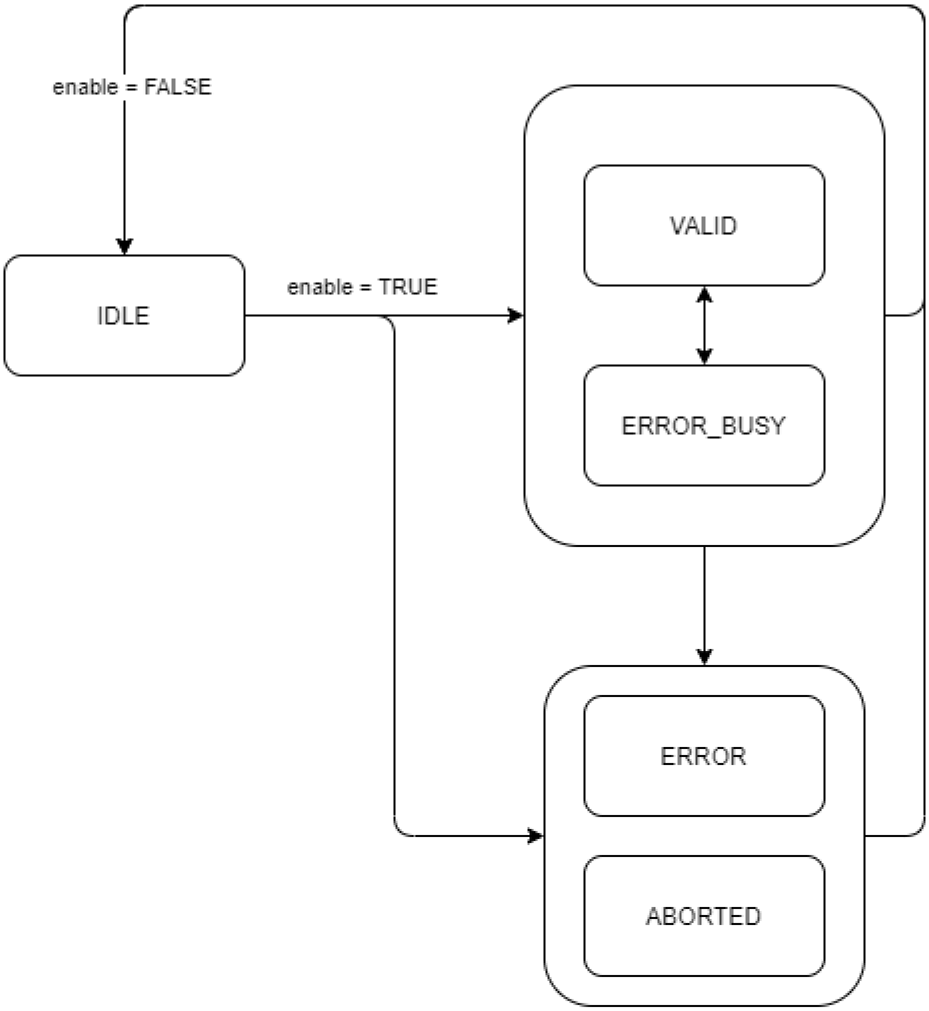


a.	A rising edge on execute starts the command, setting the busy flag to true.
b.	Execute is automatically cleared by the command.
c.	When the command completes, busy is set to false and done is set to true.
d.	When command is executed again, done is set to false and busy is set to true.
e.	Additional execute signals while busy are ignored, rather than restarting the command.
f.	If a problem occurs while processing the command, the error flag is set to true.
g.	Similarly, if something interrupts the command, the aborted flag is set to true.

Note that the `execute` input will automatically be cleared by the AOI; there is no need to reset the trigger from outside logic. This provides additional flexibility, e.g. when controlling from browser-based HMIs that lack momentary push buttons. The input can still be written from a regular coil - either way, the command will behave the same.

Some commands (but not all) may include a `cancel` input for stopping a command early. `cancel` inputs work the same way as `execute`, except they trigger a transition from BUSY to ABORTED.

2.1.4 Enable model

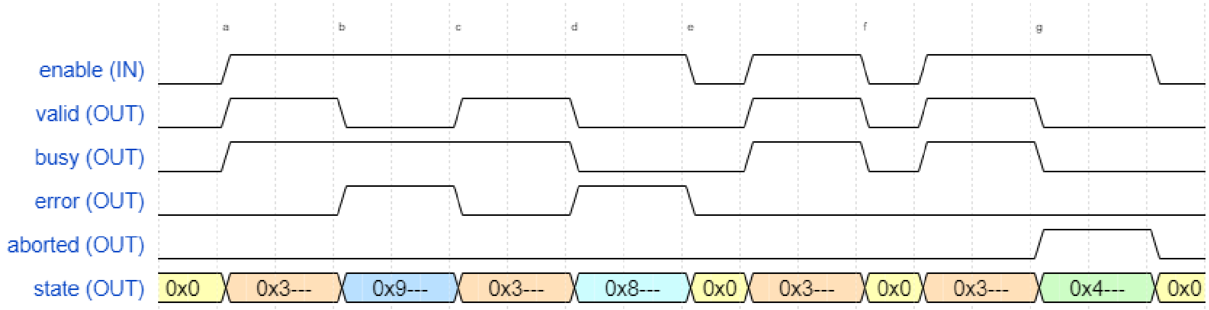


The enable model is used for commands that run indefintely. This differs from the execute model in that the enable model does not have a DONE condition. Instead, this model refers to normal operation as VALID. The enable model is useful for things like jog operations.

