Manual del operador

Introducción a RAPID

IRC5
RobotWare 5.0

Manual del operador
Introducción a RAPID

RobotWare 5.0

ID de documento: 3HAC029364-005
Revisión: -

La información de este manual puede cambiar sin previo aviso y no puede entenderse como un compromiso por parte de ABB. ABB no se hace responsable de ningún error que pueda aparecer en este manual.

Excepto en los casos en que se indica expresamente en este manual, ninguna parte del mismo debe entenderse como garantía alguna por parte de ABB por las pérdidas, lesiones, daños materiales, idoneidad para un fin determinado ni garantías similares.

ABB no será en ningún caso responsable de los daños accidentales o consecuentes que se produzcan como consecuencia del uso de este manual o de los productos descritos en el mismo.

Se prohíbe la reproducción o la copia de este manual o cualquiera de sus partes si no se cuenta con una autorización escrita de ABB. Ninguna parte de este manual debe ser entregada a terceros ni utilizada para fines no autorizados. Cualquier incumplimiento de esta norma será perseguido legalmente.

Usted puede obtener copias adicionales de este manual a través de ABB, con el coste aplicable en el momento de su solicitud.

ABB AB
Robotics Products
SE-721 68 Västerås
Suecia

Descripción general

Acerca de este manual

Este manual ha sido concebido como una primera introducción a RAPID. Se ha omitido una buena parte de la funcionalidad de RAPID, pero se describen las partes esenciales, de forma que sea fácilmente comprensible para todos los usuarios. Este manual no le permitirá alcanzar los conocimientos de RAPID de un programador experto, pero puede ayudarle a comprender el concepto de la programación con RAPID. Encontrará los detalles en todo momento en los manuales de referencia.

Utilización

Este manual debe leerse antes de empezar a programar. No contiene todo lo que puede necesitar saber, pero debe familiarizarse con la mayoría de los aspectos de este manual antes de empezar a escribir un programa de RAPID.

Este manual no sustituye a los cursos de formación acerca de RAPID, pero puede complementarlos.

¿A quién va destinado este manual?

Este manual está destinado a personas son experiencia previa en programación, por ejemplo un operador de robots que desee aprender a programar el robot.

Requisitos previos

No existe ningún requisito previo asociado a este manual.

Organización de los capítulos

Este manual está organizado en los capítulos siguientes:

Capítulo	Contenido
1. Conceptos básicos de RAPID	Los fundamentos de la programación. Esta funcionalidad es similar en la mayoría de los lenguajes de programación de alto nivel.
2. Funcionalidad de robots en RAPID	Describe la funcionalidad que hace de RAPID un lenguaje único, es decir las instrucciones de movimiento, las señales de E/S y la comunicación con un FlexPendant.
3. Estructura	Describe cómo crear procedimientos. También contiene una introducción breve sobre cómo aplicar el diseño estructurado de un programa.
4. Datos con varios valores	Describe las matrices y los tipos de datos complejos.
5. Instrucciones y funciones de RAPID	Una explicación breve de qué son las instrucciones y las funciones de RAPID.
6. Qué debe leer a continuación	Dónde encontrar más información si desea proseguir con sus estudios de RAPID.

Referencias

Referencia	ID de documento
Manual de referencia técnica - Descripción general de RAPID	3HAC16580-5

Continúa en la página siguiente

Continuación

Referencia	ID de documento
Manual de referencia técnica - RAPID Instrucciones, funciones y tipos de datos	3HAC16581-5
Technical reference manual - RAPID Kernel	3HAC16585-1
Manual del operador - IRC5 con FlexPendant	3HAC16590-5

Revisiones

Revisión	Descripción
-	Primera edición

Documentación del producto, M2004

General

La documentación del robot se divide en varias categorías diferentes. Esta lista se basa en el tipo de información contenida en los documentos, independientemente de si los productos son estándar u opcionales. Esto significa que un suministro determinado de productos de robot *no contendrá todos* los documentos enumerados, sino sólo aquellos que se corresponden con el equipo suministrado.

Sin embargo, puede pedir a ABB cualquiera de los documentos de la lista. Los documentos enumerados son válidos para los sistemas de robot M2004.

Manuales de producto

Todo el hardware, robots y controladores se suministran con un **Manual del producto**, que contiene:

- Información de seguridad
- Instalación y puesta en servicio (descripciones de la instalación mecánica y las conexiones eléctricas)
- Mantenimiento (descripciones de todos los procedimientos de mantenimiento preventivo necesarios, incluidos sus intervalos)
- Reparaciones (descripciones de todos los procedimientos de reparación recomendados, incluidos los repuestos)
- Procedimientos adicionales, si los hay (calibración, retirada del servicio)
- Información de referencia (referencias de la documentación a la que se hace referencia en el manual del producto, procedimientos, listas de herramientas, estándares de seguridad)
- Lista de piezas
- Láminas o vistas ampliadas
- Diagramas de circuitos

Manuales de referencia técnica

Los manuales siguientes describen el software del robot en general y contienen la información de referencia pertinente.

- **Descripción general de RAPID**: Una descripción general del lenguaje de programación RAPID.
- **Instrucciones, funciones y tipos de datos de RAPID**: Descripción y sintaxis de todos los tipos de datos, instrucciones y funciones de RAPID.
- **Parámetros del sistema**: Una descripción de los parámetros del sistema y los flujos de trabajo de configuración.

Continúa en la página siguiente

Continuación

Manuales de aplicaciones

Las aplicaciones específicas (por ejemplo opciones de software o hardware) se describen en **Manuales de aplicaciones**. Un mismo manual de aplicaciones puede describir una o varias aplicaciones.

Generalmente, un manual de aplicaciones contiene información acerca de:

- Finalidad de la aplicación (para qué sirve y en qué situaciones resulta útil)
- Contenido (por ejemplo cables, tarjetas de E/S, instrucciones de RAPID, parámetros del sistema, CD con software para PC)
- Forma de uso de la aplicación
- Ejemplos sobre cómo usar la aplicación

Manuales del operador

Este grupo de manuales está orientado a las personas que van a tener contacto de uso directo con el robot, es decir, operadores de células de producción, programadores y técnicos de resolución de problemas. El grupo de manuales se compone de:

- **Información de seguridad para emergencias**
- **Procedimientos iniciales - IRC5 y RobotStudioOnline**
- **IRC5 con FlexPendant**
- **RobotStudio Online**
- **Resolución de problemas - IRC5** para el controlador y el robot

Seguridad

Seguridad del personal

Los robots son pesados y tienen una fuerza extraordinaria independientemente de su velocidad. Una pausa o una parada larga en un movimiento puede ir seguida de un movimiento rápido y peligroso. Incluso si es posible predecir un patrón de movimientos, una señal externa puede disparar un cambio de funcionamiento y dar lugar a un movimiento inesperado.

