

CHAPTER FIVE

5 Custom Profiling

IN THIS CHAPTER

This chapter explains how to use these custom profiling features:

- S-Curve Profiling 120
- Linear Interpolation..... 123
- Contouring (circular interpolation) 124
- Compiled Motion Profiling..... 136
- On-the-Fly Motion (pre-emptive GOs)..... 151
- Registration 155
- Synchronizing Motion..... 159

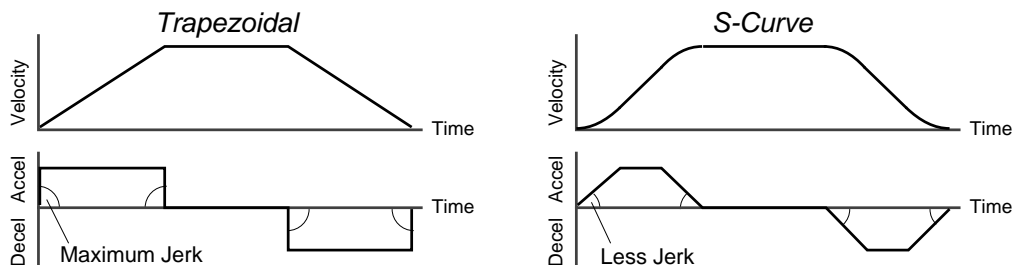
For basic motion (accel/velocity/distance), see page [53](#).

For homing profiles, see page [59](#).

For Following profiles, see page [166](#).

S-Curve Profiling

6K controllers allow you to perform *S-curve* move profiles, in addition to the usual trapezoidal profiles. S-curve profiling provides smoother motion control by reducing the *jerk* (rate of change) in acceleration and deceleration portions of the move profile (see drawing below). Because S-curve profiling reduces jerk, it improves position tracking performance, especially in linear interpolation applications (not contouring).



S-Curve Programming Requirements

To program an S-curve profile, you must use the *average accel/decel* commands provided in the 6K Series programming language. For every maximum accel/decel command (e.g., A, AD, HOMA, HOMAD, JOGA, JOGAD, etc.) there is an *average* command for S-curve profiling (see table below).

Maximum Accel/Decel Commands		Average ("S-Curve") Accel/Decel Commands	
Command	Function	Command	Function
A	Acceleration	AA	Average Acceleration
AD	Deceleration	ADA	Average Deceleration
HOMA	Home Acceleration	HOMAA	Average Home Acceleration
HOMAD	Home Deceleration	HOMADA	Average Home Deceleration
JOGA	Jog Acceleration	JOGAA	Average Jog Acceleration
JOGAD	Jog Deceleration	JOGADA	Average Jog Deceleration
JOYA	Joystick Acceleration	JOYAA	Average Joystick Acceleration
JOYAD	Joystick Deceleration	JOYADA	Average Joystick Deceleration
LHAD	Hard Limit Deceleration	LHADA	Average Hard Limit Deceleration
LSAD	Soft Limit Deceleration	LSADA	Average Soft Limit Deceleration
PA	Path Acceleration	PAA	Average Path Acceleration
PAD	Path Deceleration	PADA	Average Path Deceleration

Determining the S-Curve Characteristics

The command values for average accel/decel (AA, ADA, etc.) and maximum accel/decel (A, AD, etc.) determine the characteristics of the S-curve. To smooth the accel/decel ramps, you must enter average accel/decel command values that satisfy the equation $\frac{1}{2} A \leq AA < A$, where A represents maximum accel/decel and AA represents average accel/decel. Given this requirement, the following conditions are possible:

Acceleration Setting	Profiling Condition
AA > ½ A, but AA < A.....	S-curve profile with a variable period of constant acceleration. Increasing the AA value above the pure S-curve level (AA > ½ A), the time required to reach the target velocity and the target distance is decreased. However, increasing AA also increases jerk.
AA = ½ A	Pure S-curve (no period of constant acceleration—smoothest motion).
AA = A	Trapezoidal profile (but can be changed to an S-curve by specifying a new AA value less than A).
AA < ½ A; or AA > A.....	When you issue the GO command, the move will not be executed and an error message, *INVALID CONDITIONS FOR S_CURVE ACCELERATION-FIELD n, will be displayed.
AA = zero.....	S-curve profiling is disabled. Trapezoidal profiling is enabled. AA tracks A. (Track means the command's value will match the other command's value and will continue to match whatever the other command's value is set to.) However, if you enter an average decel command (e.g., ADA, HOMADA, etc.) equal to zero, you will receive the "INVALID DATA-FIELD n" error.
AA ≠ zero and AA ≠ A	S-curve profiling is enabled only for standard moves (e.g., not for contouring, compiled motion, or on-the-fly motion changes). All subsequent standard moves for that axis must comply with this equation: $\frac{1}{2} A \leq AA < A$.
AA > ½ A	Average accel/decel is raised above the pure S-curve level; this decreases the time required to reach the target velocity and distance. However, increasing AA also increases jerk. After increasing AA, you can reduce jerk by increasing A, but be aware that increasing A requires a greater torque to achieve the commanded velocity at the mid-point of the acceleration profile.
No AA value ever entered	Profile will default to trapezoidal. AA tracks A.

If you never change the A or AA deceleration commands, AA deceleration will track AA acceleration. However, once you change A deceleration, AA deceleration will no longer track changes in AA acceleration. For example, if you never change the AD or ADA command values, ADA will track the AA command value. But once you change AD, the ADA command value will no longer track the changes in the AA command value.

The calculation for determining S-curve average accel and decel move times is as follows (calculation method identical for S-curve and trapezoidal moves):

$$\text{Time} = \frac{\text{Velocity}}{A_{\text{avg}}} \quad \text{or} \quad \text{Time} = \sqrt{\frac{2 * \text{Distance}}{A_{\text{avg}}}}$$

Scaling affects the AA average acceleration (AA, ADA, etc.) the same as it does for the A maximum acceleration (A, AD, etc.). See page 48 for details on scaling.

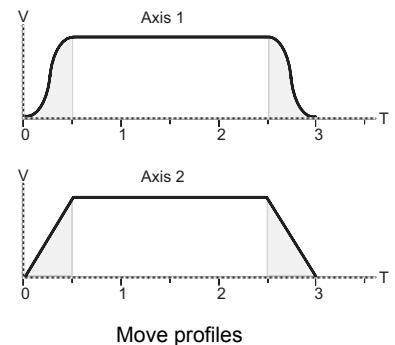
NOTE: Equations for calculating jerk are provided on page 122.

Programming Example

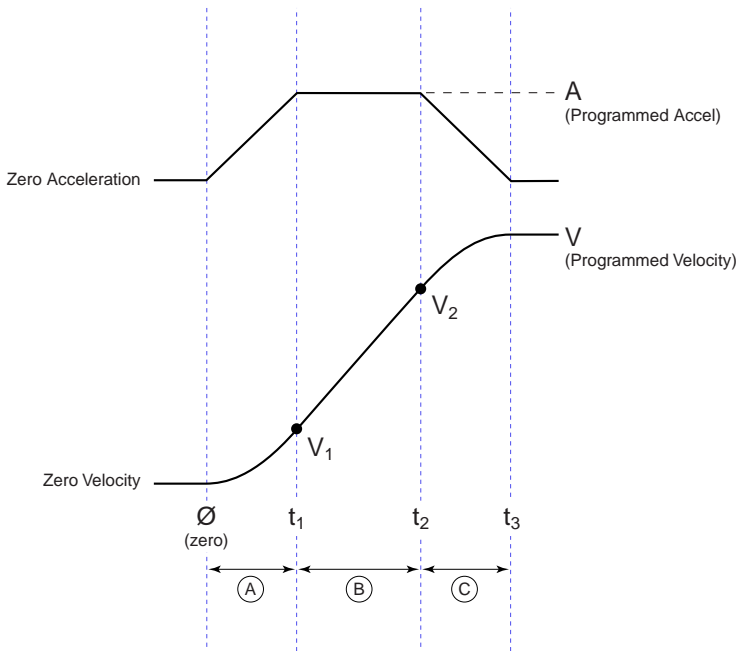
```

; In this example, axis 1 executes a pure S-curve and takes 1 second
; to reach a velocity of 5 rps; axis 2 executes a trapezoidal profile
; and takes 0.5 seconds to reach a velocity of 5 rps.
SCALE0      ; Disable scaling
DEF SCURV   ; Begin definition of program SCURV
@MA0       ; Select incremental positioning mode
@D40000    ; Set distances to 40,000 positive-
           ; direction steps
A10,10     ; Set max. accel to 10 rev/sec/sec
           ; on axes 1 and 2
AA5,10     ; Set avg. accel to 5 rev/sec/sec on
           ; axis 1, & 10 rev/sec/sec on axis 2
AD10,10    ; Set max. decel to 10 rev/sec/sec
           ; on axes 1 and 2
ADA5,10    ; Set avg. decel to 5 rev/sec/sec on
           ; axis 1, & 10 rev/sec/sec on axis 2
V5,5       ; Set velocity to 5 rps on axes 1 & 2
GO11      ; Execute motion on axes 1 and 2
END        ; End definition of program

```



Calculating Jerk



Rules of Motion:

$$\text{Jerk} = \frac{da}{dt}$$

$$a = \frac{dv}{dt}$$

$$v = \frac{dx}{dt} \quad (x = \text{distance})$$

Assuming the accel profile starts when the load is at zero velocity and the ramp to the programmed velocity is not compromised:

$$\text{Jerk} = J_A = \frac{A^2 * AA}{V (A-AA)}$$

- A = programmed acceleration (A, AD, HOMAD, etc.)
- AA = average acceleration (AA, ADA, HOMAA, etc.)
- V = programmed velocity (V, HOMV, etc.)

$$t_1 = \frac{A}{J_A}$$

$$t_2 = \frac{V}{AA} - \frac{A}{J_A}$$

$$t_3 = \frac{V}{AA}$$

NOTE: $t_3 - t_2 = t_1$

$$V_1 = \frac{J_A * t_1^2}{2} = \frac{A^2}{2 * J_A}$$

$$V_2 = V - \frac{A^2}{2 * J_A}$$

Ⓐ $t_1 \geq t \geq 0$

$$a(t) = J_A * t$$

$$v(t) = \frac{J_A * t^2}{2}$$

$$d(t) = \frac{J_A * t^3}{6}$$

a(t) = acceleration at time t
v(t) = velocity at time t
d(t) = distance at time t

Ⓑ $t_2 \geq t > t_1$

$$a(t) = A$$

$$v(t) = \frac{A^2}{2J_A} + A * (t - t_1)$$

$$d(t) = \frac{J_A * t_1^3}{6} + \left(\frac{A * (t - t_1)^2}{2} \right) + \left(V_1 * (t - t_1) \right)$$

Ⓒ $t_3 \geq t > t_2$

$$a(t) = A - (J_A * (t - t_2))$$

$$v(t) = V - \left(\frac{J_A * (t_3 - t)^2}{2} \right)$$

$$d(t) = \frac{V^2}{2AA} + \left(\frac{J_A * (t_3 - t)^3}{6} \right) - \left(V * (t_3 - t) \right)$$

Starting at a Non-Zero Velocity: If starting the acceleration profile with a non-zero initial velocity, the move comprises two components: a constant velocity component, and an s-curve component. Typically, the change of velocity should be used in the S-curve calculations. Thus, in the calculations above, you would substitute “(V_F - V_O)” for “V” (V_F = final velocity, V_O = initial velocity). For example, the jerk equation would be:

$$\text{Jerk} = J_A = \frac{A^2 * AA}{(V_F - V_O) (A-AA)}$$

Linear Interpolation

NOTE
 2-axis products can accommodate only 2-axis (X & Y) linear interpolation.

The controller allows you to perform *linear interpolation*, the process of moving two or three orthogonal (right angle) linear axes to achieve linear (straight line) motion; a fourth axis can also participate in the move profile. The task is to derive appropriate move parameters to move from a current location to a new location, where each position is specified by a set of *Cartesian coordinates*. All axes must start, accelerate, decelerate, and stop in a synchronized manner.

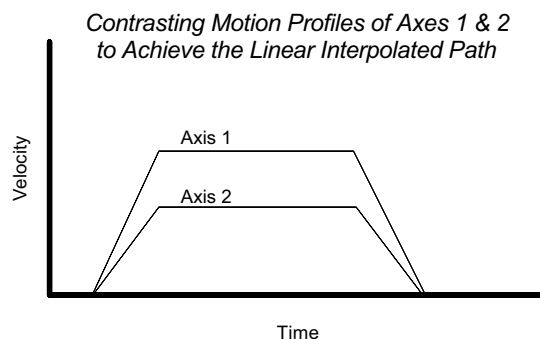
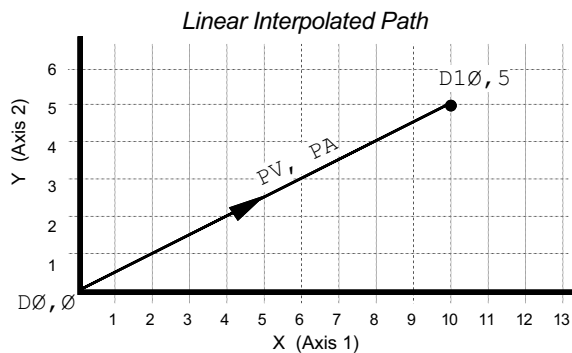
The Initiate Linear Interpolated Motion (GOL) command initiates linear interpolation moves based on the parameters set with the D, PA, PAD, and PV commands. You simply enter the desired path acceleration (PA), the path deceleration (PAD), and the path velocity (PV) to arrive at the point in space (*end point*) specified with the distance (D) command; the controller internally calculates each axis' actual move profiles to achieve a straight-line path with these parameters.

You can scale the acceleration, velocity, and distance with the SCLD command (see example below).

The GOL command starts motion on either or both axes. If the GOL command is issued without any arguments, motion will be started on both axes.

Code Sample

```
SCALE1      ; Enable scaling
@SCLD4000   ; Set distance, accel/decel, and velocity scale factor
             ; to 4000 steps/unit
PA25        ; Set the path acceleration to 25 units/sec/sec
PAD20       ; Set the path deceleration to 20 units/sec/sec
PV2         ; Set the path velocity to 2 units/sec
D10,5       ; Set the distance to 10 & 5 units on axes 1 & 2
GOL11       ; Initiate linear interpolated motion on axes 1 & 2
             ; (see drawings below)
```



Contouring (Circular Interpolation)

NOTE
2-axis products can accommodate only 2-axis (X & Y) contouring.

6K Series controllers allow you to define and execute two-dimensional motion paths. A *path* refers to the path traveled by the load in an X-Y plane, and must be defined before any motion takes place along that path. The X and Y axes can be specified as any of the controller's two axes

Controllers with ≥ 4 axes: A third axis, labeled the C axis, can be included to keep an angular position that changes linearly with the path direction. The path direction is the vector addition of the travel of axes X and Y. A fourth axis labeled the P axis can be included to keep a position that is proportional to the distance traveled along the path described by X and Y. The X, Y, C and P axes can be specified as any of the controller's four axes.

A *path* consists of one or more line or arc segments whose end-points are specified in terms of X and Y positions. The end-point position specifications can be made using either absolute or incremental programming. The segments can be lines or arcs, both of which are described in greater detail in the following sections. Each path segment is determined by the end-point coordinates, and in the case of arcs, by the direction and radius or center. It is possible to accelerate, decelerate or travel at constant velocity (feedrate) during any type of segment, even between segments. For each segment, the user can also specify an output pattern that can be applied to the programmable outputs at the beginning of that segment.

All paths are continuous paths, (i.e., the motion will not stop between path segments, but must stop at the end of a path). It is not possible to define a path that stops motion within the path definition and then continues that path. To achieve this result, two individual paths must be defined and executed. A path can, however, be stopped and resumed by using a *Pause/Continue input* (see page 83) while the path is executing. In this case, motion will be decelerated and resumed along the path without loss of position. If one axis is stopped due to any other reason, the other axis will stop abruptly, and motion cannot be resumed. Causes for motion being stopped can include encountering an end-of-travel limit, issuing a Kill (!K) command, detecting a stall (stepper axes), exceeding the maximum allowable position error (servo axes), etc.

Path Definition

Contouring paths are defined like programs (using the DEF and END commands), but are compiled with the PCOMP command and executed with the PRUN command. Programs intended to be compiled as paths are stored in *Program* memory. After they are compiled with the PCOMP command, they remain in program memory and the *segments* (PARCM, PARCOM, PARCOP, PARCP, and PLIN statements) from the compiled profile are stored in *Compiled* memory. The TDIR command reports which programs are stored as a compiled profile.



Compiled *GOBUF* segments and *PLCP* programs are also stored in compiled memory (see page 11).

The amount of RAM allocated for storing contouring path segments is determined by the MEMORY command setting. The list below identifies memory allocation defaults and limits for 6K Series products. Further details on re-allocating memory are provided on page 11.

Total memory (bytes)	300,000
Default allocation (program,compiled)	MEMORY150000, 150000
Maximum allocation for compiled profiles	MEMORY1000, 299000
Maximum No. of compiled profiles	300
Maximum No. of compiled profile segments	2069

CAUTION

Issuing a memory allocation command (e.g., MEMORY70000, 80000) will erase all existing programs and compiled contouring path segments. However, issuing the MEMORY command by itself (i.e., MEMORY—to request the status of how the memory is allocated) will not affect existing programs or path segments.