Enable model commands use an `enable` boolean input to control their state. When `enable` is false, the command is IDLE and does nothing. When true, the command runs its logic sequence and writes to its outputs.

An example of the enable model is the MainControl AOI. When enabled, the AOI responds to inputs and updates the system mode accordingly. Disabling the AOI allows it to be “detached” from the device’s registers, which could then be controlled through some other logic.

Here is a timing diagram for the enable model:



a.	A rising edge on enable starts the command. The busy and valid flags are set to true.
b.	When an error occurs, error is set to true and valid is set to false. In this case, the block logic can automatically recover from the error, so the busy flag stays at true.
c.	The block logic handled the error - valid is set back to true and error is set back to false.
d.	In this case, an error occurred that requires user intervention. Error is set to true, and both valid and busy are set to false.
e.	Setting enable to false resets the block and clears the error.
f.	Setting enable to false also sets both valid and busy to false.
g.	The command may also be aborted, causing the aborted flag to be true and valid/busy to be false.

Unlike the execute model, the state flags for the enable model are not all mutually-exclusive. Under normal operation, the VALID and BUSY flags are true, meaning the command is running without issues. If an issue does occur, there are two possibilities - either it is something the AOI can deal with automatically, or it is something outside of its control.

In the former scenario, the command enters an ERROR_BUSY state, which sets both the ERROR and BUSY flags to true. This means the command has identified an issue and is working to resolve it, e.g. retrying a failed message.

In the latter scenario, the command enters an ERROR state, without the BUSY flag. The command has experienced an issue that it cannot fix, and it is up to the calling code to resolve it. When this happens, the command must be reset by setting `enable` to false. It can then be re-enabled once the issue has been dealt with.

The ABORTED state works the same way as the ERROR state, but this is typically used for situations where another process in the program has overwritten some value the command was using. Because enable model commands can be disabled at any point, they do not need a `cancel` input, like in the execute model.

Note that, while enable model commands are designed to be stopped at any time, there may be several cycles where the command performs some cleanup logic before returning to IDLE state.

2.2 Sharing access to device registers

Many of the AOIs in this SDK write to the same sets of Ethernet/IP module data. For example, The ReadAlarmInfo and ReadRecipe AOIs both call SendDCS internally, which uses the DCS registers to query data from the PD2K. Only one DCS request can be sent at a time, so care must be taken to avoid writing the registers from multiple locations in logic at the same time. For example, if ReadAlarmInfo and ReadRecipe were both executed at the same time and allowed to write to the DCS registers, they would interfere with one another and result in a race condition.

To avoid this, each of the AOIs in this SDK are designed using a concept known as a semaphore. In general, semaphores prevent simultaneous writes to a shared resource (like the device registers) using a locked/unlocked mechanism. In short, the AOIs will take turns without interfering with one another.

Take the example of executing both the ReadAlarmInfo and ReadRecipe AOIs at the same time. Before either AOI does anything, they will each read the DCS acknowledge and DCS command register values. If both these are 0 (i.e. no operation, or NOP), then that AOI can assume no other AOI is currently using the DCS registers and thus, continue its sequence. If either register is non-zero, however, then some other AOI has accessed it first, so the AOI will go into a wait state until either the registers become available again or the AOI times out.

Note that this mechanism only works because all AOIs follow the same rules of only accessing registers when they are considered unlocked. Nothing prevents a programmer from writing to a register directly, and doing so will likely conflict with the SDK AOIs.

It’s also worth noting that this coordinating behavior works best with only 2-3 active AOIs at a time. If many blocks are waiting, It’s possible some of them will never be called because the earlier blocks will always take control first. For this reason, the programmer may still need to execute these in a controlled sequence.

2.3 Accessing module data

Nearly all the AOIs are designed to operate directly on a PD2K module’s data. These AOIs use two InOut parameters, `moduleStatus` and `moduleCmds`. The former should be passed the tag with data coming from the PD2K, while the latter should be passed tag with data going out to the PD2K. For example, given a module named `GRACO_PD2K`, the `moduleStatus` parameter should be passed `GRACO_PD2K:I.Data`, and the `moduleCmds` parameter should be passed `GRACO_PD2K:O.Data`. Note that the `.Data` subtag is used, which is an array of DINTs. This allows one to use the AOIs with dummy tags not associated with a real device, e.g. for simulations/testing.

CHAPTER

THREE

API DOCUMENTATION

This chapter provides a reference of all AOIs and UDTs defined in the SDK.