Por tanto, es importante respetar toda la normativa de seguridad al entrar en un espacio protegido.

Normativa de seguridad

Antes de empezar a trabajar con el robot, asegúrese de familiarizarse con la normativa de seguridad descrita en el *Manual del operador - IRC5 con FlexPendant*.

Terminología

Acerca de los términos

Este manual se ha escrito generalmente para principiantes, tanto en programación como en robots. Sin embargo, algunos términos utilizados pueden resultar familiares sólo a personas con ciertos conocimientos de programación y/o robots industriales. Estos términos se describen en esta terminología.

Términos

Término	Descripción
FlexPendant	Un terminal de mano para controlar un sistema de robot.
Controlador de robot	El controlador de robot es básicamente un ordenador que controla el robot.
Sintaxis	Las reglas que determinan cómo se permite escribir un lenguaje. Puede contemplarse como la gramática del lenguaje de programación. La sintaxis de un lenguaje de programación es mucho más estricto que el de un idioma natural humano. Los humanos son inteligentes y comprenderían algo como "Yo rápido corro" en lugar de "Yo corro rápido". Los ordenadores, por otro lado, son más tontos y no comprenderían algo a no ser que su sintaxis sea totalmente correcta.

1 Conceptos básicos de RAPID

1.1. Acerca de RAPID

Qué es RAPID

Si quiere que un ordenador haga algo, necesita un programa. RAPID es un lenguaje de programación que permite escribir tal programa.

El lenguaje nativo de los ordenadores se compone exclusivamente de ceros y unos. Se trata de algo prácticamente imposible de comprender para las personas. Por tanto, se enseña a los ordenadores a comprender un lenguaje relativamente fácil de comprender: un lenguaje de programación de alto nivel. RAPID es un lenguaje de programación de alto nivel. Utiliza algunas palabras en inglés (como IF y FOR) para hacerlo comprensible para las personas.

Ejemplo de programa simple de RAPID

Veamos un ejemplo simple para ver qué aspecto puede tener un programa de RAPID:

```
MODULE MainModule
  VAR num length;
  VAR num width;
  VAR num area;

  PROC main()
     length := 10;
     width := 5;
     area := length * width;
     TPWrite "The area of the rectangle is " \Num:=area;
  END PROC
ENDMODULE
```

Este programa calculará el área de un rectángulo y la escribirá en el FlexPendant:

```
The area of the rectangle is 50
```


1.2 Datos de RAPID

1.2.1. Variables

Tipos de datos

RAPID cuenta con muchos tipos de datos diferentes. De momento, nos centraremos en los tres tipos de datos generales:

Tipo de dato	Descripción
num	Datos numéricos, que pueden ser enteros y con decimales, por ejemplo 10 ó 3,14159.
string	Una cadena de texto. Por ejemplo "Esto es una cadena". Máximo 80 caracteres.
bool	Una variable booleana (lógica). Sólo puede tener los valores `TRUE` o `FALSE`.

Todos los demás tipos de datos se basan en estos tres. Comprendiendo estos tipos, la forma de realizar operaciones con ellos y cómo pueden ser combinados para formar tipos de datos complejos, podrá comprender fácilmente todos los tipos de datos.

Características de las variables

Las variables con tienen valores de datos. Al detener el programa y volverlo a poner en marcha, la variable conserva su valor, pero si se mueve el puntero de programa a Main el valor del dato de la variable se pierde.

Declaración de una variable

La declaración de una variable es la forma de definir un nombre de variable y determina el tipo de dato que debe tener. Las variables se declaran con la palabra clave `VAR`, siguiendo la sintaxis:

```
VAR datatype identifier;
```

Ejemplo

```
VAR num length;
VAR string name;
VAR bool finished;
```

Asignación de valores

La asignación de un valor a una variable se hace con la instrucción `:=`

```
length := 10;
name := "John"
finished := TRUE;
```

Recuerde que `:=` no es un signo de igual que. Significa la asignación de la expresión que aparece a su derecha de la variable que aparece a la izquierda. Sólo puede indicarse una única variable a la izquierda de `:=`

Por ejemplo, lo siguiente es un código de RAPID correcto que da como resultado que `reg1` tenga el valor 3:

```
reg1 := 2;
reg1 := reg1 + 1;
```

Continúa en la página siguiente

Continuación

La asignación puede hacerse al mismo tiempo que la declaración de la variable:

```
VAR num length := 10;
VAR string name := "John";
VAR bool finished := TRUE;
```


1.2.2. Variables persistentes

Qué es una variable persistente

Las variables persistentes son básicamente iguales a una variable normal, pero con una diferencia importante. La variable persistente recuerda el último valor que se le haya asignado, incluso si el programa es detenido y puesto en marcha de nuevo desde el principio.

Declaración de una variable persistente

Las variables persistentes se declaran con la palabra clave `PERS`. En el momento de la declaración es necesario indicar un valor inicial.

```
PERS num nbr := 1;
PERS string string1 := "Hello";
```

Ejemplo

Considere el ejemplo de código siguiente:

```
PERS num nbr := 1;
PROC main()
  nbr := 2;
ENDPROC
```

Si se ejecuta este programa, el valor inicial cambia a 2. La próxima vez que se ejecute el programa, el código del programa tendrá el aspecto siguiente:

```
PERS num nbr := 2;
PROC main()
  nbr := 2;
ENDPROC
```

1.2.3. Constantes

¿Qué es una constante?

Las constantes contienen valores, como cualquier variable, pero el valor se asigna siempre en el momento de la declaración y posteriormente no es posible cambiar el valor en ningún caso. La constante puede usarse en el programa de la misma forma que una variable, si bien no se permite asignarle un nuevo valor.

Declaración de constantes

La constante se declara con la palabra clave `CONST` seguida del tipo de dato, el identificador y la asignación de un valor.

```
CONST num gravity := 9.81;
CONST string greating := "Hello"
```

¿Por qué usar constantes?