You can store the maximum number of paths possible (see table above) as long as each path has at least one segment, and the sum of all the segments of all the paths does not exceed the controller's *compiled* memory limitation. All path definitions can be compiled and ready to execute at any time. Paths defined using G commands are specified with a path name. Once a path definition is compiled, it can be executed repeatedly without being re-compiled.

Deleting (DEL) an existing path name will automatically delete the existing path compilation with that name. The PUCOMP command only deletes (“uncompiles”) the path compilation, not the path program.

Example Code

In the example commands below, storage space is made available for the definition of path WID3 by first deleting the compiled version of paths WID1 and WID2. The DEF statement begins the definition of the path WID3.

```
PUCOMP WID1 ; Remove compilation of WID1
PUCOMP WID2 ; Remove compilation of WID2
DEF WID3 ; Begin definition of WID3
```

Participating Axes

Units of Measure; Scaling

- If no scaling (SCALE0): Path acceleration, velocity, and distance are based on the resolution (DRES for steppers, ERES for servos) of axis 1. If multi-tasking is used, path motion units are based on the resolution of the first (lowest number) axis associated with the task (TSKAX).
- If scaling (SCALE1): The units used for path distance, acceleration, and velocity is determined by the SCLD value. For example, suppose you have 2 servo axes (axes 1 & 2) involved in contouring, both axes use encoder feedback with a resolution of 4000 counts/rev, axis 1 uses a 10:1 (10 turns per inch) leadscrew and axis 2 uses a 5:1 (5 turns per inch) lead screw, and you want to program in inches. For this application you would use the SCLD40000,20000 command to establish path motion units in inches: distance is inches, acceleration is inches/sec/sec, and velocity is inches/sec.

2-Axis Contouring

You can change the X-Y plane to an Y-X plane by using the PAXES command. A path definition default is PAXES1, 2. For any path that uses axis 2 as the X axis and axis 1 as the Y axis, the path definition must start with PAXES2, 1 (see example below).

Example Code

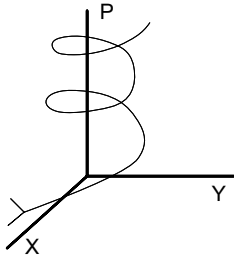
```
DEF DRAW1 ; Begin definition of DRAW1
PAXES2,1 ; Set contouring axes (axis 2 is X axis, axis 1 is Y axis)
```

4-Axis Contouring

Some contouring applications can require the execution of more than one path to complete a part or finish an operation. The application can require that different paths take place in different planes of a three dimensional work area. In addition, some of the paths can require a third axis to move either tangent, or proportional to the path. For these reasons, a four-axis controller offers great flexibility in the specification of participating axes.

You may want to begin your path definition with the PAXES command; this will ensure that you have specified the appropriate axes to participate in the path. The X, Y, tangent (C) and proportional (P) axes can be specified as any of the four axes, and this specification must be made before any of the path travel specifications are made. The X and Y axes must be specified, the third (tangent) axis labeled C and the fourth (proportional) axis labeled P are optional.

The C axis will maintain an angular position that changes linearly with the direction of travel in the X-Y plane. This allows the C axis to control an object, which must stay tangent (or normal) to the direction of travel such as a cutting tool. The C axis must also be specified by its signed resolution. The magnitude of the resolution is the number of C axis motor steps in 360 degrees of an arc drawn by the X and Y axes. The sign of the resolution specifies the direction of rotation of the C axis. Refer to the PTAN command.



The P axis will keep a position that is proportional to the distance traveled along the X-Y path as the path is executed. This allows the P axis to act as the Z axis in *helical interpolation* (see drawing at left), or to control the motion of any object that moves with distance and velocity proportional to the path. The P axis must also be specified by the signed ratio of P axis travel to path travel. The magnitude of this ratio can range from 0.001 to 1000. The sign of this ratio specifies the direction of rotation of the P axis. Refer to the PPRO command.

A sewing machine application can require all four axes (X, Y, C, and P). The X and Y axes would direct the sewing head along the required path. The C axis would keep the sewing head pointed into the direction of travel. The P axis would control the speed of the needle, so that an even stitch is made, regardless of path speed.

Example Code

The following example begins the definition of a path named DRAW1. The X and Y axes are specified to be axes 4 and 2. The path includes the C axis to be axis 1, with a resolution of 100,000 steps. It also includes the P axis to be axis 3, with a ratio of P axis travel to path travel specified as 2.5:1.

```
DEF DRAW1      ; Begin definition of DRAW1
PAXES4,2,1,3  ; Set contouring axes
PTAN100000    ; Define C axis resolution
PPRO2.5       ; Define P axis ratio
```

Path Acceleration, Deceleration, and Velocity

A path can be composed of many segments, each with their own motion parameters. The path velocity, acceleration, and deceleration specifications currently in effect at the time a segment is defined will apply to that segment. This allows construction of a path that moves at one velocity for a section of the path, then moves at a different velocity for another section.

In most cases, it will be desirable to maintain a constant velocity throughout the path, but it is easy to define a path in which each segment has its own velocity. For example, this can be useful when a tool needs to slow down to round a corner, or to allow the rate of glue application to be controlled by the path speed. Acceleration and deceleration can also be specified separately.

Example Code

The short code example below illustrates the specification of velocity, acceleration, and deceleration in that order. Note that the scaling commands cannot be placed inside a program.

```
PV0.5         ; Path velocity 10,000 counts/sec
PA16         ; Path acceleration 400,000 counts/sec/sec
PAD28       ; Path deceleration 700,000 counts/sec/sec
```

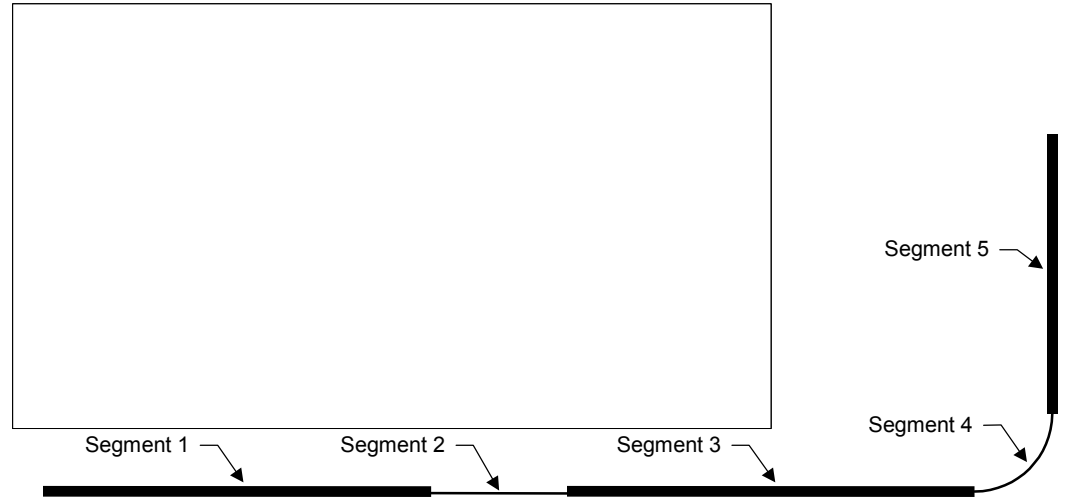
Path Final Velocity

The PVF command allows a line or arc segment to terminate with a final segment velocity (PVF) that can be different from the velocity traveled for the majority of that segment (specified with PV). PVF must be smaller than or equal to PV, and the path velocity change will take place at the PAD deceleration rate. Like the other path motion parameters (e.g., PV, PA, PAD), the PVF velocity is applied to the next line or arc compiled; however, unlike these other commands, the PVF command applies only to the next line or arc compiled. All subsequent lines and arcs terminate at the PV value currently in effect. For each line or arc that needs to terminate at a velocity different than PV, a new PVF command must be issued, even if the PVF value has not changed.

The most common use for the PVF command will be to cause a preceding line segment to decelerate to the path velocity at which the next line or arc needs to travel. In this case, the PVF value for the preceding line would be the same as the PV value of the reduced speed segment. This feature eliminates the need to create a special deceleration segment in the path in order to have the entire subsequent line or arc travel at the reduced speed.

Example Code

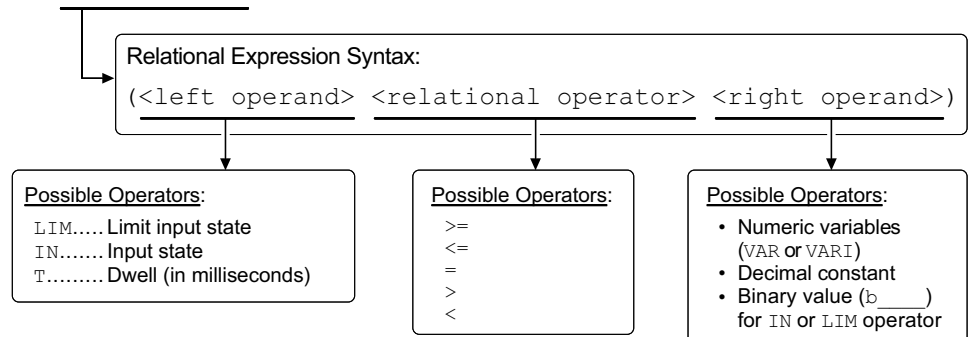
In the example below, segments 2 and 4 must travel at a path velocity of 1, while the remainder to the path travels at a path velocity of 5.



Conditional Path Execution

The PGOWHN command allows a fixed or conditional delay in path travel to occur as part of the contouring motion profile, without requiring program execution to monitor the PGOWHN conditions. Combined with the PVF command, PGOWHN provides a convenient method of imbedding a dwell within a contour.

PGOWHN (expression)



The PGOWHN expression can be specified as a time (dwell) in milliseconds (e.g., PGOWHN (T=2000)), or as inputs or limits matching the specified binary pattern (e.g., PGOWHN (LIM. 3=B1) or PGOWHN (2IN. 6=B0)). All participating axes will be at rest during a dwell, even if the previous segment had not ended in zero path velocity. In the latter case (limit or input condition), there will be an abrupt change to zero path velocity, until the PGOWHN condition is satisfied.

When the PGOWHN condition is satisfied, path velocity will ramp to the next segment's PV value. Although no motion occurs during a PGOWHN segment, it occupies one segment of compiled memory. Like the line and arc segments, an output pattern can be asserted for the duration of the PGOWHN dwell by preceding the PGOWHN statement with a POUT statement (see example below).

Example Code

In the example below, the total path travel is six inches. At five inches, motion must stop until input 3 goes active. During this time, output 2 must be asserted. When input 3 goes active, output 2 must go off and motion then resumes for the final inch.

```
DEF BISEG
PAB1                ; Use absolute positions
PA10                ; Path acceleration of 10
PAD10              ; Path deceleration of 10
PV5                ; Starting path velocity of 5
PVF0               ; Ending segment velocity of 0
POUT.2-0           ; Start with onboard output 2 off
PLIN5,0            ; 5 units right
POUT.2-1           ; Output 2 on during dwell
PGOWHN(IN.3=B1)    ; Dwell until onboard input 3 is active
POUT.2-0           ; Output 2 off during motion
PLIN6,0            ; Another unit right to finish
END
```

Segment End-point Coordinates

Stepper axes: All end-point position specifications are in units of commanded counts.

The end-point position specifications of lines and arcs can be either absolute or incremental. The controller stores the end-point data for all of its compiled segments internally as incremental, relative to the start of the segment. But in order to ease the programming task, absolute coordinates and multiple coordinate systems can be used.

When incremental coordinates are used to specify an end-point, the X and Y end-point values represent the distances from the X and Y start point of the segment being specified. Center specifications of an arc are always incremental (i.e., relative to the start of that arc segment). When absolute coordinates are used to specify an end-point, the X and Y end-point values represent that segment's position in the specified coordinate system. Incremental and absolute programming are specified with the PAB command. Incremental programming is the default state at the beginning of a path definition.

Coordinate systems allow the assignment of an arbitrary X-Y position as a reference position for subsequent absolute end-point specifications. The controller allows the use of two coordinate systems for use with absolute coordinate programming. These are called the *Work* coordinate system and the *Local* coordinate system. These are specified with the PWC and PLC commands. Neither coordinate system needs to represent the actual absolute position of the axes when the path actually executes.

The Work and Local coordinate systems are provided to allow absolute end-point definition of a segment without needing to know the actual position of each axis when the segment is executed. If no PWC command precedes the first segment command when a path definition begins, the controller will place the start of the first segment at location (0,0) in the Work coordinate system. By using the PWC *xpos, ypos* command, the programmer defines subsequent absolute end-points to refer to the Work coordinate system, and also locates that coordinate system such that the starting position of the next segment is at (xpos, ypos) of the Work coordinate system.

The Local coordinate system is provided so that if a section of a path is to appear in multiple locations along the path, the segments that compose that section can be programmed in absolute coordinates. By using the PLC *xpos, ypos* command, the programmer defines subsequent absolute end-points to refer to the Local coordinate system, and also locates that coordinate system such that the starting position of the next segment is at (xpos, ypos) of the Local coordinate system.

A single path definition can include both absolute and incremental programming, and be required to switch between Work and Local coordinates several times. At any point along a path definition, coordinates can be switched from absolute to incremental, or from incremental to absolute. When switching to absolute, all subsequent end-point specifications are assumed to be absolute with respect to the coordinate system in effect at that time. This

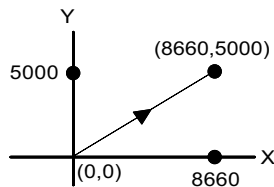
remains true until the reference system is switched to incremental, or to a new absolute reference.

When switching from Work coordinates to Local coordinates, the Local X and Y start positions of the following segment must be specified with the PLC command. When starting a path definition with Work coordinates, or when switching to Work coordinates, the starting position of the next segment can either be specified or assumed. The controller toggles between the Work coordinate system and the Local coordinate system with the PL command.

Ease of programming results from the ability to switch between absolute and incremental, and to re-define the coordinate systems between sections of a path. This allows individual sections of path definition to have Local coordinate systems, yet still be integrated into the complete path.

Line Segments

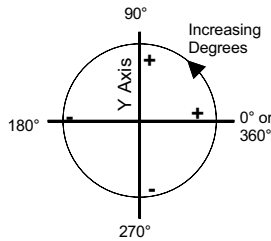
Lines are the simpler of the two path segment types. The placement, length, and orientation of the line is completely specified by the end-point of the line segment and the end-point of the previous segment. As described above, end-points can be specified with absolute or incremental coordinates.



The example below is specified with incremental coordinates and results in a line segment 10,000 steps in length, at 30 degrees in the X-Y plane.

`PLIN8660,5000 ; Line segment to (8660,5000) – see illustration at left`

Arc Segments



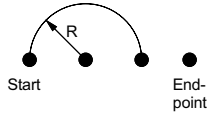
Arcs are more complex to specify than lines, because there are four possible ways to get from the start point to the end point. The radius of an arc can either be specified directly or implied by the center specification. In the controller, all path descriptions refer to the X-Y plane. The general convention describing the X-Y plane, as viewed from a drawing, is as follows. The X axis is shown as the left-right axis, with left being negative and right being positive. The Y axis is the up-down axis with down being negative and up being positive. Angles start at zero and increase in the CCW direction of rotation. A line segment, or the radius of an arc is at zero degrees if the incremental end-point has a positive X component and zero Y component. The angle is 90 degrees if the end-point has a positive Y component and zero X component.

Radius Tolerance Specifications

All arcs have an associated radius. In the controller, the radius can either be specified explicitly, or implied by a center specification. In both cases, it is possible that the radius cannot be consistent with the specified end-point of the arc. This could be a result of improper specification, user calculation error, or of round-off error in the internal arithmetic of the controller. For this reason, the controller allows the specification of a radius tolerance (PRTOL). The radius tolerance is specified in the same units as the radius and X and Y data.

The radius tolerance has a factory default of \pm one step, which is just enough to overcome round-off errors. The radius tolerance can be specified at any point along the path definition, and can be changed between one arc and the next. Each arc definition will be compared to the most recently specified radius tolerance. The radius tolerance should be about the same as the dimension tolerances of the finished product. The following paragraphs explain how the radius tolerance is used for the two types of arc specifications, and gives syntax examples for the radius tolerance specification.

Radius Specified Arcs

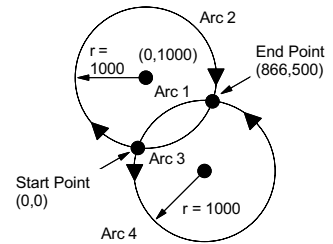


Specification of an arc using the radius method requires knowledge of the start point, the end point, and the sign and magnitude of the radius. The controller knows the start point to be either the start of the path, or the end of the previous segment. The end point and radius are provided by the user's program. It is possible to specify an impossible arc by specifying an end point that is more than twice the radius away from the start point (see drawing at left). In this case, the controller will automatically extend the radius to reach the end-point, provided that the automatic radius change does not exceed the user specified radius tolerance. If the required radius extension exceeds the radius tolerance, the controller will respond with an execution error, and no arc will be generated.