Note: In the project files, all AOIs and UDTs start with prefixes, e.g. "GracoPd2kDual_". These prefixes identify which library each component belongs to. This helps prevent naming collisions when multiple SDKs are used in the same PLC program.

3.1 GracoPd2kDual API

3.1.1 Add-On Instructions (AOIs)

MainCtrl

Provides several inputs for controlling the PD2K proportioner (e.g. system mode, setpoints).

The enable input must be on for any other inputs to take effect. The boolean command inputs each function like execute inputs, meaning the command will automatically reset these once set to true.

The current control mode setting (pressure or flow) determines which of the setpoint inputs are written to the PD2K system. This switches automatically when the control mode is changed.

clearAlarm (INPUT BOOL)
Clear alarm

completeJob (INPUT BOOL)
Complete job

ctrlMode (INPUT BOOL)
Ctrl mode SP (0=flow, 1=pressure)

enable (INPUT BOOL)
Enable command

mix (INPUT BOOL)
Goto mix

mixFill (INPUT BOOL)
Goto mix fill

powerOff (INPUT BOOL)
Power off

powerOn (INPUT BOOL)
Power on

quickStop (INPUT BOOL)
Goto quick stop

recipePurge (INPUT BOOL)
Goto recipe purge

standby (INPUT BOOL)
Goto standby

mixUnitNum (INPUT DINT)
Mix unit number (1-2)

flowCtrlSP (INPUT DINT)
Flow control setpoint (cc/min)

presCtrlSP (INPUT DINT)
Pressure control setpoint (psi)

busy (OUTPUT BOOL)
Command busy flag

valid (OUTPUT BOOL)
Command valid flag

aborted (OUTPUT BOOL)
Command aborted flag

error (OUTPUT BOOL)
Command error flag

state (OUTPUT DINT)
State code

moduleStatus (INOUT DINT[62])
Data from module to PLC

moduleCmds (INOUT DINT[30])
Data from PLC to module

State Codes

- 16#0000 - Idle
- 16#3XXX - Valid
- 16#8000 - Error, invalid mix unit number

ReadAlarmInfo

Reads data from an alarm record.

indexNum (INPUT DINT)
the chronological alarm index number to look up, where 0 is the most recent, and 199 is the 200th most recent.

execute (INPUT BOOL)
executes the command.

code (INOUT *STRING4*)
the code for the alarm.

dateTime (INOUT *TypeDateTime*)
The date and time the alarm occurred.

moduleStatus (INOUT DINT[62])
The module input data.

moduleCmds (INOUT DINT[30])
The module output data.

busy (OUTPUT BOOL)
The busy command flag

done (OUTPUT BOOL)
The done command flag

aborted (OUTPUT BOOL)
The aborted command flag

error (OUTPUT BOOL)
The error command flag

state (OUTPUT DINT)
State data for the command

State Codes

- 16#1xxx - Busy
- 16#2000 - Done
- 16#4000 - Aborted, command overwritten by another process
- 16#8000 - Error, invalid command ID
- 16#8001 - Error, timeout waiting for DCS registers to become available
- 16#8002 - Error, command unsuccessful
- 16#8003 - Error, timeout waiting for acknowledge from PD2K

ReadEventInfo

Reads data from an event record.

indexNum (INPUT DINT)
the chronological event index number to look up, where 0 is the most recent, and 199 is the 200th most recent.

execute (INPUT BOOL)
executes the command.

code (INOUT *STRING4*)
the code for the event.

dateTime (INOUT *TypeDateTime*)
The date and time the event occurred.

moduleStatus (INOUT DINT[62])
The module input data.

moduleCmds (INOUT DINT[30])
The module output data.

busy (OUTPUT BOOL)
The busy command flag

done (OUTPUT BOOL)
The done command flag

aborted (OUTPUT BOOL)
The aborted command flag

error (OUTPUT BOOL)
The error command flag

state (OUTPUT DINT)
State data for the command

State Codes

- 16#1xxx - Busy
- 16#2000 - Done
- 16#4000 - Aborted, command overwritten by another process
- 16#8000 - Error, invalid command ID
- 16#8001 - Error, timeout waiting for DCS registers to become available
- 16#8002 - Error, command unsuccessful
- 16#8003 - Error, timeout waiting for acknowledge from PD2K

ReadFlushSeq

Returns the current configuration values for a flush sequence number.

flushSeqNum (INPUT DINT)
the flush sequence number to configure. The acceptable range is 1-5. If not in range, an error will be raised.

execute (INPUT BOOL)
executes the command.

moduleStatus (INOUT DINT[62])
The module input data.

moduleCmds (INOUT DINT[30])
The module output data.

gunPurgeTime (OUTPUT DINT)
the current gun purge time in seconds.

initialFlushVol (OUTPUT DINT)
the current value for the initial flush volume setpoint in cc.

finalFlushVol (OUTPUT DINT)
the current value for the final flush volume setpoint in cc.