Al usar una constante en lugar de una variable, puede asegurarse de que el valor no sea cambiado en alguna parte del programa.

El uso de una constante en lugar de escribir el valor directamente en el programa es más adecuado si necesita actualizar el programa con otro valor de la constante. A partir de ese momento, sólo tiene que cambiarlo en un lugar y tener la garantía de que no ha olvidado ninguna aparición del valor.

1.2.4. Operadores

Operadores numéricos

Estos operadores operan con el tipo de datos `num` y devuelven el tipo de datos `num`. Es decir, en los ejemplos siguientes, las variables `reg1`, `reg2` y `reg3` son del tipo de dato `num`.

Operador	Descripción	Ejemplo
+	Suma	`reg1 := reg2 + reg3;`
-	Resta Menos unario	`reg1 := reg2 - reg3;` `reg1 := -reg2;`
*	Multiplicación	`reg1 := reg2 * reg3;`
/	División	`reg1 := reg2 / reg3;`

Operadores relacionales

Estos operadores devuelven el tipo de dato `bool`.

En los ejemplos, `reg1` y `reg2` son del tipo de dato `num` mientras que `flag1` es del tipo `bool`.

Operador	Descripción	Ejemplo
=	igual a	`flag1 := reg1 = reg2;` `flag1` es `TRUE` si `reg1` es igual a `reg2`
<	menor que	`flag1 := reg1 < reg2;` `flag1` es `TRUE` si `reg1` es menor que `reg2`
>	mayor que	`flag1 := reg1 > reg2;` `flag1` es `TRUE` si `reg1` es mayor que `reg2`
<=	menor que o igual a	`flag1 := reg1 <= reg2;` `flag1` es `TRUE` si `reg1` es menor que o igual a `reg2`
>=	mayor que o igual a	`flag1 := reg1 >= reg2;` `flag1` es `TRUE` si `reg1` es mayor que o igual a `reg2`
<>	distinto de	`flag1 := reg1 <> reg2;` `flag1` es `TRUE` si `reg1` es distinto de `reg2`

Los operadores lógicos se usan con frecuencia junto con la instrucción `IF`. Para ver ejemplos de código, consulte *Ejemplos con condiciones lógicas y sentencias IF en la página 19*.

Operador de cadena

Operador	Descripción	Ejemplo

+	Concatenación de cadenas	`VAR string firstname := "John";`
		`VAR string lastname := "Smith";`
		`VAR string fullname;`
		`fullname := firstname + " " + lastname;`
		La variable `fullname` contendrá la cadena "John Smith".

1.3 Control del flujo del programa

1.3.1. IF THEN

Acerca del flujo del programa

Los ejemplos de programa que hemos visto hasta ahora se ejecutan secuencialmente, de arriba abajo. En los programas más complejos, es posible que deseemos controlar qué código se ejecuta, en qué orden y cuántas veces. En primer lugar echaremos un vistazo a cómo crear las condiciones necesarias para determinar si una secuencia de programa debe ejecutarse o no.

IF

La instrucción `IF` puede usarse cuando un conjunto de sentencias sólo debe ejecutarse si se cumple una condición determinada.

Si la condición lógica de la sentencia `IF` se cumple, el código de programa situado entre las palabras clave `THEN` y `ENDIF` se ejecuta. Si la condición no se cumple, ese código no se ejecuta y la ejecución continúa después de `ENDIF`.

Ejemplo

En este ejemplo, la cadena `string1` se escribe en el FlexPendant si no es una cadena vacía. Si `string1` es una cadena vacía, es decir, no contiene ningún carácter, no se realiza ninguna acción.

```
VAR string string1 := "Hello";

IF string1 <> "" THEN
  TPWrite string1;
ENDIF
```

ELSE

Una sentencia `IF` también puede contener código de programa para su ejecución si la condición no se cumple.

Si la condición lógica de la sentencia `IF` se cumple, el código de programa situado entre las palabras clave `THEN` y `ELSE` se ejecuta. Si la condición no se cumple, el código situado entre las palabras clave `ELSE` y `ENDIF` se ejecuta.

Ejemplo

En este ejemplo, la cadena `string1` se escribe en el FlexPendant si no es una cadena vacía. Si `string1` es una cadena vacía, se escribe el texto "The string is empty".

```
VAR string string1 := "Hello";

IF string1 <> "" THEN
  TPWrite string1;
ELSE
  TPWrite "The string is empty";
ENDIF
```


Continúa en la página siguiente

Continuación

ELSEIF

En ocasiones existen más de dos secuencias de programa alternativas. En este caso, puede usar `ELSEIF` para crear distintas alternativas.

Ejemplo

En este ejemplo se escriben textos diferentes en función del valor de la variable `time`.

```
VAR num time := 38.7;

IF time < 40 THEN
  TPWrite "Part produced at fast rate";
ELSEIF time < 60 THEN
  TPWrite "Part produced at average rate";
ELSE
  TPWrite "Part produced at slow rate";
ENDIF
```

Observe que, dado que la primera condición se cumple, se escribe el primer texto. Los otros dos textos no se escribirán (aunque es cierto que `time` tiene un valor inferior a 60).

1.3.2. Ejemplos con condiciones lógicas y sentencias IF

Ejemplo

Usar la sentencia `IF` para determinar qué texto debe escribir en el FlexPendant. Escribir en el FlexPendant qué pieza se produce a mayor velocidad.

```
VAR string part1 := "Shaft";
VAR num time1;
VAR string part2 := "Pipe";
VAR num time2;

PROC main()
  time1 := 41.8;
  time2 := 38.7;
  IF time1 < time2 THEN
     TPWrite part1 + " is fastest to produce";
  ELSEIF time1 > time2 THEN
     TPWrite part2 + " is fastest to produce";
  ELSE
     TPWrite part1 + " y " + part2 + " are equally fast to produce";
  ENDIF
ENDPROC
```

Ejemplo

Si la producción de una pieza requiere más de 60 segundos, escribir un mensaje en el FlexPendant. Si la variable booleana `full_speed` contiene `FALSE`, el mensaje indicará al operador que debe incrementar la velocidad del robot. Si `full_speed` contiene `TRUE`, el mensaje pedirá al operador que examine el motivo de la lentitud de la producción.

```
VAR num time := 62.3;
VAR bool full_speed := TRUE;

PROC main()
  IF time > 60 THEN
     IF full_speed THEN
        TPWrite "Examine why the production is slow";
     ELSE
        TPWrite "Increase robot speed for faster production";
     ENDIF
  ENDIF
ENDPROC
```


1.3.3. Bucle FOR

Repetición de una secuencia de código

Otra forma de controlar el flujo del programa es repetir una secuencia de código del programa un número determinado de veces.