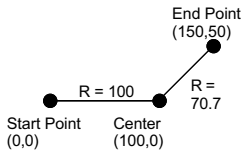
Example

The following illustration shows the four possible ways to move from the start point to the end point using an arc of radius 1000. Arc 1 and 2 both travel in the CW direction, arc 3 and 4 both travel in the CCW direction. Arc 1 and 3 are both less than 180 degrees. An arc of 180 degrees or less is specified with a positive radius. Arc 2 and 4 are both greater than 180 degrees. An arc of more than 180 degrees is specified with a negative radius. The example code below shows the radius tolerance specification and the specifications of arcs 1, 2, 3, and 4 respectively.

```
DEF arcs          ; Begin definition of path arcs
PRTOL5           ; 5 steps of radius tolerance
PARCP866,500,1000 ; Arc 1, CW < 180 degrees
PARCP866,500,-1000 ; Arc 2, CW > 180 degrees
PARCM866,500,1000 ; Arc 3, CCW < 180 degrees
PARCM866,500,-1000 ; Arc 4, CCW > 180 degrees
END              ; End path definition
;*****
;* To compile the arcs path, type "PCOMP arcs" *
;* To run the arcs path, type "PRUN arcs" *
;*****
```



Center Specified Arcs



Specifying an arc, using the center method, requires knowledge of the start point, the end point, and the center point of the arc. The X coordinate of the center is referred to with the letter I, and the Y coordinate of the center is referred to with the letter J. When an arc is specified with the center, another potential problem arises.

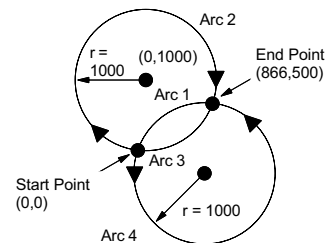
It is possible to specify the center of an arc such that the radius implied by the start point does not equal the radius implied by the end point (see illustration on left). In this case, the controller will re-locate the center so that the resulting arc has a uniform radius and the starting and ending angles come as close as possible to those implied by the user's center specification. This automatic center relocation will take place only if the start point and end point radius difference does not exceed the user specified radius tolerance. If the radius tolerance is exceeded, an execute error will result, and the arc will not be included in the path.

While automatic center relocation will ensure a continuous path, it can result in an abrupt change in path direction. This happens because a new location for the center results in a new tangent direction for an arc about that center.

Example

The example code below shows the specifications of arcs 1, 2, 3, and 4 for the drawing on the right. In the 6K commands, the order of the data is X, Y, I, J from left to right.

```
DEF arcs2         ; Begin definition of path
PARCOP866,500,866,-500 ; Arc 1, CW < 180 degrees
PARCOP866,500,0,1000 ; Arc 2, CW > 180 degrees
PARCOM866,500,0,1000 ; Arc 3, CCW < 180 degrees
PARCOM866,500,866,-500 ; Arc 4, CCW > 180 degrees
END              ; End path definition
;*****
;* To compile the arcs2 path, type "PCOMP arcs2" *
;* To run the arcs2 path, type "PRUN arcs2" *
;*****
```

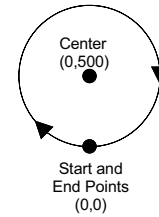


Circles

A circle is a special case of an arc whose end-point is the same as the starting point. Because these two points are the same, it is impossible to determine the location of the circle's center from a radius specification. For this reason, an arc that is a complete circle must be specified using the arc center specification method. An arc with identical starting and ending points specified with the radius method will be ignored. The circle shown below is specified with the example below.

Example

```
DEF circle      ; Begin definition of path circle
PARCOP0,0,0,500 ; Circle with center at (0,500)
END            ; End path definition
;*****
;* To compile the circle path, type "PCOMP circle" *
;* To run the circle path, type "PRUN circle"      *
;*****
```



Segment Boundary

So far, all the examples given have shown isolated line or arc segments. Most paths will consist of many segments put together. The point at which the segments are connected is called a *segment boundary* in this text. The controller automatically ensures that the path is continuous, in that segments are placed end-to-end.

The path velocity can either be constant or change from segment to segment, according to user specification. Velocity changes use the specified acceleration and deceleration and can take place even across segment boundaries.

The programmer should ensure that direction of travel is also continuous across segment boundaries (see Figure A). If the direction change is abrupt (as shown in Figure B) the X and Y axes will suffer abrupt acceleration or deceleration. The controller ensures that there will be no abrupt direction change within a segment, but the programmer is responsible for ensuring that the direction is continuous across segment boundaries. At low speeds, some motor and mechanical configurations will tolerate such abrupt changes, and the controller will accept such a program; however, it is generally good practice to design paths with smooth direction changes. This can be done by designing a path using arcs to round corners.

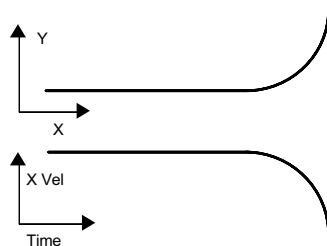


Figure A (Segment Example)

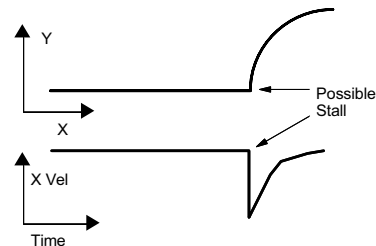


Figure B (Stall Example for stepper axis)

Using the C Axis *(products with ≥ 4 axes)*

The *C axis* is an axis whose position changes in a manner linearly related to the direction of travel in X and Y (i.e., the path direction). The C axis would be used in applications that require a work piece or tool to remain tangent or perpendicular to the path direction. Examples would be: a knife always pointing into the cut, or a welding head staying normal to the weld.

The magnitude of the C axis resolution refers to the number of steps of C axis position change for 360 degrees of direction change in the X-Y plane. This number can be the same as, or different from the C axis motor resolution, allowing any gearing that is convenient for the mechanics. If the C axis load is to be driven directly, the C axis resolution should be the same as the C axis motor resolution. This will cause the C axis motor and load to rotate once when a circle is drawn by the X and Y axes. If the C axis load is to be geared (e.g., 5:1), the C axis resolution specifications should be five times the C axis motor resolution. This will cause the actual motor to rotate five times and the load to rotate once when a circle is drawn by the X and Y axes.

The number can be positive or negative, allowing greater flexibility in C axis motor mounting orientation. If the sign is positive, the C axis will rotate in the positive direction when CCW arcs are drawn. If the sign is negative, the C axis will rotate in the negative direction when counter-clockwise arcs are drawn.

The C axis is assumed to be in the proper position when path execution begins. It will change position only as the direction of travel changes. The program must position the C axis before the path is executed. This can be done with the HOM command or a GO to a position.

Because the C axis position changes linearly with the direction of X-Y travel, it is important to avoid path definitions that result in an abrupt direction change between segments. The segment boundary considerations for the C axis are similar to those for the X and Y axes, except that abrupt direction changes will result in abrupt C axis position changes. The X and Y axis would only suffer large accelerations, which can cause a stall in stepper axes or exceed the maximum position error (SMPER value) in servo axes. The C axis will suffer impossibly high velocity commands, causing stall and position loss in steppers or position error in servos.

Using the P Axis *(products with ≥ 4 axes)*

The *P axis* is an axis whose position and velocity are proportional to the position and velocity traveled by the load along the path generated by X and Y. It can be used as the Z axis in helical interpolation, or to control other motion that must be proportional to the X-Y path motion. The proportionality of the P axis is specified as a ratio, with a range of ± 0.001 to ± 1000 . The sign of the ratio determines which direction the motor will turn. The magnitude specifies the ratio of P axis travel to path travel, regardless of path direction or segment type. This ratio is essentially a position ratio, but because the ratio is maintained at every instant it also becomes a velocity ratio.

The P axis only responds to the distance traveled along the path, and is not affected by direction changes in the path. The only caution that must be observed comes when a high ratio is specified. In this case, path velocity and acceleration are amplified, which can result in impossible velocities or stalls (stepper axes) or excessive position error (servo axes).

Outputs Along the Path

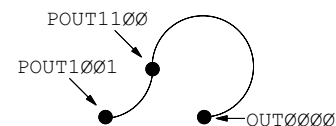
For each segment, you can also specify an output pattern (POUT), which is to be applied to the programmable outputs at the beginning of that segment and remains throughout that segment. These segment-defined output patterns are stored as part of the compiled path definition. These outputs will change state at some time between 1.5 ms before the beginning of the segment and 0.5 ms after the beginning of the segment. The programmable outputs cannot be controlled more precisely than this, because the controller updates its path position every 2 ms.

The path segment defined programmable outputs are provided so that plotting applications can raise and lower the pen, laser cutters can turn the laser on and off, glue applicators can be turned on and off, all at prescribed positions along the path. The output specification is stated before the segment definition, which holds that output state. In the example below, programmable outputs 2 and 4 are changed during the path segments.

```

Example Code
DEF prog1      ; Begin definition of path program prog1
POUT1001      ; Output pattern during first arc
PARCM5,5,5    ; Specify incremental X-Y endpoint position and
               ; radius arc <180° for 1/4 circle CCW arc
POUT1100      ; Output pattern during second arc
PARCM5,-5,-5  ; Specify incremental X-Y endpoint
               ; position and radius arc >180° for
               ; 3/4 circle CW arc
END           ; End definition of prog1
PCOMP prog1   ; Compile path program prog1
PRUN prog1    ; Execute path program prog1
OUT0000      ; Turn off programmable outputs 1-4

```



Paths Built Using 6K Series Commands

When defining a path, the commands that specify all of the path definitions must be contained in a named block defining that path. Each path definition block has a unique name that is used to distinguish one path from another. Because the path definition is stored as a program, many different paths can be stored, each defined with a unique name. A path definition block begins with a DEF command (containing its name) and ends with an END command.

The controller offers a command to compile (PCOMP) a named path definition block, and a separate command to execute (PRUN) a named path. Once a named path is compiled, it can be executed repeatedly without delay.

Compiling the Path

A PCOMP command will cause the controller to find the named path definition block and compile the path described by those commands, even if that pathname had been previously compiled. The use of variables (VAR) as parameters in path definition statements allow the same basic path to be re-defined with slightly different sizes and shapes. They can also be used to conditionally include or omit sections of the path.

Designed to allow compile-time determination of path parameters, there can be cases when the controller should prompt the operator or host computer for the value to be used for path velocity or segment end-points. Alternatively, these values can be read with the READ or DAT commands, allowing multiple calls of a single subroutine to define similar path sections with different data values. Commands that retrieve this data would be placed within the path definition, and would only prompt for the information when the path is compiled (e.g., PV (READ1)).

*Example
Code*

```

VARS1="PATH VELOCITY ? " ; Create message string
DEF path1                 ; Begin definition of path1
PAXES1,2                  ; Set path X & Y axes
PAB1                      ; Absolute path mode
PA100                    ; Path acceleration
PV(READ1)                 ; Path velocity, to be read in when compiling
PLIN25000,25000           ; Move in a line
PLIN(VAR2),(VAR3)         ; Move in a line, to be read in when compiling
END                       ; End definition of path1
PCOMP path1               ; Compile path1

```

Executing the Path

A PRUN command will cause the controller to find the named path definition block and execute the path described by those commands, if that pathname has already been compiled (PCOMP).

The use of variables as parameters in the path definition statement is a method of allowing segment parameters to take new values each time the path is compiled. When the path is executing, the values of the variables do not affect the path parameters. If a change in a variable value is intended to affect the path parameters, that path must be re-compiled. The PRUN command performs the equivalent of a GOSUB to the named path definition block.

Possible Programming Errors

It is possible to create a situation in which the segment statements are interrupted. This could occur if an enabled ON condition becomes true. If an enabled ON condition (ONCOND) becomes true while running a compiled path, the branch to the ONP program will result. Motion from the path that was being executed will continue at the last segment velocity until it is stopped. Within the ONP program, a Stop command should be issued for all axes to stop the path from executing. For more information on program interrupts (ON conditions), see page 29.

Programming Examples

Figure A and Figure B show two simple paths that illustrate most of the controller segment types. For both figures, axis 1 is X and axis 2 is Y. The C and P axes are not included.

Figure A specifies the end-points with absolute coordinates. The default Work coordinate system with start point of (0,0) is used, so no `PLØ` statement is needed.

Figure B specifies the end-points with incremental coordinates. The state of the programmable outputs needs to be different for Handles than for Knobs. No other controller actions take place during these paths.

```
6K Code  DEF HANDLE      ; Begin HANDLE path definition
          PAXES1,2    ; Set X axis as axis 1, and
          ; set Y axis as axis 2
          PAB1        ; Use absolute coordinates
          POUT1100    ; Programmable pattern for
          ; next segments
          PARCOM10,10,0,10 ; CCW quarter circle
          PLIN10,20   ; Vertical LINE segment
          PARCP20,10,-10 ; CW 3/4 circle
          PARCM20,0,5 ; CCW half circle
          END          ; End of HANDLE path definition
```

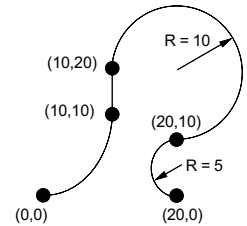


Figure A (HANDLE Example)

```
DEF KNOB      ; Begin KNOB path definition
PAXES1,2      ; Set X axis to be axis 1, and
              ; set Y axis to be axis 2
PABØ          ; Use incremental coordinates
POUT0011     ; Programmable pattern for next
              ; segments
PLIN30,0     ; Long LINE into circular knob
PARCOM0,0,0,10 ; CCW circle for the knob
PLIN10,0     ; Short LINE out of knob
END          ; End of KNOB path definition
```

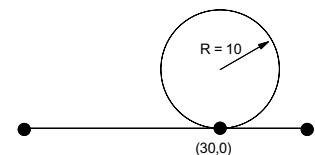


Figure A (KNOB Example)

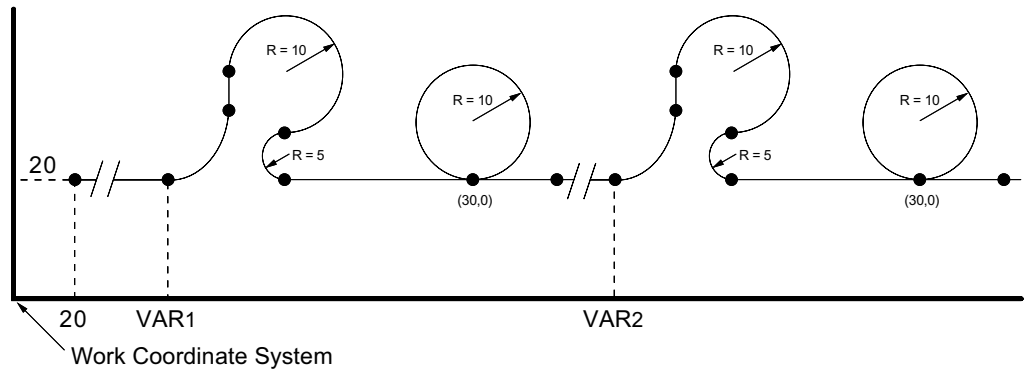
The third path consists of two pairs of the first two (see drawing below). Each pair is placed at variable locations within the Work coordinate system and the two pairs are connected with a Line segment. The Line leading into the first pair starts at (20,20) in the Work coordinate system. The first pair starts at (VAR1, 2Ø) and the second pair starts at (VAR2, 2Ø) in the Work coordinate system. Handle is defined using the Local coordinate system. Even though Handle is defined in absolute coordinates and appears in two different places along the path in PARTS, the statements describing it appear only once, in a path definition using local coordinates.

6K Code

```

DEF PARTS          ; Begin PARTS path definition
PAXES1,2          ; Set X axis to be axis 1, Y axis to be axis 2
PAB1              ; Use absolute coordinates
PWC20,20         ; Establish WORK coordinates
PLO              ; Enable WORK coordinates
PLIN(VAR1),20    ; LINE to (VAR1,20)
PLC0,0           ; Specify LOCAL coordinate system
PL1              ; Enable LOCAL coordinate system
PAB1              ; Use absolute coordinates
POUT1100        ; Programmable pattern for next segments
PARCOM10,10,0,10 ; CCW quarter circle
PLIN10,20       ; Vertical LINE segment
PARCP20,10,-10  ; CW 3/4 circle
PARCM20,0,5     ; CCW half circle
PAB0              ; Use incremental coordinates
POUT0011        ; Programmable pattern for next segments
PLIN30,0        ; Long LINE into circular knob
PARCOM0,0,0,10  ; CCW circle for the knob
PLIN10,0        ; Short LINE out of knob
PAB1              ; Use absolute coordinates
PLO              ; Return to WORK coordinates
PLIN(VAR2),20   ; LINE to (VAR2,20)
PLC0,0           ; Specify LOCAL coordinate system
PL1              ; Enable LOCAL coordinate system
PAB1              ; Use absolute coordinates
POUT1100        ; Programmable pattern for next segments
PARCOM10,10,0,10 ; CCW quarter circle
PLIN10,20       ; Vertical LINE segment
PARCP20,10,-10  ; CW 3/4 circle
PARCM20,0,5     ; CCW half circle
PAB0              ; Use incremental coordinates
POUT0011        ; Programmable pattern for next segments
PLIN30,0        ; Long LINE into circular knob
PARCOM0,0,0,10  ; CCW circle for the knob
PLIN10,0        ; Short LINE out of knob
END              ; End of PARTS path definition

PCOMP PARTS      ; Compile PARTS path definition
    
```



Compiled Motion Profiling

6K Series products allow you to construct complex motion profiles for each individual axis. The profiles can contain:

- Sequences of motion
- Loops
- Programmable output changes
- Embedded dwells
- Direction changes
- Trigger functions

Related Commands: Brief descriptions of related commands are found on page 136. For detailed descriptions, see the *6K Series Command Reference*.