numWashCycles (OUTPUT DINT)
the current number of wash cycles.

strokesPerWashCycle (OUTPUT DINT)
the current number of strokes per wash cycle.

busy (OUTPUT BOOL)
The busy command flag

done (OUTPUT BOOL)
The done command flag

aborted (OUTPUT BOOL)
The aborted command flag

error (OUTPUT BOOL)
The error command flag

state (OUTPUT DINT)
State data for the command

State Codes

- 16#1xxx - Busy
- 16#2000 - Done
- 16#4000 - Aborted, command overwritten by another process
- 16#8000 - Error, invalid command ID
- 16#8001 - Error, timeout waiting for DCS registers to become available
- 16#8002 - Error, command unsuccessful
- 16#8003 - Error, timeout waiting for acknowledge from PD2K

ReadGrandTotals

Returns the material grand total volume data.

execute (INPUT BOOL)
executes the command.

moduleStatus (INOUT DINT[62])
The module input data.

moduleCmds (INOUT DINT[30])
The module output data.

materialA (OUTPUT DINT)
the grand total volume of material A used, in gallons.

materialB (OUTPUT DINT)
the grand total volume of material B used, in gallons.

materialAB (OUTPUT DINT)
the summed grand total of materials A and B, in gallons.

solvent (OUTPUT DINT)
the grand total volume of solvent used, in gallons.

busy (OUTPUT BOOL)
The busy command flag

done (OUTPUT BOOL)
The done command flag

aborted (OUTPUT BOOL)
The aborted command flag

error (OUTPUT BOOL)
The error command flag

state (OUTPUT DINT)
State data for the command

State Codes

- 16#1xxx - Busy
- 16#2000 - Done
- 16#4000 - Aborted, command overwritten by another process
- 16#8000 - Error, invalid command ID
- 16#8001 - Error, timeout waiting for DCS registers to become available
- 16#8002 - Error, command unsuccessful
- 16#8003 - Error, timeout waiting for acknowledge from PD2K

ReadJobInfo

Reads data from a job record.

indexNum (INPUT DINT)
the chronological job index number to look up, where 0 is the most recent, and 199 is the 200th most recent.

execute (INPUT BOOL)
executes the command.

userID (INOUT *STRING9*)
the user ID used for the job.

dateTime (INOUT *TypeDateTime*)
the completion date and time of the job record.

moduleStatus (INOUT DINT[62])
The module input data.

moduleCmds (INOUT DINT[30])
The module output data.

mixUnitNum (OUTPUT DINT)
Mix unit number

jobNum (OUTPUT DINT)
The actual job number of the record.

recipeNum (OUTPUT DINT)
The recipe number used for the job.

abVol (OUTPUT DINT)
The sum of materials (A+B) used for the job in cc.

busy (OUTPUT BOOL)
The busy command flag

done (OUTPUT BOOL)
The done command flag

aborted (OUTPUT BOOL)
The aborted command flag

error (OUTPUT BOOL)
The error command flag

state (OUTPUT DINT)
State data for the command

State Codes

- 16#1xxx - Busy
- 16#2000 - Done
- 16#4000 - Aborted, command overwritten by another process
- 16#8000 - Error, invalid command ID
- 16#8001 - Error, timeout waiting for DCS registers to become available
- 16#8002 - Error, command unsuccessful

- 16#8003 - Error, timeout waiting for acknowledge from PD2K

ReadRecipe

Returns the current configuration values for a recipe number.

mixUnitNum (INPUT DINT)
mix unit number (1-2)

recipeNum (INPUT DINT)
the recipe number to read. The acceptable range is 0-60.

execute (INPUT BOOL)
Executes the command.

moduleStatus (INOUT DINT[62])
The module input data.

moduleCmds (INOUT DINT[30])
The module output data.

materialA (OUTPUT DINT)
the current value for the first material number (paint component).

materialB (OUTPUT DINT)
the current value for the second material number (catalyst component).

flushSeqA (OUTPUT DINT)
the current flush sequence number associated with material A.

flushSeqB (OUTPUT DINT)
the current flush sequence number associated with material B.

mixRatioSP (OUTPUT REAL)
the current mix ratio setpoint. The value corresponds to the ratio antecedent, i.e. the material A amount. For example, a value of 2.5 corresponds to a ratio of 2.5:1 (material A to material B).

potLifeTimeSP (OUTPUT DINT)
the current pot life time setpoint in minutes.

mixPressureTol (OUTPUT DINT)
mix pressure tolerance (%)

busy (OUTPUT BOOL)
The busy command flag

done (OUTPUT BOOL)
The done command flag

aborted (OUTPUT BOOL)
The aborted command flag

error (OUTPUT BOOL)
The error command flag

state (OUTPUT DINT)
State data for the command

State Codes

- 16#1xxx - Busy
- 16#2000 - Done
- 16#4000 - Aborted, command overwritten by another process
- 16#8000 - Error, invalid command ID
- 16#8001 - Error, timeout waiting for DCS registers to become available
- 16#8002 - Error, command unsuccessful
- 16#8003 - Error, timeout waiting for acknowledge from PD2K

ReadUserID

Returns the current user ID for the system.