Cómo funciona el bucle FOR

El código siguiente repetirá 5 veces la escritura de "Hola":

```
FOR i FROM 1 TO 5 DO
  TPWrite "Hello";
ENDFOR
```

La sintaxis de la sentencia `FOR` es la siguiente:

```
FOR contador FROM valorinicial TO valorfinal DO
  código de programa a repetir
ENDFOR
```

El valor del *contador* no tiene por qué declararse, pero actúa como una variable numérica dentro del bucle `FOR`. La primera vez que se ejecuta el código, el *contador* tiene el valor especificado por *valorinicial*. A continuación, el valor del *contador* se incrementa en 1 cada vez que se ejecuta el código. La última vez que se ejecuta el código es cuando el valor de *contador* iguala a *valorfinal*. A continuación, la ejecución continúa con el código de programa que aparezca a continuación de `ENDFOR`.

Utilización del valor del contador

El valor del *contador* puede usarse en el bucle `FOR`.

Por ejemplo, el cálculo de la suma de todos los números de 1 a 50 (1+2+3+...+49+50) puede programarse de la forma siguiente:

```
VAR num sum := 0;

FOR i FROM 1 TO 50 DO
  sum := sum + i;
ENDFOR
```

No se permite asignar un valor para el *contador* dentro del bucle `FOR`.

1.3.4. Bucle WHILE

Repetición con una condición

La repetición de una secuencia de código puede combinarse con la ejecución condicional de la secuencia de código. Con el bucle `WHILE`, el programa seguirá ejecutando la secuencia de código siempre y cuando se cumpla la condición.

Sintaxis de WHILE

La sintaxis del bucle `WHILE` es la siguiente:

```
WHILE condición DO
  código de programa a repetir
ENDWHILE
```

Si la *condición* no se cumple desde el principio, la secuencia de código no llega a ejecutarse nunca. Si la *condición* se cumple, la secuencia de código se ejecuta repetidamente hasta que la *condición* deja de cumplirse.

Ejemplo

El código de programa siguiente suma números (1+2+3+...) hasta que la suma llegue a 100.

```
VAR num sum := 0;
VAR num i := 0;

WHILE sum <= 100 DO
  i := i + 1;
  sum := sum + i;
ENDWHILE
```

No cree bucles eternos o pesados sin una instrucción wait.

Si la condición nunca deja de cumplirse, el bucle continuará constantemente y consumirá enormes cantidades de potencia del ordenador. La escritura de un bucle eterno está permitida, pero en ese caso debe contener alguna instrucción de espera que permita al ordenador realizar otras tareas entretanto.

Los bucles pesados (con gran cantidad de cálculos y escrituras en el FlexPendant, sin instrucciones de movimiento) pueden requerir alguna instrucción de espera incluso si existe un número limitado de bucles.

```
WHILE TRUE DO
  ! Some code
  ...
  ! Wait instruction that waits for 1 second
  WaitTime 1;
ENDWHILE
```

Recuerde que las instrucciones de movimiento funcionan como instrucciones de espera, dado que la ejecución no continúa hasta que el robot ha alcanzado su objetivo.

1.4 Reglas y recomendaciones de sintaxis de RAPID

1.4.1. Reglas generales de sintaxis de RAPID

Punto y coma

La regla general es que cada sentencia termina con un punto y coma.

Ejemplos

Declaración de variables:

```
VAR num length;
```

Asignación de valores:

```
area := length * width;
```

Mayoría de llamadas a instrucciones:

```
MoveL p10,v1000,fine,tool0;
```

Excepciones

Algunas instrucciones especiales no terminan con un punto y coma. En su lugar, existen palabras clave especiales para indicar dónde terminan.

Ejemplos de instrucciones que no terminan con un punto y coma:

Palabra clave de instrucción	Palabra clave final
IF	ENDIF
FOR	ENDFOR
WHILE	ENDWHILE
PROC	ENDPROC

Estas palabras clave son muy importantes para crear una buena estructura en un programa de RAPID. Se describen en detalle más adelante en este manual.

Comentarios

Las líneas que comienzan con ! no son interpretadas por el controlador de robot. Utilícelo para escribir comentarios acerca del código.

Ejemplo

```
! Calculate the area of the rectangle
area := length * width;
```

1.4.2. Recomendaciones para el código de RAPID

Palabras clave en mayúsculas

RAPID no distingue entre mayúsculas y minúsculas, pero se recomienda que todas las palabras reservadas (por ejemplo `VAR`, `PROC`) se escriban con mayúsculas. Para obtener una lista completa de palabras reservadas, consulte el *Manual de referencia técnica - Descripción general de RAPID*.

Indentaciones

Para facilitar la comprensión del código de programación, utilice indentaciones. Todo lo que esté dentro de un procedimiento `PROC` (entre `PROC` y `ENDPROC`) debe estar indentado. Todo lo que esté dentro de una sentencia `IF`-, `FOR`- o `WHILE` debe estar aún más indentado.

Al programar con el FlexPendant, la indentación se realiza automáticamente.

Ejemplo

```
VAR bool repeat;
VAR num times;

PROC main()
  repeat := TRUE;
  times := 3;
  IF repeat THEN
     FOR i FROM 1 TO times DO
        TPWrite "Hello!";
     ENDFOR
  ENDIF
END PROC
```

Observe que es fácil comprobar dónde comienza y termina la sentencia `IF`. Si tiene varias sentencias `IF` sin ninguna indentación, sería prácticamente imposible saber qué `ENDIF` corresponde a cada `IF`.

1.4.2. Recomendaciones para el código de RAPID

2 Funcionalidad de robots en RAPID

2.1 Instrucciones de movimiento

2.1.1. Instrucción MoveL

Descripción general

La ventaja de RAPID es que, excepto por el hecho de que presenta una funcionalidad que aparece en otros lenguajes de programación de alto nivel, ha sido diseñado especialmente para controlar robots. Y lo que es más importante, existen instrucciones para hacer que el robot se mueva.