⇒
Contouring path segments and *PLCP* programs are also stored in compiled memory (see page 11).

Compiled motion profiles are defined like programs (using the `DEF` and `END` commands); the commands used to construct the motion profile segments are stored in a program (stored in *Program* memory). This program is then compiled (using the `PCOMP` command) and the compiled profile *segments* (`GOBUF`, `PLOOP`, `GOWHEN`, `TRGFN`, `POUTA`, `POUTB`, `POUTC`, and `POUTD` statements) from the program are stored in *Compiled* memory. (**HINT:** The `TDIR` command reports which programs are compiled as a compiled profile.) You can then execute the compiled profile with the `PRUN` command.

The amount of RAM allocated for storing compiled profile segments is determined by the `MEMORY` command setting. The list below identifies memory allocation defaults and limits for 6K Series products. Further details on re-allocating memory are provided on page 11.

Total memory (bytes)	300,000
Default allocation (program,compiled)	<code>MEMORY150000</code> , 150000
Maximum allocation for compiled profiles	<code>MEMORY1000</code> , 299000
Maximum No. of compiled profiles	300
Maximum No. of compiled profile segments	2069

CAUTIONS

- Issuing a memory allocation command (e.g., `MEMORY70000`, `230000`) will erase all existing programs and compiled path segments. However, issuing the `MEMORY` command by itself (i.e., `MEMORY`—to request the status of how the memory is allocated) will not affect existing programs or segments.
- After compiling (`PCOMP`) and running (`PRUN`) a compiled profile. The profile segments will be deleted from *compiled* memory if you cycle power or issue a `RESET` command.

After compiling (`PCOMP`), you can execute the profiles with the `PRUN` command, and all of the motion and functions compiled into the profile are executed without any further commands during profile execution.

For multi-axis products, profiles on any combination of axes can be launched simultaneously with a single `PRUN` command. This provides a very powerful method of synchronizing the action of multiple axes with very simple programming. For example, in a four-axis product, one axis could be running a complex Following profile, while two other axes are contouring, and the fourth could be performing a multi-tiered velocity motion profile.

Because the motion and functions are *pre-compiled*, delays associated with command processing are eliminated during profile execution, allowing more rapid sequencing of actions than would be possible with programs that are not compiled. Command processing is then free to monitor other activities, such as I/O and communications.

NOTES

- During compilation (PCOMP), most commands are executed the same as if no profile were being defined, even those that are not relevant to the construction of a profile. This is also true of non-compiled motion commands embedded in a compiled motion program during PCOMP. For this reason, it's good to limit commands between DEF and END to those that actually assist in the construction of the profile. Even for those that do actually assist in the construction of the profile, such as A, V, and D, it is important to remember that the command is executed and data actually changes, and it is not restored after compilation is completed.
- If your compiled motion program contains variables, the variables are evaluation only at compile time.

Each motion segment in a compiled motion profile can have its own distance, velocity, acceleration, and deceleration, as shown in the program example below:

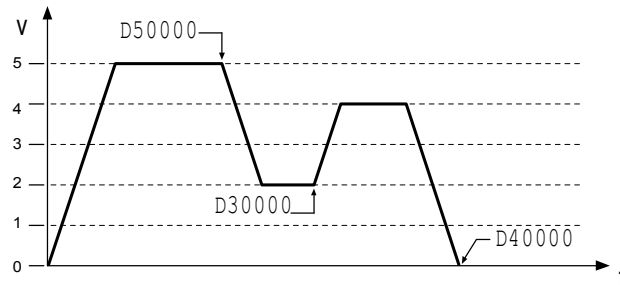
Programming Example

```

DEF simple          ; Begin definition of program simple
MCO                ; Preset positioning mode (disable continuous mode)
D50000             ; Distance is 50000
A10                ; Acceleration is 10
AD10               ; Deceleration is 10
V5                 ; Velocity is 5
GOBUF1             ; Store the first motion segment for axis 1.
                   ; Profile attributes are: MCO, A10, AD10, V5, D50000
D30000             ; Distance is 30000
V2                 ; Velocity is 2
GOBUF1             ; Store the second motion segment for axis 1
                   ; Profile attributes are: MCO, A10, AD10, V2, D30000
D40000             ; Distance is 40000
V4                 ; Velocity is 4
GOBUF1             ; Store the third motion segment for axis 1
                   ; Profile attributes are: MCO, A10, AD10, V4, D40000
                   ; Because this is the last segment in a preset profile,
                   ; the velocity will automatically end at zero.
END                ; End program definition

PCOMP simple       ; Compile simple
PRUN simple        ; Run simple
  
```

The resulting profile from the above program:



STATUS COMMANDS

Use these commands to check the status of compiled profiles.

System Status (TSSF, TSS, & SS):

- Bit 29 is set if compiled memory is 75% full.
- Bit 30 is set if compiled memory is 100% full.
- Bit 31 is set if a compile (PCOMP) failed; this bit is cleared on power-up, reset, or after a successful compile. Possible causes include:
 - Errors in profile design (e.g., change direction while at non-zero velocity, distance & velocity equate to <1 count/system update, preset move profile ends in non-zero velocity).
 - Profile will cause a Following error (see TFS, TFSF & FS status).
 - Out of memory (see system status bit 30)
 - Axis already in motion at the time of the PCOMP command
 - Loop programming errors (e.g., no matching PLOOP or PLN, more than 4 embedded PLOOP/END loops)

Axis Status (TASF, TAS, & AS): Bit 31 is set while compiled a GOBUF profile is executing.

Rules for Using
Velocity in Preset
Compiled Motion

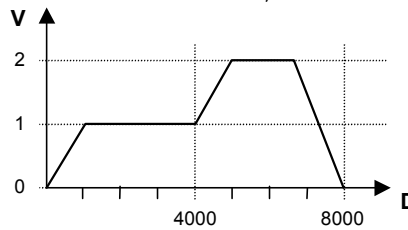
TSEG & SEG: Reports the number of available segments in compiled memory.

TDIR: Identifies programs that are “compiled as a path” (compiled with the PCOMP command) and reports the percentage of remaining compiled memory.

When defining preset mode (MC0) compiled profiles there are several rules that govern the velocity.

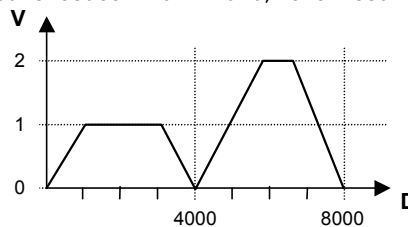
Rule 1: The last segment in the compiled profile will automatically end at zero velocity (only if not in a PLOOP/PLN loop).

```
DEF PROF5 ; Begin definition of profile 5
MC0 ; Select preset positioning
V1 ; Set velocity to 1 rev/sec
D4000 ; Set distance to 4000
GOBUF1 ; First motion segment (V1, D4000)
V2 ; Set velocity to 2 (second segment)
GOBUF1 ; Second motion segment (V2, D4000)
END ; End definition of profile 5
PCOMP ; Compile profile 5
; When you execute PRUN PROF5, the resulting profile is:
```



Rule 2: If you wish intermediate segments to end in zero velocity, use the VF0 command in the respective GOBUF segment.

```
DEF PROF6 ; Begin definition of profile 6
MC0 ; Select preset positioning
V1 ; Set velocity to 1 rev/sec
VF0 ; End this segment at zero velocity
D4000 ; Set distance to 4000
GOBUF1 ; First motion segment (V1, D4000, VF0)
V2 ; Set velocity to 2 (second segment)
VF0 ; End this segment at zero velocity
GOBUF1 ; Second motion segment (V2, D4000, VF0)
END ; End definition of profile 6
PCOMP ; Compile profile 6
; When you execute PRUN PROF6, the resulting profile is:
```



Rule 3: CAUTION: With compiled loops (PLOOP and PLN), the last segment within the loop must end at zero velocity or there must be a final segment placed outside the loop. Otherwise, when the profile is compiled with PCOMP you’ll receive the “ERROR: MOTION ENDS IN NON-ZERO VELOCITY-AXIS n” error message and system status bit 31 will be set. after the final segment is completed, the motor will continue moving at the last segment’s velocity.

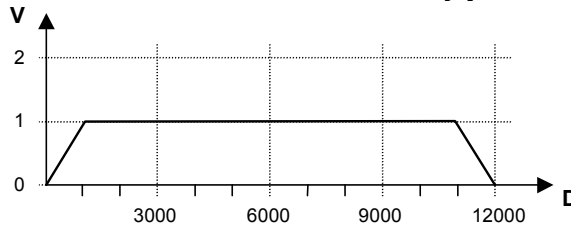
```
DEF PROF7 ; Begin definition of profile 7
MC0 ; Select preset positioning
D3000 ; Set distance to 3000
PLOOP4 ; Loop (between PLOOP & PLN) 4 times
V1 ; Set velocity to 1 rev/sec
GOBUF1 ; First motion segment
PLN1 ; End loop
END ; End definition of profile 7
PCOMP ; Compile profile 7, but instead of compile, you receive
; an error message
```

To fix the profile, reduce the PLOOP count by one and add a GOBUF statement after the PLN command:

```

DEF PROF7 ; Begin definition of profile 7
MCO      ; Select preset positioning
D3000    ; Set distance to 3000
PLOOP3   ; Loop (between PLOOP & PLN) 3 times
V1       ; Set velocity to 1 rev/sec
GOBUF1   ; Looped motion segment
PLN1     ; End loop
GOBUF1   ; Last motion segment (end at zero velocity)
END      ; End definition of profile 7
PCOMP    ; Compile profile 7
; When you execute PRUN PROF7, the resulting profile is:

```

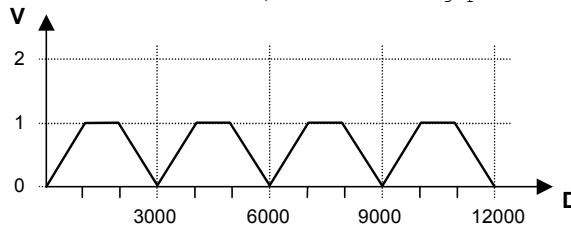


Rule 4: With compiled loops (PLOOP and PLN), if you wish the velocity at the end of each loop to end at zero, use a VF0 command.

```

DEF PROF8 ; Begin definition of profile 8
MCO      ; Select preset positioning
D3000    ; Set distance to 3000
PLOOP4   ; Loop (between PLOOP & PLN) 4 times
V1       ; Set velocity to 1 rev/sec
VF0      ; End each segment at zero velocity
GOBUF1   ; Looped motion segment
PLN1     ; End loop
END      ; End definition of profile 8
PCOMP    ; Compile profile 8
; When you execute PRUN PROF8, the resulting profile is:

```



Compiled Following Profiles

More details on Following are found in Chapter 6 (page 166).

The new FOLRNF command designates that the motor will move the load the distance designated in a preset GOBUF segment, completing the move at the specified final ratio. The only allowable value for FOLRNF is zero (0). FOLRNF is allowed for a segment only if the starting ratio is also zero, i.e., it must be the first segment, or the previous segment must have ended in zero ratio. FOLRNF is only useful with compiled preset Following moves because the starting and final ratios are already zero for motion initiated with GO.

Compiled motion profiles can be constructed with any combination of preset or continuous motion segments. A continuous (MC1) Following segment will start with the final ratio of the previous segment, and end with the ratio given by FOLRN and FOLRD. The motion segment will consist of one ramp from the starting ratio to the final ratio. Just as with continuous Following ramps outside of a compiled profile, the master travel over which the ramp takes place is specified with FOLMD. The slave travel over which the ramp takes place is simply the product of master travel and average ratio. Because the slave travel is not specified explicitly, it is possible for arithmetic round-off errors to cause actual slave travel during a ramp to differ

from theoretical calculations. For applications in which slave distance is important, preset segments should be used.

A preset (MCØ) Following segment will also start with the final ratio of the previous segment, but can end in one of two ways. FOLRNF specifies the final ratio of a preset Following segment. As previously described, the only valid value for FOLRNF is zero (0). If FOLRNFØ is given before the GOBUF, the resulting motion segment will be constructed exactly as preset Following moves are outside of compiled profiles. In this case, the starting ratio must be zero, the final ratio will be zero, and the maximum intermediate ratio will be given by FOLRN and FOLRD. The relationships between ratio, master distance, and slave distance for this case are given on page 195 under the heading *Master and Follower Distance Calculations*. The FOLRNF command affects only the immediately subsequent preset Following segment, and must be given explicitly for each preset segment that is to end in zero ratio.

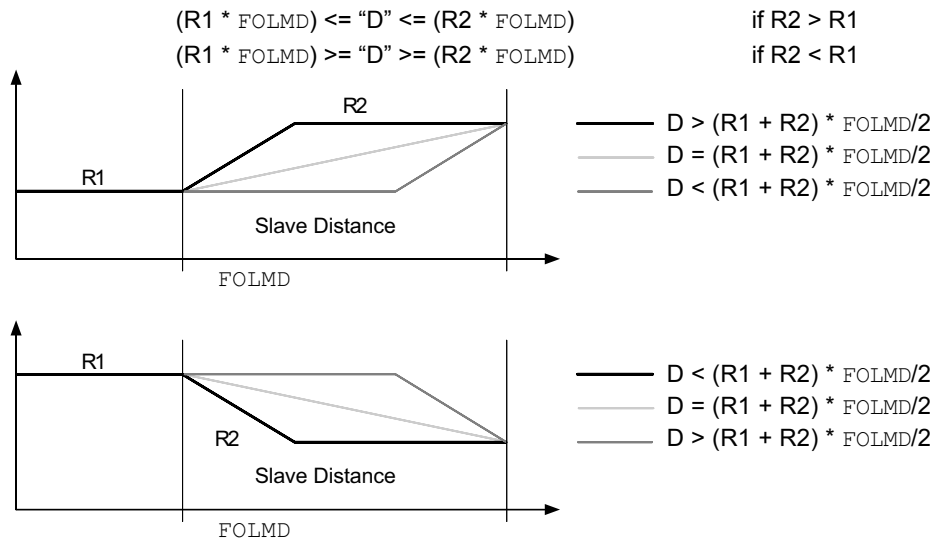
If FOLRNFØ is not given before the GOBUF, the segment will end with the ratio given by FOLRN and FOLRD, and need not start with zero ratio. This type of motion segment is constrained, however, to intermediate ratios that fall between the starting and final ratios.

Compiled profiles are built from motion segments created with the GOBUF command. For each individual axis, all motion segments in a compiled profile must use the same state of Following. That is, the motion segments can be all Following or all non-Following, but not a mixture of Following and non-Following (but at any point in time, separate axes can have different Following mode states).

The GOBUF command builds the appropriate type of motion segment based on the values of FOLMAS and FOLEN during compilation. These parameters cannot be changed inside a compiled program after a GOBUF. The choice of zero or non-zero FOLMAS must be the same during PRUN as during PCOMP (if non-zero, the value can be changed, but still must be non-zero). If a non-zero FOLMAS is given, the value of FOLEN must be the same during PRUN as during PCOMP.

Distance Calculations For Compiled Following Moves

The graph below shows 6 possibilities of ratio change profiles for preset segments, with legal FOLMD and “D” values constrained by the requirement that the average ratio (given by “D”/FOLMD) is between R1 and R2. If the distance is outside these ranges, in the profile used to get from R1 to R2 over FOLMD (covering “D” slave distance), an error message will be generated during the PCOMP command. For the graphs shown, the constraints are expressed by:



The two graphs above show the cases of $R1 < R2$ or $R1 > R2$, but the distance calculations of the ramp and constant ratio portions are the same for the two cases. For each graph, the heavy lined profile (first case) of these mimics the shape of the corresponding preset velocity change (FOLMD) segments in that the ramp takes place before the constant ratio portion. The second case occurs only if the distance specified exactly matches the start and end ratios and FOLMD1. In the third case, the ramp takes place after the constant ratio portion. In the first and third cases, only two segments are built, and the slave and master distances traveled in each segment are easily calculated with the simple formulas shown below. These formulas are based on positive ratios and master and slave distances. In the construction of Following profiles, ratios and master distances are always positive, with direction implied by the sign of the slave distance. For calculations with negative slave distances, simply use the magnitude of "D" in the formulas below, and invert the sign of the resulting slave distances.