The returned value is a string of up to 9 characters, not including a null terminator.

execute (INPUT BOOL)
Executes the command

mixUnitNum (INPUT DINT)
mix unit number (1-2)

userID (INOUT *STRING9*)
The returned user ID

moduleStatus (INOUT DINT[62])
The module input data

moduleCmds (INOUT DINT[30])
The module output data

busy (OUTPUT BOOL)
The busy command flag

done (OUTPUT BOOL)
The done command flag

aborted (OUTPUT BOOL)
The aborted command flag

error (OUTPUT BOOL)
The error command flag

state (OUTPUT DINT)
State data for the command

State Codes

- 16#1xxx - Busy
- 16#2000 - Done
- 16#4000 - Aborted, command overwritten by another process
- 16#8000 - Error, invalid command ID
- 16#8001 - Error, timeout waiting for DCS registers to become available
- 16#8002 - Error, command unsuccessful
- 16#8003 - Error, timeout waiting for acknowledge from PD2K

RecipeChange

Performs a recipe change operation.

execute (INPUT BOOL)
executes the command.

mixUnitNum (INPUT DINT)
mix unit number (1-2)

recipeNum (INPUT DINT)
The next recipe number to use.

moduleStatus (INOUT DINT[62])
The module input data.

moduleCmds (INOUT DINT[30])
The module output data.

busy (OUTPUT BOOL)
The busy command flag

done (OUTPUT BOOL)
The done command flag

aborted (OUTPUT BOOL)
The aborted command flag

error (OUTPUT BOOL)
The error command flag

state (OUTPUT DINT)
State data for the command

State Codes

- 16#1xxx - Busy
- 16#2000 - Done
- 16#4000 - Aborted, command overwritten by another process
- 16#8000 - Error, system must be powered on
- 16#8001 - Error, another recipe change is already in progress

- 16#8002 - Error, system alarm is active
- 16#8003 - Error, another operation is active
- 16#8004 - Error, alarm occurred
- 16#8005 - Error, unexpected system mode
- 16#8006 - Error, recipe change did not finish
- 16#8007 - Error, timeout waiting for system registers to become available
- 16#8008 - Error, invalid mix unit number

SendDCS

Sends a dynamic command structure (DCS) message to the PD2K system.

For more information about DCS commands, see the system operation manual.

execute (INPUT BOOL)
executes the command.

cmdID (INPUT DINT)
the command number to send.

arg0 (INPUT DINT)
argument 0.

arg1 (INPUT DINT)
argument 1.

arg2 (INPUT DINT)
argument 2.

arg3 (INPUT DINT)
argument 3.

arg4 (INPUT DINT)
argument 4.

arg5 (INPUT DINT)
argument 5.

arg6 (INPUT DINT)
argument 6.

arg7 (INPUT DINT)
argument 7.

arg8 (INPUT DINT)
argument 8.

moduleStatus (INOUT DINT[62])
The module input data.

moduleCmds (INOUT DINT[30])
The module output data.

return0 (OUTPUT DINT)
Return value 0 from the PD2K.

return1 (OUTPUT DINT)
Return value 1 from the PD2K.

return2 (OUTPUT DINT)
Return value 2 from the PD2K.

return3 (OUTPUT DINT)
Return value 3 from the PD2K.

return4 (OUTPUT DINT)
Return value 4 from the PD2K.

return5 (OUTPUT DINT)
Return value 5 from the PD2K.

return6 (OUTPUT DINT)
Return value 6 from the PD2K.

return7 (OUTPUT DINT)
Return value 7 from the PD2K.

return8 (OUTPUT DINT)
Return value 8 from the PD2K.

busy (OUTPUT BOOL)
The busy command flag

done (OUTPUT BOOL)
The done command flag

aborted (OUTPUT BOOL)
The aborted command flag

error (OUTPUT BOOL)
The error command flag

state (OUTPUT DINT)
State data for the command

State Codes

- 16#1xxx - Busy
- 16#2000 - Done
- 16#4000 - Aborted, command overwritten by another process
- 16#8000 - Error, invalid command ID
- 16#8001 - Error, timeout waiting for DCS registers to become available
- 16#8002 - Error, command unsuccessful
- 16#8003 - Error, timeout waiting for acknowledge from PD2K

StatusToUDT

Reads a PD2K module array into a structured UDT.

moduleStatus (INOUT DINT[62])
The module input data.

statusUDT (INOUT *TypePd2kDualStatus*)
The structured data from the device.