MoveL

Una instrucción de movimiento simple puede tener el aspecto siguiente:

```
MoveL p10, v1000, fine, tool0;
```

Donde:

- `MoveL` es una instrucción que mueve el robot linealmente (en una línea recta) desde su posición actual hasta la posición especificada.
- `p10` especifica la posición hasta la que debe moverse el robot.
- `v1000` especifica que la velocidad del robot debe ser de 1.000 mm/s.
- `fine` especifica que el robot debe moverse exactamente hasta la posición especificada y no tomar atajos en las esquinas en su desplazamiento hasta la siguiente posición.
- `tool0` especifica que lo que debe moverse hasta la posición especificada es la brida de montaje situada en el extremo del robot.

Sintaxis de MoveL

```
MoveL ToPoint Speed Zone Tool;
```

ToPoint

El punto de destino, definido por una constante del tipo de dato `robtarget`. Al programar con el FlexPendant, puede asignar un valor `robtarget` apuntando a una posición con el robot. Al programar fuera de línea, puede resultar complicado calcular las coordenadas de una posición.

`robtarget` se explica más adelante, en la sección *Tipos de datos compuestos en la página 48*. De momento, aceptemos simplemente que la posición x=600, y=-100, z=800 puede declararse y asignarse de la forma siguiente:

```
CONST robtarget p10 := [ [600, -100, 800], [1, 0, 0, 0], [0, 0, 0,
     0], [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9] ];
```

Speed

La velocidad del movimiento, definida por una constante del tipo de dato `speeddata`. Existe un buen número de valores predefinidos, como por ejemplo:

Valores predefinidos de speeddata	Valor
v5	5 mm/s
v100	100 mm/s

Continúa en la página siguiente

Continuación

Valores predefinidos de speeddata	Valor
v1000	1.000 mm/s
vmax	Velocidad máxima del robot

Encontrará una lista completa de valores predefinidos de `speeddata` en el *Manual de referencia técnica - RAPID Instrucciones, funciones y tipos de datos*, sección *Tipos de datos* y *speeddata*.

Al utilizar un valor predefinido, no debe declararlo ni asignarlo.

Zone

Especifica una zona de esquina definida por una constante del tipo de dato `zonedata`. Existe muchos valores predefinidos, como por ejemplo:

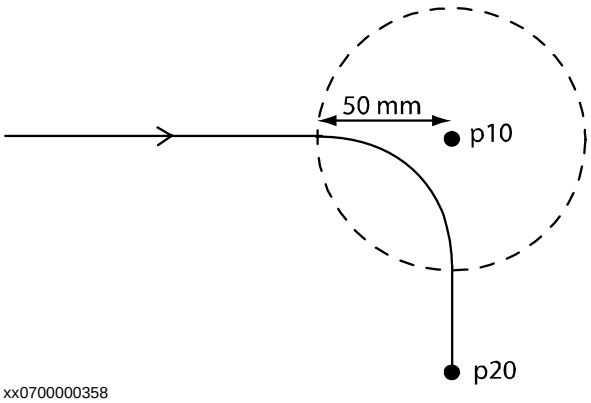
Valores predefinidos de zonedata	Valor
fine	El robot irá exactamente hasta la posición especificada.
z10	La trayectoria del robot puede tomar atajos en las esquinas si está a menos de 10 mm de *ToPoint*.
z50	La trayectoria del robot puede tomar atajos en las esquinas si está a menos de 50 mm de *ToPoint*.

Encontrará una lista completa de valores predefinidos de `zonedata` en el *Manual de referencia técnica - RAPID Instrucciones, funciones y tipos de datos*, sección *Tipos de datos* y *zonedata*.

Al utilizar un valor predefinido, no debe declararlo ni asignarlo.

Las siguientes instrucciones de RAPID darán lugar a la trayectoria de robot mostrada a continuación:

```
MoveL p10, v1000, z50, tool0;
MoveL p20, v1000, fine, tool0;
```



xx0700000358

Continúa en la página siguiente

Continuación

Tool

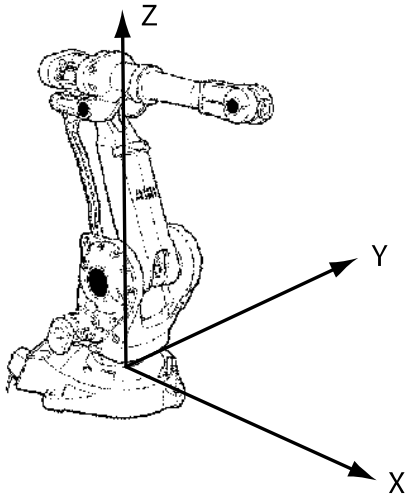
Especifica la herramienta utilizada por el robot, definida por una variable persistente del tipo de dato `tooldata`. Si hay una pistola de soldadura, una pistola de adhesivo o un rotulador fijado al robot, queremos programar el *ToPoint* de la punta de esta herramienta. Esto se realiza automáticamente si se declara, asigna y utiliza un valor de `tooldata` en la instrucción `MoveL`.

`tool0` es una herramienta predefinida que representa al robot sin ninguna herramienta montada en él y no debe ser declarada ni asignada. Cualquier otra herramienta debe ser declarada y asignada antes de usarla.

2.1.2. Sistemas de coordenadas

Sistema de coordenadas de la base

La posición de destino de una instrucción de movimiento se define en forma de coordenadas de un sistema de coordenadas. Si no se especifica ningún sistema de coordenadas, la posición se indica de forma relativa al sistema de coordenadas de la base del robot (también conocido como base de coordenadas de la base). El sistema de coordenadas de la base tiene su origen en la base del robot.



xx0700000397

Sistemas de coordenadas personalizados

Es posible definir y utilizar otro sistema de coordenadas con las instrucciones de movimiento. El sistema de coordenadas que la instrucción de movimiento debe utilizar se especifica con el argumento opcional `\WObj`.

```
MoveL p10, v1000, z50, tool0 \WObj:=wobj1;
```