Case 1 (Ramp first)	$MD1 = [D - (R2 * FOLMD)] / ((R1 - R2) / 2)$ $MD2 = FOLMD - MD1$ $D1 = .5 * (R1 + R2) * MD1$ $D2 = D - D1$	where MD1 = master distance during ramp where MD2 = master distance during flat where D1 = slave distance during ramp where D2 = slave distance during flat
Case 2 (Ramp only)	$MD1 = FOLMD$ $D1 = D$	where MD1 = master distance during ramp where D1 = slave distance during ramp
Case 3 (Ramp last)	$MD1 = [D - (R1 * FOLMD)] / ((R2 - R1) / 2)$ $MD2 = FOLMD - MD1$ $D1 = .5 * (R1 + R2) * MD1$ $D2 = D - D1$	where MD1 = master distance during ramp where MD2 = master distance during flat where D1 = slave distance during ramp where D2 = slave distance during flat

Dwells and Direction Changes

Compiled profiles can incorporate changes in direction only if the preceding motion segment has come to rest. This can be achieved for non-Following segments either by creating a continuous segment with a goal velocity of zero, or by preceding a preset segment with VFØ. It can be achieved for Following segments either by creating a continuous or preset segment with a goal ratio of zero, or by preceding a preset segment with FOLRNFØ. In all cases, motion within the profile comes to rest, although the profile is not yet complete. Even though the motor is not moving, the axis status bit 1 (AS . 1) will remain set, indicating a profile is still underway. Only then can you change direction (using the D+ or D- command, D~ is not allowed) within a profile. An attempt to incorporate changes in direction if the preceding motion segment has not come to rest will result in a compilation error.

In many applications, it can be useful to create a time delay between moves. For example, a machine cycle can require a move out, dwell for 2 seconds, and move back. To create this dwell, a compiled GOWHEN can be used between the two moves. The code within a compiled program can look like:

```

MCO           ; Preset incremental positioning used
D(VAR1)      ; Target position is in VAR1
VF0          ; Motion comes to rest at end of move
GOBUF1       ; Create move out segment
GOWHEN(T=2000) ; Profile delays for 2 seconds
D-           ; Return position is home (direction reversed)
VF0          ; Motion comes to rest at end of move
GOBUF1       ; Create move back home segment

```

In Following applications, it can be more useful to create a master travel delay between moves. For example, a machine cycle replacing a cam can require a move out, dwell for 2000

master counts, and move back. To create this dwell, a compiled GOBUF of zero slave distance can be used between the two moves. The code within a compiled program can look like:

```

MC0           ; Preset incremental positioning used
D(VAR1)      ; Target position is in VAR1
FOLMD4000    ; Move takes place over 4000 master counts
FOLRNFO      ; Motion comes to rest at end of move
GOBUF1       ; Create move out segment
D0           ; No change in target position
FOLMD2000    ; Dwell takes place over 2000 master counts
FOLRNFO      ; Motion comes to rest at end of "move" (dwell)
GOBUF1       ; Create dwell segment
D(VAR1)      ; Return position is home (direction change implied)
D-           ; Return position is home (direction reversed)
FOLMD4000    ; Move takes place over 4000 master counts
FOLRNFO      ; Motion comes to rest at end of move
GOBUF1       ; Create move back home segment

```

Compiled Motion Versus On-The-Fly Motion

The two basic ways of creating a complex profile are with compiled motion or with on-the-fly pre-emptive GO commands. With compiled motion, portions of a profile are built piece by piece, and stored for later execution. Compiled motion is appropriate for profiles with motion segments of pre-determined velocity, acceleration and distance. Compiled motion profiles allow for shorter motion segments, which results in faster cycle times because there is no command processing and execution delay. The axes can perform their own motion control and coordination, freeing program flow for other tasks, such as I/O, machine control, and host requests. The disadvantages to pre-defined compiled motion profiles are the amount of memory use and limited run-time decision making and I/O processing.

With pre-emptive GO moves, the motion profile underway is pre-empted with a new profile when a new GO command is issued. The new GO command constructs and launches the pre-empting profile. Pre-emptive GOs are appropriate when the desired motion parameters are not known until motion is already underway.

The table below summarizes the differences between the use of compiled motion and on-the-fly motion.

Command/Issue	Compiled Motion	On-The-Fly Motion
GOBUF	Constructs motion segment and appends to previously constructed segment	N/A
PRUN	Used to launch previously compiled motion	N/A
GO	GO causes move during PCOMP	GO Constructs & launches profile, even if moving
Direction changes	Only if previous motion segment comes to rest (MCØ & VFØ or MC1 & VØ), else compile error	Not allowed during motion, else AS. 3Ø, ER. 1Ø
Insufficient room for AD (decel) value	Same as on-the-fly	Decel is modified (steppers); Motion is killed, AS. 3Ø (servos)

Related Commands

GOBUF

Store a Motion Segment in Compiled Memory:

The GOBUF command creates a motion segment as part of a profile and places it in a segment of compiled memory, to be executed after all previous GOBUF motion segments have been executed. An individual axis profile is constructed by sequentially appending motion segments using GOBUF commands. Each motion segment can have its own distance to travel, velocity, acceleration, and deceleration.

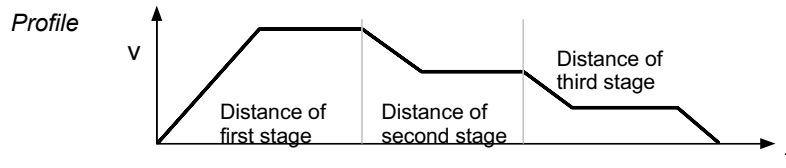
The end of a GOBUF motion segment in preset mode is determined by the distance or position specified. The end of a GOBUF motion segment in continuous mode is determined by the goal velocity specified. In both cases, the final velocity and position achieved by a segment will be the starting velocity and position for the next segment. If either type of segment is followed by

	a GOWHEN command, the segment's final velocity will be maintained until the GOWHEN condition becomes true.
PLOOP & PLN	<p><i>Loop Start & Loop End (Compiled Motion only):</i></p> <p>The PLOOP and PLN commands specify the beginning and end of an axis-specific profile loop, respectively. All segments defined between the PLOOP and PLN commands are included within that loop.</p>
VF & FOLRNF	<p><i>Final Velocity & Numerator of Slave-to-Master Final Ratio:</i></p> <p>The VF and FOLRNF commands are used to designate that the motor will move the load the distance designated in a preset GOBUF motion segment, completing the move at a final speed of zero. The VF command is used when the Following mode is disabled (FOLENØ). The FOLRNF command is used when the Following mode is enabled (FOLEN1).</p>
GOWHEN	<p><i>Conditional GO:</i></p> <p>When GOWHEN is compiled in a profile, the GOWHEN condition is stored as part of that profile instead of being executed immediately. When progress through the profile reaches the compiled GOWHEN, AS . 26 is set, and the next segment's execution will be suspended until the GOWHEN condition becomes true. This allows subsequent GOWHEN and GOBUF combinations to be issued and stored, instead of overriding each other.</p>
TRGFN	<p><i>Trigger Functions:</i></p> <p>When TRGFN is compiled in a profile, the TRGFN condition is stored as part of that profile instead of being executed immediately. When progress through the profile reaches the compiled TRGFN, the embedded trigger functions are assigned to that trigger. AS . 26 is set if the GOWHEN function has been assigned to the trigger, and the next segment's execution will be suspended until the specified trigger input goes active. This allows subsequent TRGFN, GOWHEN, and GOBUF combinations to be issued and stored, instead of overriding each other.</p>
PCOMP, PRUN & PUCOMP	<p><i>Compile a Program, Run a Compiled Program, & Un-Compile a Compiled Program:</i></p> <p>The PCOMP, PRUN, and PUCOMP commands allow you to incorporate individual axis profiles within compiled motion profiles. Compiled motion for the 6K series allows you to construct complex motion programs using an individual contour (a series of arcs and lines), individual axis profiles (a series of GOBUF commands), or a path (combination of contours and individual axis profiles).</p>
PEXE	<p><i>Execute Compiled GOBUF Profile from PLC Program:</i></p> <p>You can place a PEXE command inside a compiled PLCP program. When the PLCP program is scanned in the PLC Scan Mode (SCANP), the PEXE command launches the specified compiled profile in another task. For details on the PLC Scan Mode, see page 104.</p>
POUTA through POUTH	<p><i>Output During Compiled Motion Profile — Axes Specific:</i></p> <p>The POUTA, POUTB, POUTC, POUTD, POUTE, POUTF, POUTG, and POUTH commands turn the programmable output bits on and off for axes 1, 2, 3, 4, 5, 6, 7 and 8, respectively.</p>
TSEG & SEG	<p><i>Transfer/Display (TSEG) or Assign (SEG) the Number of Free Segment Buffers:</i></p> <p>The TSEG command returns the number of free segment buffers in compiled memory. The SEG command is used to assign the number of free segment buffers in compiled memory to a variable or to make a comparison against another value.</p>

Compiled Motion — Sample Application 1

A manufacturer has an application where wire is being wrapped onto a spindle. There is a motor controlling the rotational speed of the spindle. Every application of the spindle requires that the motor runs at a fast speed with a slow acceleration for the first few revolutions, a medium speed for the next couple of revolutions, and a slower speed as the spindle gets fuller to maintain somewhat of a constant velocity off the feed wire. The technician would like to

use an RP240 to enter the velocity and number of revolutions for each stage of winding. Programmable outputs 1, 2 and 3 are wired to status LEDs, and should go on for the respective stages of winding (output 1 for stage 1, etc.).



```

Program  DEF PROFIL          ; Define motion profile program
          VAR10 = 4000 * VAR4 ; Get distance of first stage
          ; (assuming 4000 steps/revolution)
          D(VAR10)           ; Set distance
          V(VAR1)            ; Set velocity of first stage
          POUTA.1-1         ; Turn output 1 on
          GOBUF1             ; Build motion
          VAR10 = 4000 * VAR5 ; Get distance of second stage
          D(VAR10)           ; Set distance
          V(VAR2)            ; Set velocity of second stage
          POUTA01           ; Turn output 1 off and output 2 on
          GOBUF1             ; Build motion
          VAR10 = 4000 * VAR6 ; Get distance of third stage
          D(VAR10)           ; Set distance
          V(VAR3)            ; Set velocity of third stage
          POUTAx01          ; Turn output 2 off and output 3 on
          GOBUF1             ; Build motion
          POUTA.3-0         ; Turn off output 3
          END                ; End motion profile program

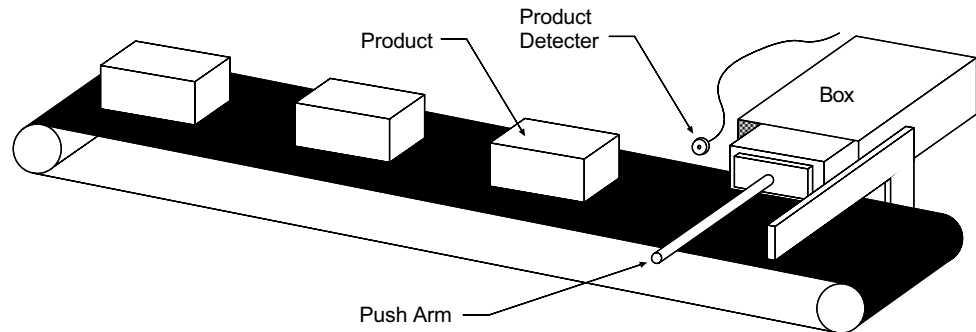
          DEF EXMPL1         ; Define program example 1
          L                  ; Continual loop of program execution
          DCLEAR0            ; Clear all lines on RP240 display
          DPCUR1,1           ; Position cursor at line 1, column 1
          DWRITE"ENTER VELOCITY STAGE 1"; Prompt user
          VAR1 = DREAD        ; Get 1st velocity from RP240 entry
          DCLEAR1            ; Clear line 1 on RP240 display
          DPCUR1,1           ; Position cursor at line 1, column 1
          DWRITE"ENTER VELOCITY STAGE 2" ; Prompt user
          VAR2 = DREAD        ; Get 2nd velocity from RP240 entry
          DCLEAR1            ; Clear line 1 on RP240 display
          DPCUR1,1           ; Position cursor at line 1, column 1
          DWRITE"ENTER VELOCITY STAGE 3" ; Prompt user
          VAR3 = DREAD        ; Get 3rd velocity from RP240 entry
          DCLEAR1            ; Clear line 1 on RP240 display
          DPCUR1,1           ; Position cursor at line 1, column 1
          DWRITE"ENTER REVOLUTIONS STAGE 1" ; Prompt user
          VAR4 = DREAD        ; Get # of windings 1st stage from RP240 entry
          DCLEAR1            ; Clear line 1 on RP240 display
          DPCUR1,1           ; Position cursor at line 1, column 1
          DWRITE"ENTER REVOLUTIONS STAGE 2" ; Prompt user
          VAR5 = DREAD        ; Get # of windings 2nd stage from RP240 entry
          DCLEAR1            ; Clear line 1 on RP240 display
          DPCUR1,1           ; Position cursor at line 1, column 1
          DWRITE"ENTER REVOLUTIONS STAGE 3" ; Prompt user
          VAR6 = DREAD        ; Get # of windings 3rd stage from RP240 entry
          PCOMP PROFIL        ; Re-compile profile with new vel/dist info
          $AGAIN              ; Label for repeating same profile
          PRUN PROFIL         ; Execute profile
          DCLEAR1            ; Clear line 1 on RP240 display
          DPCUR1,1           ; Position cursor at line 1, column 1
          DWRITE"SAME DATA (1=YES,2=NO)" ; Prompt user if perform again with old data
          VAR7 = DREAD        ; Get response
          IF(VAR7=1)          ; If user wants to perform same profile
            GOTO AGAIN        ; perform again
          NIF                  ; End conditional
          LN                  ; End command execution loop
          END                ; End definition program example 1

          ; *****
          ; * To begin, execute the EXMPL1 program *
          ; *****

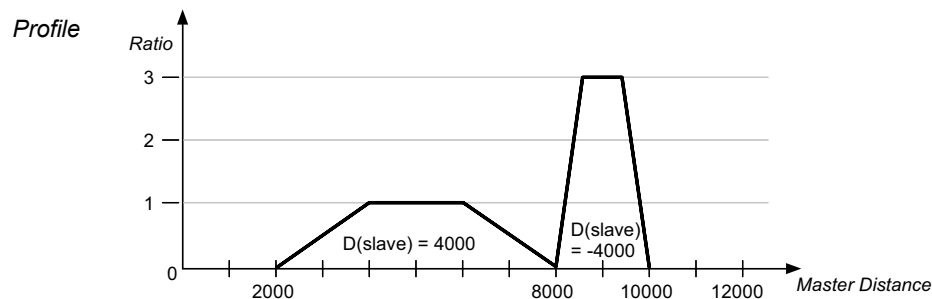
```

Compiled Motion — Sample Application 2

Here's an example of replacing a mechanical cam using a compiled Following profile. There is evenly spaced product coming in on a feeder belt. The feeder belt can vary in speed. The cam that you are replacing controls a push arm that will push the product into a box for shipping. You would also like the arm to retract at a faster rate than it extends. In other words, you would like to have a smooth push to load and a fast retract to set up for the next product. Since this is a cam, this profile must repeat continuously for each product or master cycle but won't start until the first product is detected.



The feeder belt is the master and the master cycle length (space from the front of one product to the front of the next) is 12000 master (encoder) counts on the feeder belt. The push of the product will start 2000 counts into the master cycle. The push will take place over 6K master counts, and the retract over 2000 master counts. The distance the push arm (slave) must travel is 4000 counts. Assume the detector is wired to trigger 1A (onboard trigger input 1). Below is a graph of this Following profile.