WriteFlushSeq

Configures the values for a flush sequence number.

flushSeqNum (INPUT DINT)
the flush sequence number to configure. The acceptable range is 1-5. If not in range, an error will be raised.

gunPurgeTime (INPUT DINT)
the gun purge time to set in seconds. The acceptable range is 0-999. If not in range, an error will be raised.

initialFlushVol (INPUT DINT)
the initial flush volume setpoint in cc. The acceptable range is 0-9999. If not in range, an error will be raised.

finalFlushVol (INPUT DINT)
the final flush volume setpoint in cc. The acceptable range is 0-9999. If not in range, an error will be raised.

numWashCycles (INPUT DINT)
the number of wash cycles to set. The acceptable range is 0-99. If not in range, an error will be raised.

strokesPerWashCycle (INPUT DINT)
the number of strokes per wash cycle to set. The acceptable range is 0-99. If not in range, an error will be raised.

execute (INPUT BOOL)
executes the command.

moduleStatus (INOUT DINT[62])
The module input data.

moduleCmds (INOUT DINT[30])
The module output data.

busy (OUTPUT BOOL)
The busy command flag

done (OUTPUT BOOL)
The done command flag

aborted (OUTPUT BOOL)
The aborted command flag

error (OUTPUT BOOL)
The error command flag

state (OUTPUT DINT)
State data for the command

State Codes

- 16#1xxx - Busy
- 16#2000 - Done
- 16#4000 - Aborted, command overwritten by another process
- 16#8000 - Error, invalid command ID
- 16#8001 - Error, timeout waiting for DCS registers to become available
- 16#8002 - Error, command unsuccessful
- 16#8003 - Error, timeout waiting for acknowledge from PD2K

WriteMaterialReadyFlag

Sets the flag used to signal that the upstream material management is loaded correctly prior to a recipe change.

This flag is only used when multiple materials for a pump are fed to the PD2K via a single valve at the inlet valve stack (i.e. a piggable system).

mixUnitNum (INPUT DINT)
mix unit number (1-2)

materialReadyFlag (INPUT BOOL)
the new flag value (false=not ready, true=ready).

execute (INPUT BOOL)
executes the command.

moduleStatus (INOUT DINT[62])
The module input data.

moduleCmds (INOUT DINT[30])
The module output data.

busy (OUTPUT BOOL)
The busy command flag

done (OUTPUT BOOL)
The done command flag

aborted (OUTPUT BOOL)
The aborted command flag

error (OUTPUT BOOL)
The error command flag

state (OUTPUT DINT)
State data for the command

State Codes

- 16#1xxx - Busy
- 16#2000 - Done
- 16#4000 - Aborted, command overwritten by another process
- 16#8000 - Error, invalid command ID
- 16#8001 - Error, timeout waiting for DCS registers to become available
- 16#8002 - Error, command unsuccessful
- 16#8003 - Error, timeout waiting for acknowledge from PD2K

WriteRecipe

Configures the values for a recipe number.

execute (INPUT BOOL)
Executes the command.

mixUnitNum (INPUT DINT)
mix unit number (1-2)

recipeNum (INPUT DINT)
the recipe number to configure. The acceptable range is 0-60.

materialA (INPUT DINT)
the first material number to set (paint component). The acceptable range is 0-30.

materialB (INPUT DINT)
the second material number to set (catalyst component). The acceptable range is 0,31-34.

materialAFlushSeq (INPUT DINT)
the flush sequence number to use with material A. The acceptable range is 1-5.

materialBFlushSeq (INPUT DINT)
the flush sequence number to use with material B. The acceptable range is 1-5. If not in range, an error will be raised.

mixRatioSP (INPUT REAL)
the mix ratio setpoint to use. The value corresponds to the ratio antecedent, i.e. the material A amount. For example, a value of 2.5 corresponds to a ratio of 2.5:1 (material A to material B). For single component recipes, set the value to 0. The max precision is two decimal places; anything beyond that will be rounded. The acceptable range is 0-50.

potLifeTimeSP (INPUT DINT)
the total pot life time to set in minutes. The acceptable range is 0-999 minutes.

mixPressureTol (INPUT DINT)
mix pressure tolerance (%)

moduleStatus (INOUT DINT[62])
The module input data.

moduleCmds (INOUT DINT[30])
The module output data.

busy (OUTPUT BOOL)
The busy command flag

done (OUTPUT BOOL)
The done command flag

aborted (OUTPUT BOOL)
The aborted command flag

error (OUTPUT BOOL)
The error command flag

state (OUTPUT DINT)
State data for the command

State Codes

- 16#1xxx - Busy
- 16#2000 - Done
- 16#4000 - Aborted, command overwritten by another process
- 16#8000 - Error, invalid command ID
- 16#8001 - Error, timeout waiting for DCS registers to become available
- 16#8002 - Error, command unsuccessful
- 16#8003 - Error, timeout waiting for acknowledge from PD2K

WriteUserID

Sets the current user ID for the system.