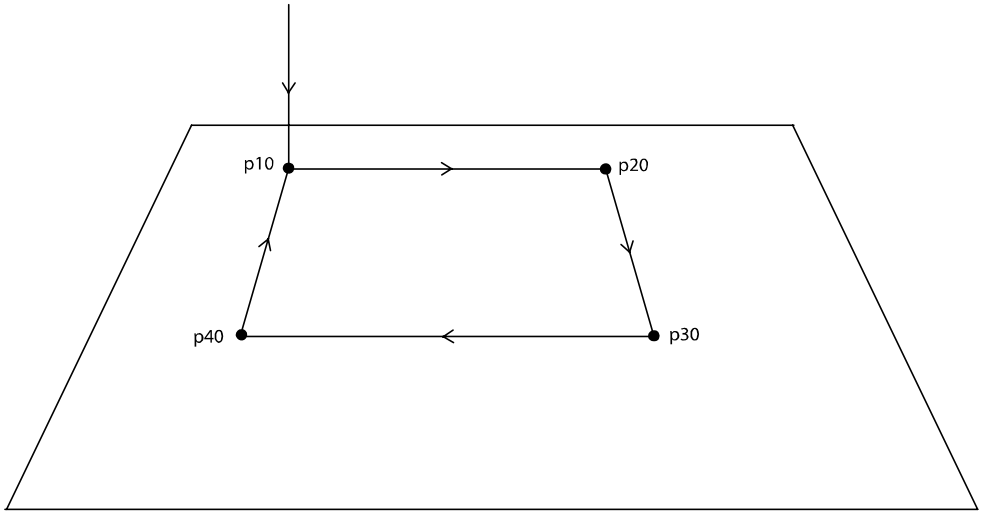
Para obtener más información acerca de cómo definir un sistema de coordenadas, consulte el *Manual de referencia técnica - RAPID Instrucciones, funciones y tipos de datos*, sección *Tipos de datos* y *wobjdata*.

Para obtener más información acerca de los sistemas de coordenadas, consulte el *Manual de referencia técnica - Descripción general de RAPID*, sección *Sistemas de coordenadas*.

2.1.3. Ejemplos con MoveL

Trazar un cuadrado

Un robot sostiene un rotulador sobre un papel colocado sobre una mesa. Queremos que el robot baje la punta del rotulador para colocarla en el papel y que a continuación trace un cuadrado.



xx0700000362

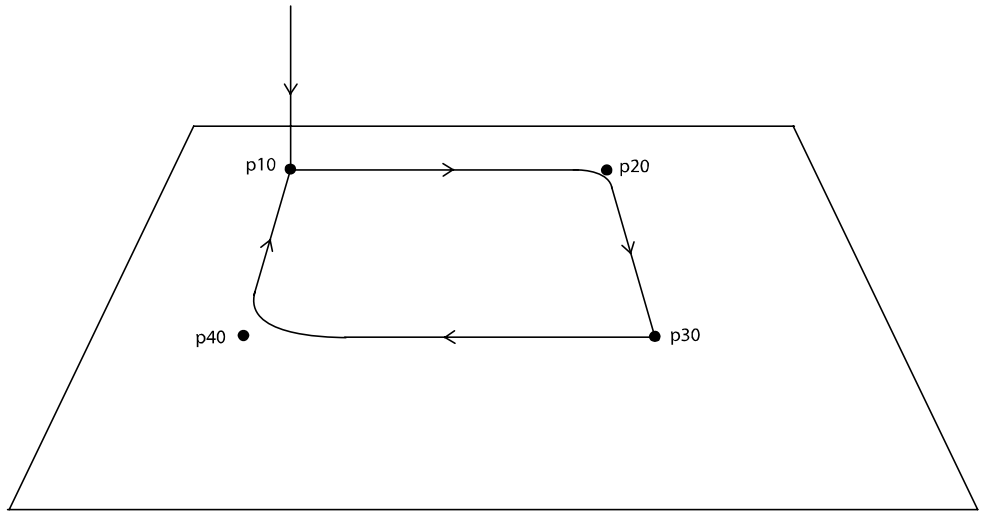
```
PERS tooldata tPen := [ TRUE, [[200, 0, 30], [1, 0, 0 ,0]], [0.8,
     [62, 0, 17], [1, 0, 0, 0], 0, 0, 0]];
CONST robtarget p10 := [ [600, -100, 800], [0.707170, 0, 0.707170,
     0], [0, 0, 0, 0], [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9] ];
CONST robtarget p20 := [ [600, 100, 800], [0.707170, 0, 0.707170,
     0], [0, 0, 0, 0], [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9] ];
CONST robtarget p30 := [ [800, 100, 800], [0.707170, 0, 0.707170,
     0], [0, 0, 0, 0], [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9] ];
CONST robtarget p40 := [ [800, -100, 800], [0.707170, 0, 0.707170,
     0], [0, 0, 0, 0], [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9] ];

PROC main()
  MoveL p10, v200, fine, tPen;
  MoveL p20, v200, fine, tPen;
  MoveL p30, v200, fine, tPen;
  MoveL p40, v200, fine, tPen;
  MoveL p10, v200, fine, tPen;
ENDPROC
```


Continúa en la página siguiente

Continuación

Trazar con zonas de esquina

Se traza la misma figura que en el ejemplo anterior, pero con una zona de esquina de 20 mm en `p20` y una zona de esquina de 50 mm en `p40`.



xx0700000363

```
VAR tooldata tPen := ...
...
VAR robtarget p40 := ...

PROC main()
  MoveL p10, v200, fine, tPen;
  MoveL p20, v200, z20, tPen;
  MoveL p30, v200, fine, tPen;
  MoveL p40, v200, z50, tPen;
  MoveL p10, v200, fine, tPen;
ENDPROC
```

2.1.4. Otras instrucciones de movimiento

Distintas instrucciones de movimiento

RAPID cuenta con distintas instrucciones de movimiento. Las más comunes son `MoveL`, `MoveJ` y `MoveC`.

MoveJ

`MoveJ` se utiliza para mover el robot rápidamente de un punto a otro cuando no es imprescindible que el movimiento siga una línea recta.

Utilice `MoveJ` para mover el robot hasta un punto en el aire situado a poca distancia de donde va a trabajar el robot. La instrucción `MoveL` no funciona si, por ejemplo, la base del robot está entre la posición actual y la posición programada o si la reorientación de la herramienta es demasiado grande. `MoveJ` puede usarse siempre en estos casos.