Program

```

; Setup code
FOLMAS21      ; Master is coming in on encoder 2
FOLEN1       ; Enable Following
INFNC1-H     ; Assign onboard input 1 (TRG-1A) as trigger interrupt

; Motion program
DEL EXPL2    ; Delete program (in case it already exists in memory)
DEF EXPL2    ; Begin definition of program example 2
1TRGFNA1    ; Launch axis 1 profile upon receiving TRG-1A
             ; (1st product detected)
PLOOP0      ; Loop continuously to mimic a mechanical cam

; Program first move - dwell
FOLRN1      ; Set up ratios - numerator
FOLRD1      ; and denominator
FOLMD2000   ; Over a distance of 2000 master steps
D0          ; slave will not move
FOLRNFO     ; and end at zero ratio
GOBUF1     ; Build motion
    
```

```

; Program second move - positive slave move
FOLMD6000      ; Over a distance of 6000 master steps
D4000          ; slave will move 4000 steps
FOLRNFO        ; and end at zero ratio
GOBUF1         ; Build motion

; Program third move - negative slave move
FOLRN3         ; New ratio to accommodate larger distance of slave travel
FOLMD2000      ; Over a distance of 2000 master steps
D-4000         ; slave will move -4000 steps
FOLRNFO        ; and end at zero ratio
GOBUF1         ; Build motion

; Program last move - dwell
FOLMD2000      ; Over a distance of 2000 master steps
D0             ; slave will not move
FOLRNFO        ; and end at zero ratio
GOBUF1         ; Build motion
PLN1           ; Close cam loop
END            ; End program example 2

PCOMP EXPL2    ; Compile program EXPL2

; ** To execute the program, enter the PRUN EXPL2 command **

```

**Program
Modification**

Let's now modify the constraints of the system. Let's say that the product will be spaced roughly 12000 master counts apart. It may or may not be exactly 12000, but it will never be less than 10000 (just to make sure the retraction finishes before the next product is detected). We can then modify the program to wait for the product to be detected each cycle. We can also take the extra "dwell" or zero distance move out of the end of the profile. See program below:

```

; Setup code
FOLMAS21       ; Master is coming in on encoder 2
FOLEN1         ; Enable Following mode
INFNC1-H       ; Assign onboard input 1 (TRG-1A) as trigger interrupt

; Motion program
DEL EXPL2B     ; Delete program (in case it already exists in memory)
DEF EXPL2B     ; Begin definition of program example 2b
PLOOP0        ; Loop continuously to mimic a mechanical cam
1TRGFNA1      ; Pause axis 1 profile until TRG-1A is activated
               ; (detect next product)

; Program first move - dwell
FOLRN1         ; Set up ratios - numerator
FOLRD1         ; and denominator
FOLMD2000      ; Over a distance of 2000 master steps
D0             ; slave will not move
FOLRNFO        ; and end at zero ratio
GOBUF1         ; Build motion

; Program second move - positive slave move
FOLMD6000      ; Over a distance of 6000 master steps
D4000          ; slave will move 4000 steps
FOLRNFO        ; and end at zero ratio
GOBUF1         ; Build motion

; Program third move - negative slave move
FOLRN3         ; New ratio to accommodate larger distance of slave travel
FOLMD2000      ; Over a distance of 2000 master steps
D-4000         ; slave will move -4000 steps
FOLRNFO        ; and end at zero ratio
GOBUF1         ; Build motion
PLN1           ; Close cam loop
END            ; End program example 2b

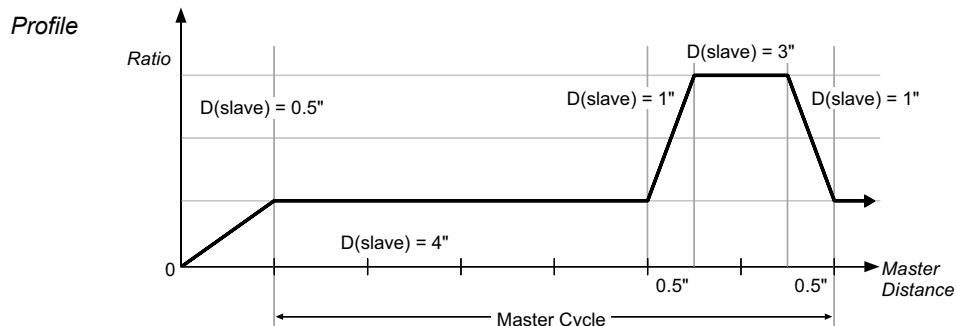
PCOMP EXPL2B   ; Compile program EXPL2B

; *****
; * To execute the program, enter the PRUN EXPL2B command *
; *****

```

Compiled Motion — Sample Application 3

In this application, there is a wheel that stamps a logo onto the product. The product is assumed to be entering at a constant and fixed spacing, each product is 4 inches in length with 2 inches separating each unit. The stamp wheel has a circumference of 9 inches, and must be traveling at a 1 to 1 ratio with the product at the time of stamping. The stamp wheel must then travel five inches in just 2 inches of master travel. There is a sensor wired to trigger A of the 6K controller to detect the first product and start the cycling. At the time of the trigger the product is 1 inch away from contact with the stamp wheel. Assume that the home position of the slave is 0.5 inches away from a stamp. The mechanics of the system give 3000 steps of master travel per inch and 1500 steps of slave travel per inch.



As you can see above, we have a multi-tiered Following profile. By multi-tiered we mean that ratio is changing from a non-zero value to another non-zero value. To program this profile effectively, we will break the profile into pieces as shown with the dotted lines in the above illustration:

```

Program FOLMAS21      ; Define the master as encoder on axis 2
FOLEN1      ; Enable Following
INFNC1-H    ; Assign onboard input 1 (TRG-1A) as trigger interrupt
SCLMAS3000  ; Set scaling of master steps per inch
SCLD1500    ; Set scaling of slave steps per inch
SCALE1      ; Enable scaling

DEF EXMPL3  ; Start definition of example program 3
1TRGFNA1   ; Launch axis 1 profile when TRG-1A is activated

; Program first ramp from ratio 0 to ratio 1
FOLRD1     ; Set Following ratio - denominator
FOLRN1     ; Set the Following ratio at 1 to 1
FOLMD1     ; Over a master distance of 1"
D0.5       ; Slave will travel 0.5"
GOBUF1     ; Build motion

PLOP0      ; Start the continuous loop

; Program constant ratio
FOLRN1     ; At a 1 to 1 ratio
FOLMD4     ; Over a master distance of 4"
D4         ; Slave will travel 4"
GOBUF1     ; Build motion

; Program ramp to new ratio
FOLRN3     ; Go to a 3 to 1 ratio
FOLMD0.5   ; Over a master distance of 0.5"
D1         ; Slave will travel 1"
GOBUF1     ; Build motion
; Program second constant ratio
FOLRN3     ; At a 3 to 1 ratio
FOLMD1     ; Over a master distance of 1"
D3         ; Slave will travel 3"
GOBUF1     ; Build motion

; Program ramp to lower ratio
FOLRN1     ; Go to a 1 to 1 ratio
FOLMD0.5   ; Over a master distance of 0.5"

```

```

D1                ; Slave will travel 1"
GOBUF1            ; Build motion
PLN1              ; Close motion loop

; Define the exit motion
FOLRN0           ; Stop slave at zero ratio (and zero velocity)
FOLMD1           ; Over a master distance of 1"
D0.5             ; And a slave distance of 0.5"
GOBUF1           ; Build motion
END              ; End definition of example program 3

PCOMP EXMPL3     ; Compile example program 3

; *****
; * To execute the program, enter the PRUN EXMPL3 command *
; *****

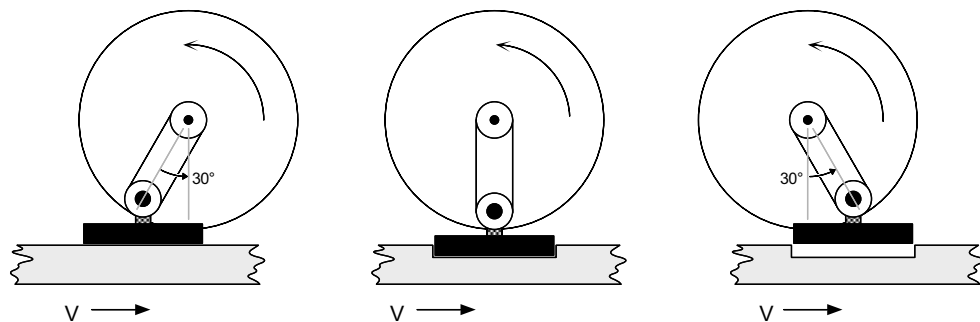
```

NOTE: The GOBUF command has been added to the “Define the exit motion” portion of the program despite the fact that an infinite loop has been programmed earlier in the program. This is to avoid an error message when the program is compiled.

Compiled Motion — Sample Application 4

A manufacturer of stamped molds needs to make a machine that will stamp molds into a continuous flow of extruded plastic material. The stamp must be lowered 0.5 inches into the plastic to leave the correct impression. Because the flow is continuous, the stamp must also move in synchronization with plastic in the direction of flow as it is lowered and raised. The initial design approach to the machine required two axes of motion. One was needed to lower and raise the stamp, the other to allow the stamp to follow the plastic. With the availability of complex Following cam profiles the job can be done with a single axis.

In the drawing below, the stamp is attached to a rotating arm in such a way that the stamp remains level as the arm rotates. The length of the arm at the stamp fixture, or radius of rotation, is exactly one inch. The arm is mounted above the plastic so that at the bottom of its rotation (270 degrees), the stamp will be 0.5 inches into the plastic. Using trigonometry, the horizontal and vertical positions and speeds can be calculated at other arm angles. Because the stamp must follow the flow of the plastic, we must adjust the ratio of rotational speed to plastic speed so that the horizontal velocity component of the arm stays at 1:1 with the plastic while the stamp is in the plastic.



The table below shows these relationships. The arm is directly driven with a servo motor having 4096 steps per revolution. The table shows increments of 30 degrees, which is about 341 servo motor steps, or about 0.524 slave inches measured around the circumference described by rotation of the arm. The plastic flow is measured with an encoder giving 1000 steps per inch of flow. To maintain ratios in terms of inches, FOLRD will always be 1000. The required FOLRN value is simply the inverse of the arm’s horizontal velocity component multiplied by the number of slave steps per inch. The corresponding ratio in terms of surface speeds is given in parentheses. The required FOLMD is the number of master steps corresponding to the horizontal component of slave rotation.

Arm angle, degrees	Horizontal component (in.) = cos(deg)	FOLMD = 1000 * delta cos(deg)	Horizontal vel component = -sin(deg)	Required FOLRN = -651.9/sin(deg)
210	-0.866	n/a	0.500	1304 (2:1)
240	-0.500	366	0.866	753 (1.155:1)
270	0.000	500	1.000	652 (1:1)
300	0.500	500	0.866	753 (1.155:1)
330	0.866	366	0.500	1304 (2:1)

The profile that we construct from these number is meant to approximate the inverse sine function in the last column, but of course, will actually be a series of ramps and constant ratio segments. Let's review the Compiled Following Move Distance Calculations to determine the exact shape and error in the first motion segment(from 210 to 240 degrees). First, we need to determine if the ramp or constant ratio is first for that segment. Using ratios and distances in inches, we have:

R1 = 2 Starting ratio
R2 = 1.155 Final ratio
D = (2*pi)/12 = 0.524 Distance at stamp hinge
FOLMD=.366 Travel along plastic

We find (R1+R2) * FOLMD/2 = 0.577, which is greater than D, so the "Ramp First" equations apply to this segment. Let's examine the error at the junction between the ramp and constant ratio portion of this segment.

MD1 = [D - (R2 * FOLMD)] / ((R1 - R2) / 2) = 0.239 master inches
D1 = 0.5 * (R1 + R2) * MD1 = 0.377 slave inches at circumference = 21.6 degrees
cos(210+21.6) - cos(210) = -0.621 - (-0.866) = 0.245 inches slave horizontal travel
error = horizontal slave travel - master travel = 0.245 - 0.239 = 0.006 inches

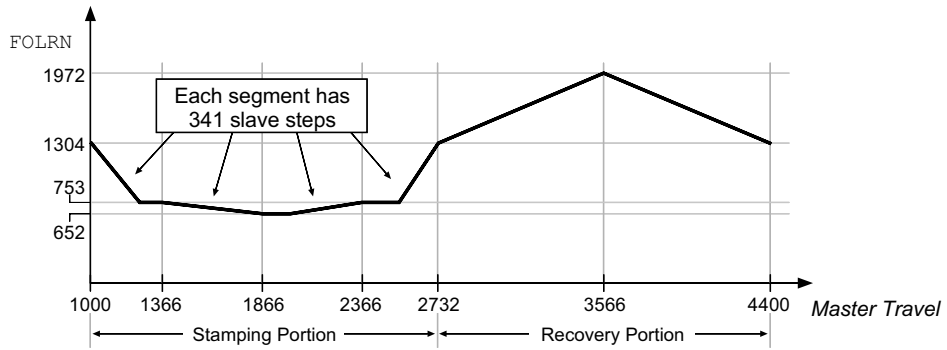
A similar calculation can be done for the "elbow" of the next of the next segment, and symmetry indicates these errors will be the same between 270 and 330 degrees. The error along intermediate points can be found with linear interpolation of ratio and master distance. In this case, the errors fall within manufacturing tolerance. If the errors were too large, the travel could be broken into more segments, each with exactly correct positions and ratios at their boundaries.

So far, we have only discussed the portion of the profile that lowers and raises the stamp. During the remainder of the profile, the arm must continue its rotation to bring the stamp to its starting position in time for the next mold. The mold is 3 inches long, and .4 inches are needed between molds for strength at the edges. This makes the total master cycle 3.4 inches long. The total slave cycle must be 4096 steps, so the segments required to bring the arm around must complete the portions of master and slave cycles not already accounted for. We will create two segments, which divide the remaining master and slave travels in two, and are mirror images of each other. The average ratio of these two segments must simply be slave travel divided by master travel, i.e., (D / FOLMD). As previously determined, the FOLRN value for the boundaries of the stamping portion of the profile is 638. From this value and the average ratio, we can calculate the peak FOLRN value.

D = 0.5 * remaining slave = 0.5 * (4096 - 4 * 341) = 1366
FOLMD = 0.5 * remaining master = 0.5 * [1000 * (3.4 - 2 * 0.866)] = 834
peak ratio = FOLRN/1000
0.5 * (FOLRN/1000 + 1304/1000) = average ratio = D / FOLMD = 1366 / 834 = 1.638
FOLRN = 1972 (solved from above)

Finally, we need to design a segment used to create a smooth entry into the repetitive portion of the profile. We'll assume that the home position of the arm is at 180 degrees, so it needs to achieve the FOLRN ratio of 1304 in 30 degrees (341 slave steps). Using the same averaging arithmetic as above, the required master distance for the entry segment is 523 steps. A sensor is positioned with this entry segment in mind, and wired to **TRG-1A**. A function to start motion when the sensor is triggered will be imbedded inside the profile. The motion segments for the stamping portion and recovery portions of the profile must be enclosed in a loop, and can be programmed by picking the numbers from the table and equations above. Because the ratio denominators are the same for all segments, and the slave distances are the same for the entry and each of the stamping segments, these are commanded only when the values change.

Repetitive Portion of Profile



```

Program  FOLMAS21      ; Follow extra encoder
        FOLEN1       ; Enable Following mode
        INFNC1-H     ; Enable trigger 1 (TRG-1A) for interrupt function
        SCALE0       ; Parameters are in steps

        DEF STAMP    ; Start program definition
        1TRGFNA1     ; Axis 1 profile starts upon trigger 1 (TRG-1A)
        FOLRD1000    ; Ratio denominator, 1000 steps per inch
        ;define the entry segment
        D341         ; Distance of 341 steps is about 30 degrees
        FOLRN1304    ; Goal ratio for start segment
        FOLMD523     ; Master distance during ramp
        GOBUF1       ; Build start segment
        PLOOP0       ; Start the continuous loop
        ;this profile section starts 2-to-1 ratio, or a starting FOLRN1304
        FOLRN753     ; Goal ratio for segment
        FOLMD366     ; Master travel in steps for segment
        GOBUF1       ; Build motion segment
        ;the 2nd section of profile starts with the final ratio of the 1st section
        FOLRN652     ; Goal ratio for segment
        FOLMD500     ; Master travel in steps for segment
        GOBUF1       ; Build motion segment
        ;the next two sections are mirror images of the first two
        FOLRN753     ; Goal ratio for third segment
        FOLMD500     ; Master travel in steps for 3rd segment
        GOBUF1       ; Build motion segment
        FOLRN1304    ; Goal ratio for 4th segment
        FOLMD366     ; Master travel in steps for 4th segment
        GOBUF1       ; Build motion segment
        ;the next two sections complete the loop and are mirror images of each other
        D1366        ; Slave travel in recovery segments
        FOLMD834     ; Master travel in steps for recovery segments
        FOLRN1972    ; Goal ratio for ramp up segment
        GOBUF1       ; Build ramp up motion segment
        FOLRN523     ; Goal ratio for ramp down segment
        GOBUF1       ; Build ramp down motion segment
        PLN1         ; End of loop cycle
        ;finally, a segment to end motion
        D341         ; Distance of 341 steps is about 30 degrees
        FOLRN0       ; Goal ratio for end segment
        FOLMD1000    ; Master distance during ramp
        GOBUF1       ; Build end segment
        END          ; End of STAMP program definition

        PCOMP STAMP  ; Compile the program

        ; *****
        ; * To execute the program, enter the PRUN STAMP command *
        ; *****

```

While motion is in progress, you can change these motion parameters to affect a new profile:

- Acceleration (A) — s-curve acceleration is not allowed
- Deceleration (AD) — s-curve deceleration is not allowed
- Velocity (V)
- Distance (D)
- Preset or Continuous Positioning Mode Selection (MC)
- Incremental or Absolute Positioning Mode Selection (MA)
- Following Ratio Numerator and Denominator (FOLRN and FOLRD, respectively)

The motion parameters can be changed by sending the respective command (e.g., A, V, D, MC) followed by the GO command. If the continuous command execution mode is enabled (COMEXC1), you can execute buffered commands; otherwise (COMEXC0), you must prefix each command with an immediate command identifier (e.g., !A, !V, !D, !MC, followed by !GO).

The new GO command pre-empts the motion profile in progress with a new profile based on the new motion parameter(s). On-the-fly motion changes are applicable only for motion started with the GO command, and not for motion started with other commands such as HOM, JOG, JOY, PRUN or GOL.

On-the-fly motion changes are most likely to be used to change the velocity and/or goal position of a preset move already underway. In the event that the goal position is completely unknown before motion starts, a move can be started in continuous mode (MC1), with a switch to preset mode (MC0), a distance command (D), and a GO given later. In absolute positioning mode (MA1) the new goal position given with a pre-emptive GO is explicit in the D command. In incremental positioning (MA0) the distance given with a new pre-emptive GO is always measured from the at-rest position before the original GO. If a move is stopped (with the S command), and then resumed (with the C command), this resumed motion is considered to be part of the original GO. A subsequent distance given with a new pre-emptive GO is measured from the at rest position before the original GO, not the intermediate stopped position.