The user ID can be up to 9 characters in length. The value sent to the PD2K will include a null terminator for a total of 10 characters. Only ASCII characters are supported.

execute (INPUT BOOL)
Executes the command

mixUnitNum (INPUT DINT)
Mix unit number (1-2)

userID (INOUT *STRING9*)
The new user ID

moduleStatus (INOUT DINT[62])
The module input data

moduleCmds (INOUT DINT[30])
The module output data.

busy (OUTPUT BOOL)
The busy command flag

done (OUTPUT BOOL)
The done command flag

aborted (OUTPUT BOOL)
The aborted command flag

error (OUTPUT BOOL)
The error command flag

state (OUTPUT DINT)
State data for the command

State Codes

- 16#1xxx - Busy
- 16#2000 - Done
- 16#4000 - Aborted, command overwritten by another process
- 16#8000 - Error, invalid command ID
- 16#8001 - Error, timeout waiting for DCS registers to become available
- 16#8002 - Error, command unsuccessful
- 16#8003 - Error, timeout waiting for acknowledge from PD2K

3.1.2 User-Defined Types (UDTs)

TypePd2kPumpStatusFlags

Status flags for a PD2K pump.

off (BOOL)
pump is off.

standby (BOOL)
pump is in standby.

busy (BOOL)
pump is busy.

flushing (BOOL)
pump is flushing.

priming (BOOL)
pump is priming.

TypePd2kDualStatus

Status data read from a PD2K Dual system.

gunTriggerStatus1 (BOOL)
gun 1 trigger status (1 = on)

gunTriggerStatus2 (BOOL)
gun 2 trigger status(1 = on)

mixUnit1 (*TypePd2kMixUnitStatus*)
mix unit 1

mixUnit2 (*TypePd2kMixUnitStatus*)
mix unit 2

pump1 (*TypePd2kPump*)
pump 1

pump2 (*TypePd2kPump*)
pump 2

pump3 (*TypePd2kPump*)
pump 3

pump4 (*TypePd2kPump*)
pump 4

dcsReturn0 (DINT)
DCS return 0

dcsReturn1 (DINT)
DCS return 1

dcsReturn2 (DINT)
DCS return 2

dcsReturn3 (DINT)
DCS return 3

dcsReturn4 (DINT)
DCS return 4

dcsReturn5 (DINT)
DCS return 5

dcsReturn6 (DINT)
DCS return 6

dcsReturn7 (DINT)
DCS return 7

dcsReturn8 (DINT)
DCS return 8

dcsAck (DINT)
DCS acknowledge

TypePd2kMixUnitStatus

Status data for a mix unit

eventFlag (BOOL)
event flag

standby (BOOL)
in standby

interlockStatus (BOOL)
interlock status

systemMode (DINT)
system mode

systemModeFlags (*TypePd2kSystemModeFlags*)
system mode flags

actualMixFlow (DINT)
actual mix flow (cc/min)

actualMixRatio (DINT)
actual mix ratio

actualPotLifeRemaining (DINT)
actual pot life remaining

activeRecipeNum (DINT)
active recipe number

activeMaterialA (DINT)
active material A

activeMaterialB (DINT)
active material B

activeFlushSeqA (DINT)
active flush sequence A

activeFlushSeqB (DINT)
active flush sequence B

activeRatioSP (DINT)
active ratio SP

activePotLifeSP (DINT)
active pot life SP

activeJobNum (DINT)
active job number

jobSprayVolA (DINT)
job sprayed volume A (cc)

jobSprayVolB (DINT)
job sprayed volume B (cc)

jobSolventVol (DINT)
job solvent volume used (cc)

TypePd2kPump

Status data for a PD2K pump

status (DINT)
pump status

statusFlags (*TypePd2kPumpStatusFlags*)
pump status flags

actualMaterial (DINT)
actual material number

actualFlowRate (DINT)
actual flow rate (cc/min)

actualFluidPres (DINT)
actual fluid pressure (psi)

TypePd2kSystemModeFlags

Flags for the PD2K system mode.

pumpOff (BOOL)
pump off

colorChange (BOOL)
color change enabled

colorChangePurgeA (BOOL)
color change purging component A

colorChangePurgeB (BOOL)
color change purging component B

colorChangeFilling (BOOL)
color change filling

colorChangeMixFill (BOOL)
color change mix fill

colorChangeMix (BOOL)
color change mixing

colorChangeMixIdle (BOOL)
color change mix idle

purgeA (BOOL)
purging component A

purgeB (BOOL)
purging component B

standbyMixReady (BOOL)
standby mix ready

standbyFillReady (BOOL)
standby fill ready

standbyMixNotReady (BOOL)
standby mix not ready

standbyAlarm (BOOL)
standby alarm

lineFillingFlushing (BOOL)
line is filling or flushing

pumpPrimeFlush (BOOL)
pump is priming or flushing

maintCalibration (BOOL)
maintenance/calibration

mixSolventPush (BOOL)
mix solvent push

3.2 GracoCore API

3.2.1 Add-On Instructions (AOIs)

DecodeDateTime

Convert from Graco-formatted date and time data into a UDT.

dateSource (INOUT DINT)
Date source

timeSource (INOUT DINT)
Time source

dateTimeDest (INOUT *TypeDateTime*)
Datetime UDT destination

DecodeEventCode

Parses an event code from a series of characters stored in a DINT into a Rockwell String (4 characters max).