La sintaxis de `MoveJ` es similar a la de `MoveL`.

Ejemplo

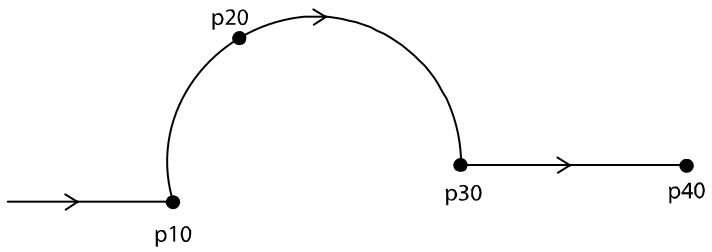
```
MoveJ p10, v1000, fine, tPen;
```

MoveC

`MoveC` se usa para mover el robot en círculo describiendo un arco.

Ejemplo

```
MoveL p10, v500, fine, tPen;
MoveC p20, p30, v500, fine, tPen;
MoveL p40, v500, fine, tPen;
```



xx0700000364

2.1.5. Comportamiento de ejecución en las zonas de esquina

¿Por qué se requiere una ejecución especial en las zonas de esquina?

La ejecución de un programa suele realizarse en el orden en el que se escriben las sentencias.

En el ejemplo siguiente, el robot se mueve en primer lugar a `p10`, luego calcula el valor de `reg1` y por último se mueve hasta `p20`:

```
MoveL p10, v1000, fine, tool0;
reg1 := reg2 + reg3;
MoveL p20, v1000, fine, tool0;
```

Sin embargo, observe ahora el ejemplo siguiente:

```
MoveL p10, v1000, z50, tool0;
reg1 := reg2 + reg3;
MoveL p20, v1000, fine, tool0;
```

Si el cálculo de `reg1` no comenzara hasta que el robot se encontrara en `p10`, el robot tendría que detenerse en ese punto y esperar a la siguiente instrucción de movimiento. Lo que ocurre en realidad es que el código se ejecuta con antelación al movimiento del robot. Se calcula `reg1` y se calcula la trayectoria del robot en la zona de esquina antes de que el robot alcance `p10`.

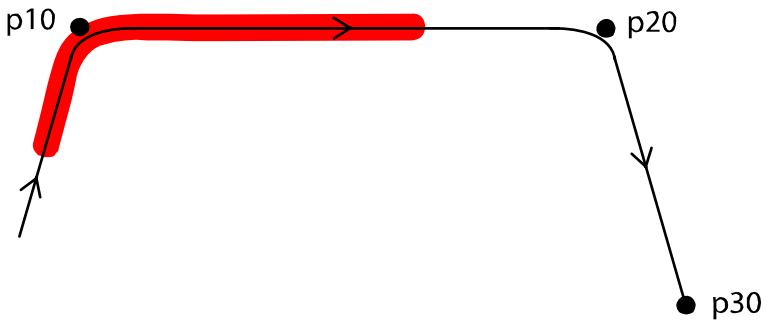
Cómo afecta esto a mi programa

En muchos casos, el momento exacto de la ejecución no afecta al programa. Sin embargo, hay ejemplos en los que sí afecta al programa.

Si desea trazar una línea con una pistola pulverizadora entre `p10` y `p20` y escribe el programa de la forma siguiente:

```
MoveL p10, v300, z10, tspray;
! Empezar a pulverizar
SetDO do1, 1;
MoveL p20, v300, z10, tspray;
! Dejar de pulverizar
SetDO do1, 0;
MoveL p30, v300, fine, tspray;
```

El resultado puede parecerse a lo siguiente:



xx0700000387

Continúa en la página siguiente

Solución

Si desea activar señales en las zonas de esquina y no utiliza `fine`, existen instrucciones especiales para resolverlo, por ejemplo `MoveLDO`, `TriggL` y `DispL`. Para obtener más información acerca de estas instrucciones, consulte el *Manual de referencia técnica - RAPID Instrucciones, funciones y tipos de datos*.

Evite las instrucciones de espera o los cálculos pesados a continuación de las zonas de esquina

El controlador del robot puede calcular el movimiento del robot en las trayectorias de esquina incluso si hay instrucciones entre las instrucciones de movimiento. Sin embargo, si hay una instrucción de espera a continuación de una instrucción de movimiento con una zona de esquina, el robot no será capaz de gestionarlo. Utilice `fine` en la instrucción de movimiento antes de una instrucción de espera.

También existe una limitación en cuanto al número (y la complejidad) de los cálculos que puede realizar el controlador de robot entre las instrucciones de movimiento con zonas de esquina. Esto constituye principalmente un problema al llamar a procedimientos a continuación de una instrucción de movimiento con una zona de esquina.

2.2 Señales de E/S

2.2.1. Señales de E/S

Acerca de las señales

Las señales se utilizan para la comunicación con los equipos externos con los que coopera el robot. Las señales de entrada son activadas por los equipos externos y pueden usarse en el programa de RAPID para iniciar la realización de alguna operación con el robot. Las señales de salida son activadas por el programa de RAPID y actúan para comunicar a los equipos externos que deben hacer algo.

Configuración de señales

Las señales son configuradas en los parámetros de sistema del sistema de robot. Es posible definir nombres personalizados para las señales. No deben declararse en el programa de RAPID.

Entrada digital

Las señales digitales de entrada pueden tener los valores 0 ó 1. El programa de RAPID puede leer su valor pero no cambiarlo.

Ejemplo

Si la señal digital de entrada `di1` tiene el valor 1, el robot se mueve.

```
IF di1 = 1 THEN
  MoveL p10, v1000, fine, tPen;
ENDIF
```

Salida digital

Las señales digitales de salida pueden tener los valores 0 ó 1. El programa de RAPID puede establecer el valor de una señal digital de salida y con ello influir en el comportamiento de los equipos externos. El valor de una señal digital de salida se establece con la instrucción `SetDO`.

Ejemplo

El robot cuenta con una herramienta pinza que puede cerrarse con la señal digital de salida `do_grip`. El robot se mueve hasta la posición en la que se encuentra el rotulador y cierra la pinza. A continuación, el robot se mueve hasta donde debe trazar, ahora usando la herramienta `tPen`.

```
MoveJ p0, vmax, fine, tGripper;
SetDO do_grip, 1;
MoveL p10, v1000, fine, tPen;
```

Otros tipos de señales

Las señales digitales son las más comunes y fáciles de usar. Si una señal necesita tener un valor distinto de 0 ó 1, existen señales analógicas y grupos de señales digitales que pueden tener otros valores. Estos tipos de señales no se tratan en este manual.