<p>Programming Example:</p> <pre> SCALE0 ; Disable scaling DEL OTF ; Delete program (in case program is already in memory) DEF OTF ; Begin definition of program PSET0 ; Set position to zero COMEXC1 ; Enable continuous command processing mode MC0 ; Select preset positioning MA0 ; Select incremental positioning A20 ; Set accel to 20 revs/sec/sec AD20 ; Set decel to 20 revs/sec/sec V9 ; Set velocity to 9 revs/sec D500000 ; Set distance to 20 revs GO1 ; Initiate motion WAIT(PC>100000) ; Wait until commanded position > 100000 steps (4 revs) V4 ; Slow down for machine operation GO1 ; Initiate new profile with new velocity WAIT(PC>450000) ; Wait until the motor position > 450000 steps (18 revs) AD5 ; Set decel for gentle stop V1 ; Slow down for gentle stop GO1 ; Initiate new profile with new velocity END ; End program definition </pre>	<p><i>This program creates a 2-tiered profile (single-axis) that changes velocity and deceleration at specific motor positions.</i></p>
---	---

The table below summarizes the restrictions on pre-emptive GOs.

Condition	Possible?
Execute GO during MC1 & FOLENØ	Yes
Execute GO during MC1 & FOLEN1	Yes
Execute GO during MCØ & FOLENØ	Yes
Execute GO during MCØ & FOLEN1	No
Change MC setting during motion	Yes (but cannot change MC1 to MCØ during FOLEN1)
Change ENC setting during motion	No
Change FOLENØ to FOLEN1 during motion	No
Change FOLEN1 to FOLENØ during motion	Only while MC1, constant ratio, and not shifting

OTF Error Conditions

The ability to change the goal position on the fly raises the possibility of several error conditions:

Further instructions about handling error conditions are provided on page 30.

Error Conditions	6K Response		
	Set axis status bit 30	Set error status bit 10	Kill motion (decelerate at the LHAD value)
The new position goal of an on-the-fly GO cannot be reached with the current direction, velocity, and decel.	YES	YES	NO
The direction of the new goal position is opposite that of current travel	YES	YES	YES
There has not yet been an overshoot, but it is not possible to decelerate to the new distance from the current velocity using the specified AD value.	YES	YES	YES

RELATED STATUS COMMANDS

Axis Status — Bit 30: (this status bit is cleared with the next GO command)

AS.30Assignment & comparison operator — use in a conditional expression (see page 25).

TASFFull text description of each status bit. (see “Preset Move Overshot” line item)

TASBinary report of each status bit (bits 1-32 from left to right). See bit 30.

Error Status — Bit 10: The error status is monitored and reported only if you enable error-checking bit 10 with the ERROR command (e.g., ERROR.1Ø-1). **NOTE:** When the error occurs, the controller will branch to the error program (assigned with the ERRORP command). (this status bit is cleared with the next GO command)

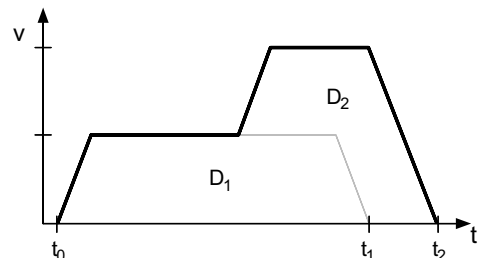
ER.10Assignment & comparison operator — use in a conditional expression (see page 25).

TERFFull text description of each status bit. (see “Preset Move Overshot” line item)

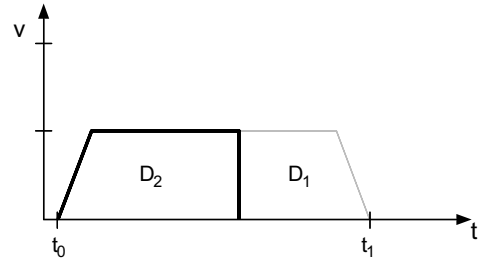
TERBinary report of each status bit (bits 1-32 from left to right). See bit 10.

Scenarios

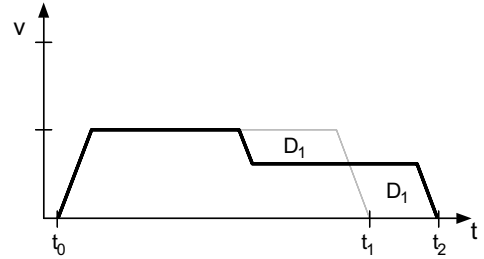
Scenario 1: OTF change of velocity and distance, where new commanded distance (D_2) is greater than the original distance (D_1) that was pre-empted [$D_2 > D_1$]. The distances are the areas under the profiles, starting at t_0 for both. If the original move had continued, D_1 would have been reached at time t_1 . D_2 is reached at time t_2 .



Scenario 2: OTF change of distance, where new commanded distance (D_2) is less than the original distance (D_1) that was pre-empted [$D_2 < D_1$]. In this example, the position where the OTF change was entered is already beyond D_2 (or D_2 cannot be reached with the commanded deceleration). The result is an error and motion is killed (decel at the LHAD value) and TAS bit 30 and TER bit 10 are set.



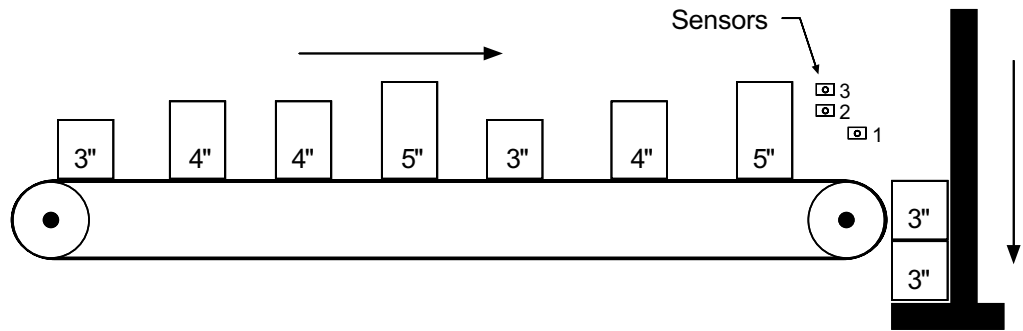
Scenario 3: OTF change of velocity. Note that motion must continue for a longer time at the reduced velocity to reach the original commanded distance than if it had continued at the original velocity ($t_2 > t_1$).



On-The-Fly Motion — Sample Application

A manufacturer of three products wishes to produce a “sampler-pak” package that will contain a few of each of his products. The products all have the same width and length, but are 3, 4, and 5 inches high respectively. The 3 products are fed from individual lines into a common conveyor, and arrive at a stacking and wrapping station. At this station, a tray accepts a product and must have moved down by that product’s height by the time the next product arrives. This means that each time a new product arrives, the velocity of the tray must be changed to match the height of that product. Although product spacing will be regular, the ordering of product type on the common conveyor will be random, due to variations in the input lines. Also, a finished sampler-pak should contain 5 products or be at least 18 inches high, whichever occurs first. This means that the total move distance of the tray will be unknown until the last product arrives. When the last product is stacked, an output is asserted that will pause the conveyor and start the wrapping process. When wrapping is complete, the sampler-pak is removed from the tray, and the tray returns to the starting position.

The basic problems in this application are that the move distance is not known until near the end, and the velocity must change on the fly. As the products approach the tray, they are detected with a near vertical arrangement of three sensors. Products of heights 3, 4, and 5 inches are detected by 1, 2, or all 3 sensors respectively. Input 1 always detects a product, and switches last, so that the others will be stable. When each product is identified, the motion profile is modified accordingly.



**Program
(portion only)**

```

VAR1=0           ; Initialize product count
VAR2=0           ; Initialize move distance variable
VAR3=0           ; Initialize velocity
A10              ; Moderate acceleration
MC0              ; Start with preset move
WHILE (VAR1<5 AND VAR2<18) ; Loop until cycle complete
WAIT (IN.1=B1)   ; Wait for start of next product
                 ; (onboard input 1 is activated)
VAR1=VAR1+1     ; Update product count
IF (IN.2=B1)    ; If not a 3" product
  IF (IN.3=B1)  ; If it is a 5" product
    VAR3=5     ; Set velocity
    VAR2=VAR2+5 ; Update distance
  ELSE        ; If not 5", must be 4"
    VAR3=4     ; Set velocity
    VAR2=VAR2+4 ; Update distance
  NIF        ; End of 5" case check
ELSE        ; 3" inch case
  VAR3=3     ; Set velocity
  VAR2=VAR2+3 ; Update distance
NIF        ; End of 3" case check
V(VAR3)    ; New velocity
D(VAR2)    ; New distance
WAIT (IN.1=B0) ; Wait for end of this product
GO1        ; Implement new distance and velocity
NWHILE    ; Sampler-pak completed product detection
WAIT (AS.1=B0) ; Wait for move to complete
OUT1      ; Output to indicate stacking complete

```

Registration

A “registration input” is a trigger input assigned the “trigger interrupt” function with the `INFNCi-H` command.

When a *registration input* is activated, the motion profile currently being executed is replaced by the registration profile with its own distance (`REG`), acceleration (`A` & `AA`), deceleration (`AD` & `ADA`), and velocity (`V`) values. The registration move can interrupt any preset, continuous, or registration move in progress.

The registration move does not alter the rest of the program being executed when registration occurs, nor does it affect commands being executed in the background if the controller is operating in the continuous command execution mode (`COMEXC1`).

Registration moves will not be executed while the motor is not performing a move, while in the joystick mode (`JOY1`), or while decelerating due to a stop, kill, soft limit, or hard limit.

How to Set up a Registration Move

Before you can initiate a registration move, you must program these elements (refer also to the programming examples below):

1. Configure one of the trigger inputs (`TRG-nA` or `TRG-nB` per axis) to function as a trigger interrupt input; this is done with the `INFNCi-H` command, where “i” is the input bit number representing the targeted trigger input. Note that the “Master Trigger” input cannot be used for registration.
2. Specify the distance of the registration move with the `REG` command. For servo axes, the distance refers to the encoder position (registration cannot be used with analog input feedback). For stepper axes, the distance refers to commanded position if `ENCCNT0` (default setting) or encoder position if `ENCCNT1`.
3. Enable the registration function with the `RE` command. Registration is performed only on the axis or axes with the registration function enabled, and with a non-zero distance specified in the respective axis-designation field of the `REG` command; the other axes will not be affected. Each trigger has a distinct move defined for its axis; for example, trigger 3 (called `TRG-2A`) can initiate a registration move for axis 2 with the specified `2REGA` distance.

NOTE: The registration move is executed using the `A`, `AA`, `AD`, `ADA`, and `V` values that were in effect when the `REG` command was entered.

⇒
ENCCNT: Encoder capture options for stepper axis are discussed on page 66.

Registration Move Accuracy (see also *Registration Move Status* below)

The accuracy of the registration move distance specified with the `REG` command is ± 1 count (servo axes: encoder count; stepper axes: commanded count if `ENCCNT0` or encoder count if `ENCCNT1`).

RULE OF THUMB: To prevent position overshoot, make sure the `REG` distance is greater than 4 ms multiplied by the incoming velocity.

The lapse between activating the registration input and commencing the registration move (this does not affect the move accuracy) is less than one position sample period (2 ms).

The `REG` distance will be scaled by the distance scale factor (`SCLD` value) if scaling is enabled (`SCALE1`). See page 51 for details on scaling.

Preventing Unwanted Registration Moves (*methods*)

- **Registration Input Debounce:** Registration Input Debounce: By default, the registration inputs are debounced for 2 ms before another input on the same trigger is recognized. (The debounce time is the time required between a trigger's initial active transition and its secondary active transition.) Therefore, the maximum rate that a registration input can initiate registration moves is 500 times per second. If your application requires a shorter debounce time, you can change it with the TRGLOT command.
- **Registration Single-Shot:** The REGSS command allows you to program the 6K controller to ignore any registration commands after the first registration move has been initiated. Refer to the REGSS command description for further details and an application example.
- **Registration Lockout Distance:** The REGLD command specifies what distance an axis must travel before any trigger assigned as a registration input will be recognized. Refer to the *Sample Application 3* below.

Registration Move Status & Error Handling

Axis Status — Bit 28: This status bit is set when a registration move has been initiated by any registration input (trigger). This status bit is cleared with the next GO command.

AS . 28 Assignment & comparison operator — use in a conditional expression (see page 25).
TASF..... Full text description of each status bit. (see “Reg Move Commanded” line item)
TAS..... Binary report of each status bit (bits 1-32 from left to right). [See bit 28.](#)

Axis Status — Bit 30: If, when the registration input is activated, the registration move profile cannot be performed with the specified motion parameters, the 6K controller will kill the move in progress and set axis status bit 30. This status bit is cleared with the next GO command.

AS . 30 Assignment & comparison operator — use in a conditional expression (see page 25).
TASF..... Full text description of each status bit. (see “Preset Move Overshot” line item)
TAS..... Binary report of each status bit (bits 1-32 from left to right). [See bit 30.](#)

Error Status — Bit 10: This status bit can be set if axis status bit 30 is set. The error status is monitored and reported only if you enable error-checking bit 10 with the ERROR command (e.g., ERROR . 10-1). **NOTE:** When the error occurs, the controller will branch to the error program (assigned with the ERRORP command). This status bit is cleared with the next GO command.

ER . 10 Assignment & comparison operator — use in a conditional expression (see page 25).
TERF..... Full text description of each status bit. (see “Preset Move Overshot” line item)
TER..... Binary report of each status bit (bits 1-32 from left to right). [See bit 10.](#)

Trigger Status — Bits 1-17: Trigger status bits are set when a registration move has been initiated by trigger inputs A or B for each axis, or with the TRIG-M (master trigger) input. This also indicates that the positions of all axes has been captured. As soon as the captured information is transferred or assigned/compared (see page 84), the respective trigger status bit is cleared (set to ∅).

TRIG..... Assignment & comparison operator — use in a conditional expression.(see page 25).
TTRIG..... Binary report of each status bit (bits 1-17 from left to right). From left to right the bits represent trigger A and B for axes 1-8, the 17th bit is master trigger M (the “MASTER TRIG” input terminal) — see page 76.

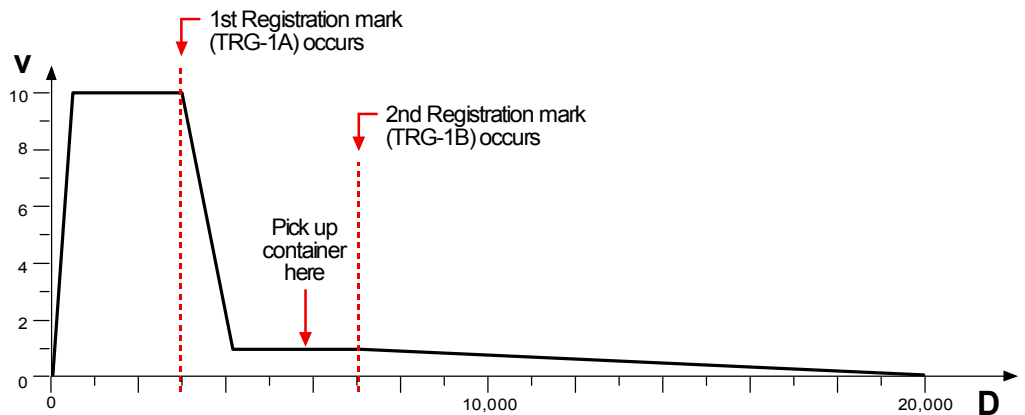
Further instructions about handling error conditions are provided on page 30.