source (INOUT DINT)
Byte source

dest (INOUT *STRING4*)
String destination

DecodeUserID

Parses a user ID from a series of characters stored in DINTs into a Rockwell String (9 characters max).

source1 (INOUT DINT)
Source 1 (characters 1-4)

source2 (INOUT DINT)
Source 2 (characters 5-8)

source3 (INOUT DINT)
Source 3 (character 9)

dest (INOUT *STRING9*)
String destination

EncodeUserID

Parses a user ID String (max 9 characters) into a series of ASCII values stored in DINTs. These can then be written to a Graco CGM.

source (INOUT *STRING9*)
String source

dest1 (INOUT DINT)
Destination 1 (characters 1-4)

dest2 (INOUT DINT)
Destination 2 (characters 5-8)

dest3 (INOUT DINT)
Destination 3 (character 9)

Step

An implementation of the step pattern, useful for sequences in ladder logic.

doneTrigger (INPUT BOOL)
Trigger to set step to done

preCond (INPUT BOOL)
Precondition for step

isDone (OUTPUT BOOL)
Step is done

isRunning (OUTPUT BOOL)
Step is running

3.2.2 User-Defined Types (UDTs)

TypeDateTime

Represents a point in time.

year (DINT)
the year value (e.g. 2021).

month (DINT)
the month number (1=January, 2=February, …).

dayOfMonth (DINT)
the day number within the month.

dayOfWeek (DINT)
the day number within the week (1=Monday, 2=Tuesday, …, 7=Sunday).

hour (DINT)
the 24-hour value (0-23, where 0=midnight).

min (DINT)
the minute value (0-59).

sec (DINT)
the second value (0-59).

3.2.3 String Types

STRING4

Maximum characters: 4

STRING9

Maximum characters: 9

CHAPTER

FOUR

CHANGELOG

4.1 v0.3.1 (latest)

Release date: 2022-11-03

Added:

- Documentation is now available as an interactive HTML application - see html_manual directory.

Changed:

- Removed material ready flag input from RecipeChange AOI
- Removed range checking on most input parameters; letting this be handled at PD2K software level instead
- Updated execute model handling for AOIs - no longer uses an idle state nor a ready output.
- Removed ACM libs due to little interest from end users. May be added again in the future if they become more useful.
- Update PDF manual with new formatting and revised content.
- Changelog moved from text file to manual
- Various improvements to internal logic

4.2 v0.2.1

Release date: 2022-01-28

Added:

- Initial release of MainCtrl AOI
- initial release for GracoPd2kDual_Module.L5X file.
- Added rate-limiting and input buffers on execute model AOIs
- Including this changelog file with releases going forward.

Changed:

- AOIs now use Graco_Core AOIs internally
- Simplified ExampleProgram to use MainCtrl instead of separate rungs
- Removed polling rungs in ExampleProgram
- Changed example ACD to only use 1 PD2K instance

- modified ExampleProgram to use fewer intermediate tags
- changed all AOIs to accept DINT arrays rather than the specific module datatype. This allows the AOIs to be used more flexibly (e.g. operating on copies of module data if needed).
- changed the moduleInputs and moduleOutputs parameters to moduleStatus and moduleCmds respectively. This terminology is more explicit and is consistent with other product SDKs.
- added prefixes to all AOI parameters (i_, o_, and io_). This makes the parameter type more obvious when looking at an instance in ladder, since Logix does not clearly distinguish between inputs and outputs visually.
- Reworked the standard sequence code numbers for better consistency and simpler logic.
- renamed the status output params to "state", since "status" could be confused with the "moduleStatus" params.
- renamed ReadStatus AOI to StatusToUDT. Modified this AOI to output data as a UDT instead of separate output parameters.
- Rewrite of manual to be more consistent with other PD2K family ISDKs:
  - Changed overall format. No longer based on MS Word, so no .docx is included, only the .pdf.
  - Better internal links throughout document

4.3 v0.1.1

Release date: 2020-10-08

Added:

- initial release for GracoPd2kDual_AOIs.L5X
- initial release for GracoPd2kDual_Example.ACD
- initial release for manual
- initial release for GracoPd2kDual_Module library object (v1.1)
- initial release for GracoPd2kDual_AOI library object (v1.2)
- initial release for GracoPd2kDual_ExampleProgram library object (v1.2)
- initial release for GracoPd2kDual_Example ACM project
- initial release for ReadAlarmInfo AOI
- initial release for ReadData AOI
- initial release for ReadEventInfo AOI
- initial release for ReadFlushSeq AOI
- initial release for ReadGrandTotals AOI
- initial release for ReadJobInfo AOI
- initial release for ReadRecipe AOI
- initial release for ReadUserID AOI
- initial release for RecipeChange AOI
- initial release for SendDCS AOI
- initial release for WriteFlushSeq AOI
- initial release for WriteMaterialReadyFlag AOI

- initial release for WriteRecipe AOI
- initial release for WriteUserID AOI