Registration — Sample Application 1

In this example, two-tiered registration is achieved (see illustration below). While axes 1 is executing it's 50,000-unit move, trigger input 1 (TRG-1A) is activated and executes registration move A to slow the load's movement. An open container of volatile liquid is then placed on the conveyor belts. After picking up the liquid and while registration move A is still in progress, trigger input 2 (TRG-1B) is activated and executes registration move B to slow the load to gentle stop.

```

DEL REGI1      ; Delete program (in case program already resides in memory)
DEF REGI1      ; Begin program definition
INFNC1-H      ; Define trigger input 1 (TRG-1A) as trigger interrupt input
INFNC2-H      ; Define trigger input 2 (TRG-1B) as trigger interrupt input
A20           ; Set acceleration on axis 1 to 20 units/sec2
AD40          ; Set deceleration on axis 1 to 40 units/sec2
V1           ; Set velocity on axis 1 to 1 unit/sec
1REGA4000     ; Set TRG-1A's registration distance on axis 1 to 4000 units
              ; (registration A move will use the A, AD, & V values above)
A5           ; Set acceleration on axis 1 to 5 units/sec/sec
AD2          ; Set deceleration on axis 1 to 2 units/sec/sec
V.5         ; Set velocity on axis 1 to 0.5 units/sec
1REGB13000   ; Set TRG-1B's registration distance on axis 1 to 13,000 units
              ; (registration B move will use the A, AD, & V values above)
RE10        ; Enable registration on axis 1 only
A50         ; Set acceleration to 50 units/sec/sec on axis 1
AD50        ; Set deceleration to 50 units/sec/sec on axis 1
V10        ; Set velocity to 10 unit/sec on axis 1
D50000     ; Set distance to 50000 units on axis 1
GO10       ; Initiate motion on axis 1
END         ; End program definition
    
```

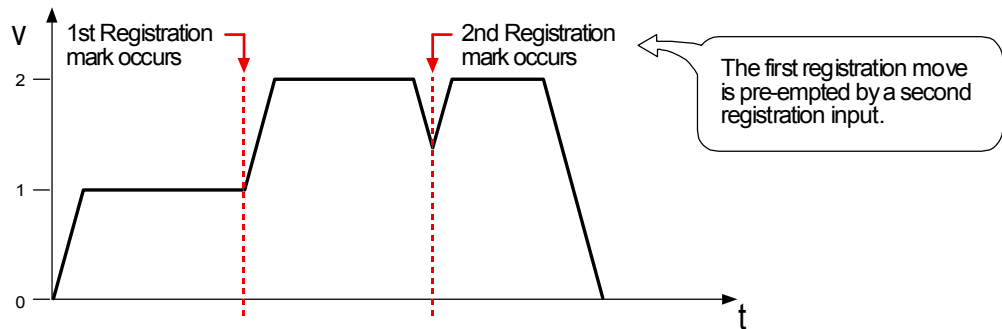


Registration — Sample Application 2

A user has a line of material with randomly spaced registration marks. It is known that the first mark must initiate a registration move, and that each registration move cannot be interrupted or the end product will be destroyed. Since the distance between marks is random, it is impossible to predict if a second registration mark will occur before the first registration move has finished.

```

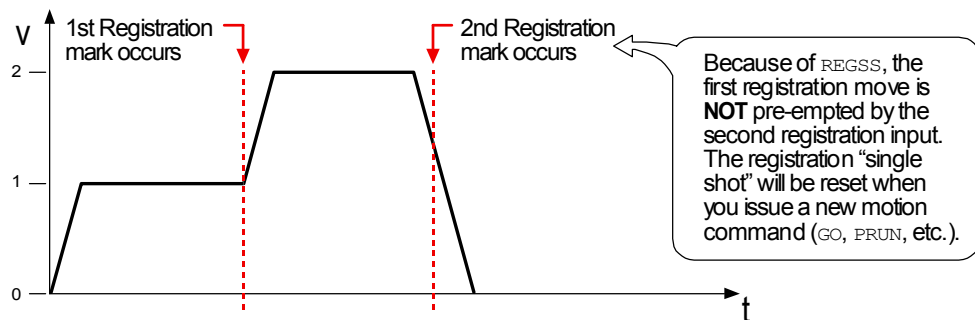
DEL REGI2      ; Delete program (in case program already resides in memory)
DEF REGI2      ; Begin program definition
INFNC1-H      ; Trigger capture mode for trigger 1 (TRG-1A)
RE1           ; Enable registration
V2           ; Set registration move to a velocity of 2 rps
AD.5         ; a deceleration of 0.5 rev/sec/sec
1REGA20000   ; and a distance of 20000 steps for axis 1
MC1          ; Start a mode continuous
V1           ; move at a velocity of 1 rps
GO1          ; Initiate motion
END           ; End program definition
    
```



In order to stop the second registration from occurring, REGSS can be used:

```

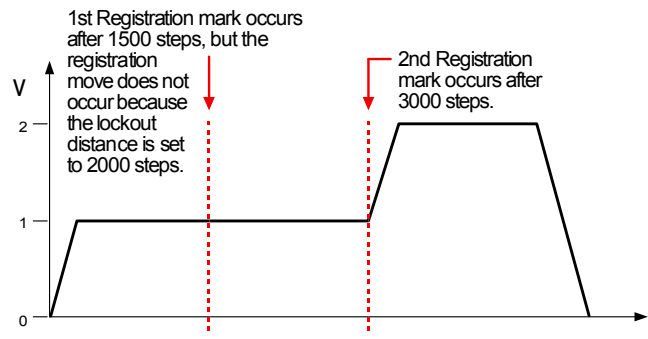
DEL REGI2b    ; Delete program (in case program already resides in memory)
DEF REGI2b    ; Begin program definition
INFNC1-H      ; Trigger capture mode for trigger 1 (TRG-1A)
RE1           ; Enable registration
V2           ; Set registration move to a velocity of 2 rps
AD.5         ; and a distance of 20000 steps for axis 1
REGSS1       ; Enable registration single shot mode
MC1          ; Start a mode continuous
V1           ; move at a velocity of 1 rps
GO1          ; Initiate motion
END           ; End program definition
    
```



Registration — Sample Application 3

A print wheel uses registration to initiate each print cycle. From the beginning of motion, the controller should ignore all registration marks before traveling 2000 steps. This is to ensure that the unit is up to speed and that the registration mark is a valid one.

```
DEL REGI3      ; Delete program (in case program already resides in memory)
DEF REGI3      ; Begin program definition
INFNC1-H      ; Trigger capture mode for trigger 1 (TRG-1A)
RE1           ; Enable registration
V2            ; Set registration move to a velocity of 2 rps
1REGA2500     ; and a distance of 2500 steps for axis 1
REGLOD2000    ; Set registration lockout distance to 2000 steps
MC1          ; Start a mode continuous
V1           ; move at a velocity of 1 rps
GO1          ; Initiate motion
END           ; End program definition
```



Synchronizing Motion (GOWHEN and TRGFN operations)

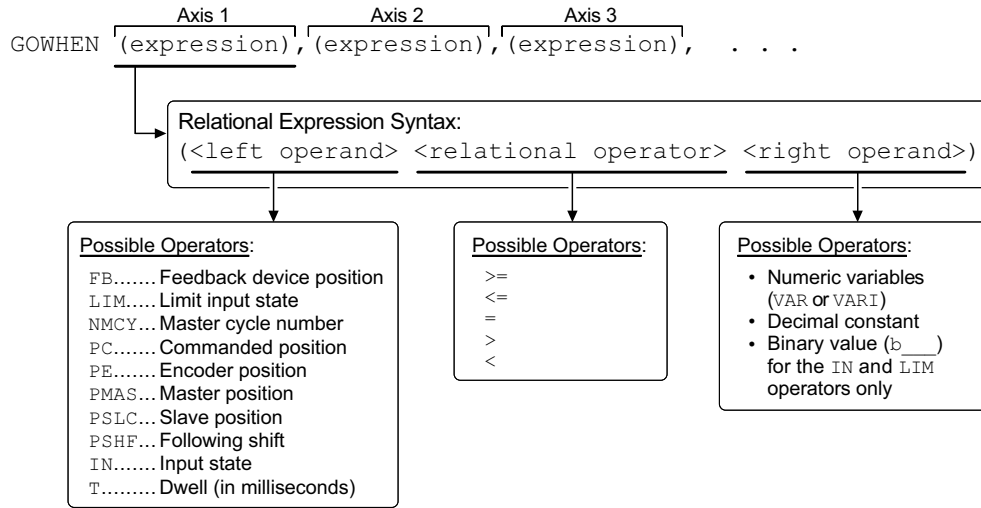
GOWHEN and TRGFN allow you to synchronize the execution of motion on one or more axes:

- GOWHEN — synchronize execution of the subsequent start-motion command (GO, GOL, FGADV, FSHFC, or FSHFD) to:
 - Position (commanded, feedback device, motor, master, slave, Following shift)
 - Master cycle number
 - Input status
 - Time delay (dwell)
- TRGFN:
 - Suspend execution of the next start-motion command (GO, GOL, FGADV, FSHFC, or FSHFD) until the specified trigger input goes active.
 - Suspend beginning a new Following master cycle until the specified trigger input goes active.

Conditional “GO”s (GOWHEN) The GOWHEN command is used to synchronize a motion profile of an axis with a specified position count, input status, dwell (time delay), or master cycle number on that axis or other axes. Command processing does not wait for the GOWHEN conditions (relational expressions) to become true during the GOWHEN command. Rather, the motion from the subsequent start-motion command (GO, GOL, FGADV, FSHFC, and FSHFD) will be suspended until the condition becomes true.

Start-motion type commands that **cannot** be synchronized using the GOWHEN command are: HOM, JOG, JOY, and PRUN. A preset GO command that is already in motion can start a new profile using the GOWHEN and GO sequence of commands. Continuous moves (MC1) already in progress can change to a new velocity based upon the GOWHEN and GO sequence. Both preset and continuous moves can be started from rest with the GOWHEN and GO sequence.

GOWHEN
Syntax



EXAMPLES

```
GOWHEN (1PE>40000) ; suspend next GO until axis 1 encoder position > 40000
GOWHEN (IN.6=b1) ; suspend next GO until onboard input #6 is activated (b1)
GOWHEN (2PMAS>255) ; suspend next GO until the master for axis 2 has
; traveled 255 master distance units
```

SCALING

If scaling is enabled (SCALE1), the right-hand operand is multiplied by SCLD if the left-hand operand is FB, PC, PE, PSLV, or PSHF. The right-hand operand is multiplied by the SCLMAS value if the left-hand operand is PMAS. (The SCLD or SCLMAS values used correlate to the axis specified with the variable—e.g., a GOWHEN expression with 3PE scales the encoder position by the SCLD value specified for axis 3.)

GOWHEN
Status

Axis Status — Bit 26: Bit 26 is set when motion has been commanded by a GO, GOL, FGADV, FSHFC, or FSHFD command, but is suspended due to a pending GOWHEN condition. This status bit is cleared when the GOWHEN condition is true or when a stop (!S) or kill (!K or ^K) command is executed. An individual axis' GOWHEN command can be cleared using an axis-specific S or K command (e.g., !S11X0 or !K0XX1).

- AS.26.....Assignment & comparison operator — use in a conditional expression (see page 25).
- TASF.....Full text description of each status bit. (see “Gowhen is Pending” line item)
- TAS.....Binary report of each status bit (bits 1-32 from left to right). See bit 26.

Axis Status — Bit 29: Bit 29 is set when the input state of position relationship specified in the GOWHEN expression was already true when the subsequent GO, GOL, FGADV, FSHFC or FSHFD command was executed.

- AS.29.....Assignment & comparison operator — use in a conditional expression (see page 25).
- TASF.....Full text description of each status bit. (see “Gowhen Error” line item)
- TAS.....Binary report of each status bit (bits 1-32 from left to right). See bit 29.

Further instructions about handling error conditions are provided on page 30.

Error Status — Bit 14: Bit 14 is set if the input state or position relationship specified in the GOWHEN expression was already true when the subsequent GO, GOL, FGADV, FSHFC, or FSHFD command is issued. The error status is monitored and reported only if you enable error-checking bit 14 with the ERROR command (e.g., ERROR.14-1). **NOTE:** When the error occurs, the controller will branch to the error program (assigned with the ERRORP command).

- ER.14Assignment & comparison operator — use in a conditional expression (see page 25).
- TERF.....Full text description of each status bit. (see “GOWHEN condition true” line item)
- TER.....Binary report of each status bit (bits 1-32 from left to right). See bit 14.

GOWHEN ...
On a Trigger Input

If you wish motion to be triggered with a trigger input, use the aTRGFNC1 command. The aTRGFNC1 command executes in the same manner as the GOWHEN command, except that motion is executed on specified axis “a” when the specified trigger input “c” is activated. For more information, see *Trigger Functions* below.

GOWHEN VS.WAIT

A WAIT will cause the 6K controller program to halt program flow (except for execution of immediate commands) until the condition specified is satisfied. Common uses for this function include delaying subsequent I/O activation until the master has achieved a required position or an object has been sensed.

By contrast, a GOWHEN will suspend the motion profile for a specific axis until the specified condition is met. It does **not** affect program flow. If you wish motion to be triggered with a trigger input, use the TRGFNC1 command. The TRGFNC1 command executes in the same manner as the GOWHEN command, except that motion is executed when the specified trigger input (c) is activated (see *Trigger Functions* below for details). In addition, GOWHEN expressions are limited to the operands listed above; WAIT can use additional operands such as FS (Following status) and VMAS (velocity of master).

Factors Affecting
GOWHEN Execution

If, on the same axis, a second GOWHEN command is executed **before** a start-motion command (GO, GOL, FSHFC, or FSHFD), then the first GOWHEN is over-written by the second GOWHEN command. (GOWHEN commands are not nested.) An error is not generated when a GOWHEN command is over-written by another GOWHEN.

While waiting for a GOWHEN condition to be met **and** a start-motion command **has** been issued, if a second GOWHEN command is encountered, then the first sequence is disabled and another start-motion command is needed to re-arm the second GOWHEN sequence.

A new GOWHEN command must be issued for each start-motion command (GO, GOL, FGADV, FSHFC, or FSHFD). That is, once a GOWHEN condition is met and the motion command is executed, subsequent motion commands will not be affected by the same GOWHEN command.

If the GOWHEN and start-motion commands are issued, the motion profile is delayed until the GOWHEN condition is met. If a second start-motion command is encountered, the second start-motion command will override the GOWHEN command and start motion. If this override situation is not desired, it can be avoided by using a WAIT condition between the first start-motion command and the second start-motion command.

It is probable that the GOWHEN command, the GO command, and the GOWHEN condition becoming true can be separated in time, and by other commands. Situations can arise, or commands can be given that make the GOWHEN invalid or inappropriate. In these cases, the GOWHEN condition is cleared, and any motion pending the GOWHEN condition becoming true is canceled. These situations include execution of the JOG, JOY, HOM, PRUN, and DRIVEØ commands, as well motion being stopped due to hard or soft limits, a drive fault, an immediate stop (!S), or an immediate kill (!K or ^K).

⇒
GOWHEN in Compiled
 Motion (Compiled
 Motion is discussed on
 page 136)

When used in a compiled program, a *GOWHEN* will pause the profile in progress (motion continues at constant velocity) until the *GOWHEN* condition evaluates true. When executing a compiled Following profile, the *GOWHEN* is ignored on the reverse Following path (i.e., when the master is moving in the opposite direction of that which is specified in the *FOLMAS* command). A compiled *GOWHEN* can require up to 4 segments of compiled memory storage.

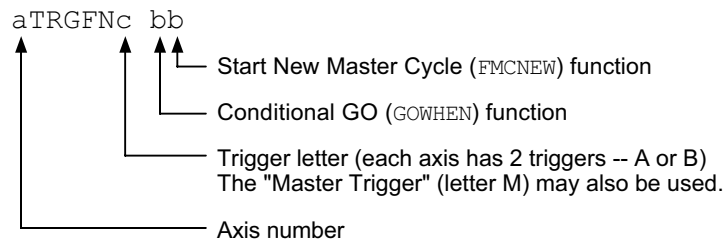
Sample 6K Code

In the example below, axis 2 must start motion when the actual position of axis 1 has reached 4. While axis 1 is moving, the program must be monitoring inputs and serving other system requirements, so a *WAIT* statement cannot be used; instead, a *GOWHEN* and *GO* sequence will delay the profile of axis 2.

```
MC00           ; Set both axes to preset move mode
D20,20        ; Set distance end-point
COMEXC1       ; Enable continuous command execution mode
V1,1          ; Set velocity
A100,100      ; Set acceleration
GOWHEN(,1PE>4) ; Delay axis 2 profile. When expression is true (position
               ; of encoder 1 is > 4), allow axis 2 to start motion.
GO11          ; Command both axes to move. Axis 2 will not start until
               ; conditions in the GOWHEN statement are true.
               ; Command processing does not wait, so other system
               ; functions can be performed.
```

Trigger Functions (TRGFN)

The Trigger Functions command (*TRGFN*) allows you to assign additional functions to trigger inputs that have been defined as “trigger interrupt” inputs (*INFNCi-H*):



- “**Conditional GO**” Function (*aTRGFNc1x*): Suspend execution of the next start-motion command on axis “a” until specified trigger input “c” goes active. Start-motion commands are listed below. Refer to page 159 or to the *GOWHEN* command description for additional details.
 - *GO* (standard command to begin motion)
 - *GOL* (begin linear interpolated motion)
 - *FGADV* (begin geared advance – for Following motion)
 - *FSHFC* (begin continuous shift – for Following motion)
 - *FSHFD* (begin preset shift – for Following motion)

Axis status bit 26 (reported with *TASF*, *TAS*, or *AS*) is set to one (1) when there is a pending “Conditional GO” condition initiated by a *TRGFN* command; this bit is cleared when the trigger is activated or when a stop command (*S*) or a kill command (*K*) is issued. If you need execution to be triggered by other factors (e.g., input state, master position, encoder position, etc.) use the *GOWHEN* command.

- “**New Master Cycle**” Function (`aTRGFNCx1`): This is equivalent to executing the `FMCNEW` command. When specified trigger input “c” goes active, the controller begins a new Following master cycle on axis “a”. For more information on master cycles, see page 183 or to the `FMCNEW` command description.

These trigger functions are cleared once the function is complete (trigger input is activated). To use the trigger to perform a `GOWHEN` function again, the `TRGFN` command must be given again.

```

Sample  INFNC2-H      ; Define trigger 2 (TRG-1B) as a trigger interrupt input
6K Code INFNC4-H      ; Define trigger 4 (TRG-2B) as a trigger interrupt input
1TRGFNBx1 ; When TRG-1B goes active, axis 1 will begin a
          ; new master cycle
2TRGFNB1x ; When TRG-2B goes active, axis 2 will execute the move
          ; commanded with the GO command.
GO01      ; The move on axis 2 is commanded, but will not execute
          ; until trigger 2B goes active.

```