2.3 Interacción con el usuario

2.3.1. Comunicación con el FlexPendant

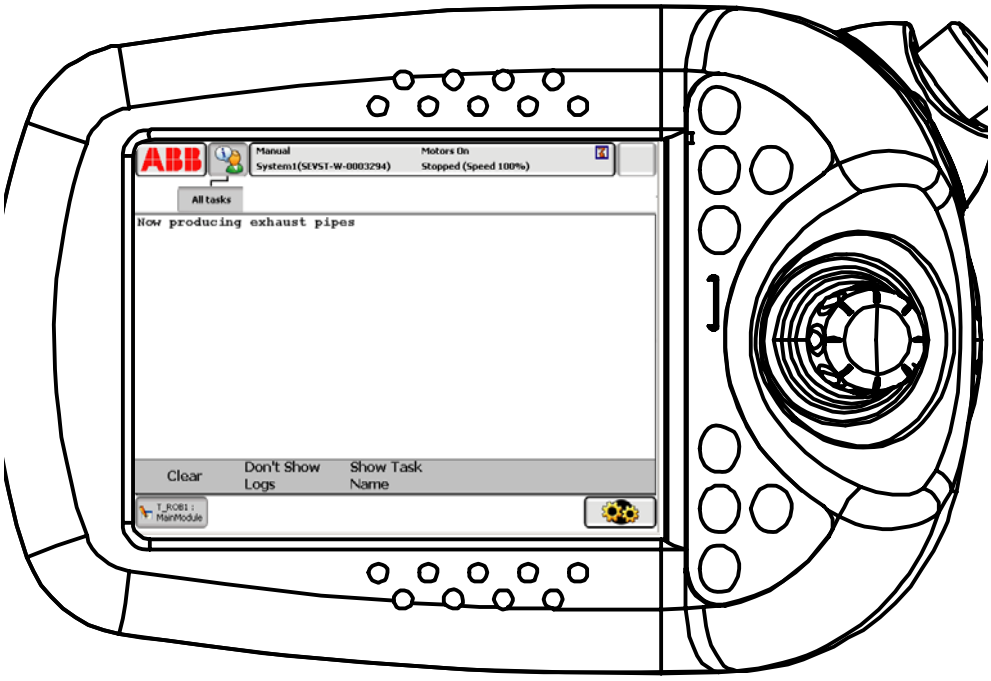
Acerca de las instrucciones de lectura y escritura

RAPID cuenta con varias instrucciones para escribir información para el operador del robot, así como para recibir información del operador. Ya hemos visto `TPWrite` en los ejemplos anteriores. Las únicas instrucciones que estudiaremos aquí son `TPWrite`, `TPReadFK` y `TPReadNum`.

TPWrite

La escritura de un mensaje para el operador puede hacerse con la instrucción `TPWrite`.

```
TPWrite "Now producing exhaust pipes";
```



xx0700000374

Escrita de una variable de cadena

La cadena de texto escrita en el FlexPendant puede provenir de una variable de cadena. El texto escrito también puede ser el resultado de concatenar una variable de cadena con otra cadena.

```
VAR string product := "exhaust pipe";
! Write only the product on the FlexPendant
TPWrite product;
! Write "Producing" and the product on the FlexPendant
TPWrite "Producing " + product;
```


Continúa en la página siguiente

Continuación

Escritura de una variable numérica

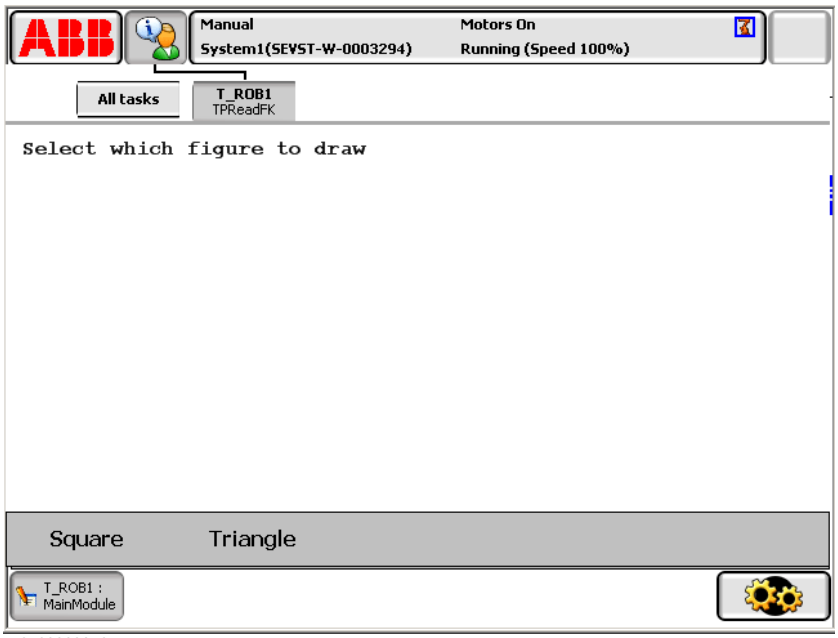
Es posible añadir una variable numérica a continuación de la cadena utilizando el argumento opcional `\Num`.

```
VAR num count := 13;
TPWrite "The number of produced units is: " \Num:=count;
```

TPReadFK

Al escribir un programa de RAPID que requiere que el operador elija una opción, `TPReadFK` es una instrucción útil. Permite la visualización de hasta cinco teclas de función y el operador puede elegir cuál de ellas tocar. Los botones corresponden a los valores 1 a 5.

```
VAR num answer;
TPReadFK answer, "Select which figure to draw", "Square",
     "Triangle", stEmpty, stEmpty, stEmpty;
IF answer = 1 THEN
  ! code to draw square
ELSEIF answer = 2 THEN
  ! code to draw triangle
ELSE
  ! do nothing
ENDIF
```



xx0700000376

Si el usuario selecciona "Square", la variable numérica `answer` recibe el valor 1. Si el usuario selecciona "Triangle", la variable numérica `answer` recibe el valor 2.

Es posible especificar cinco teclas de función. Si no necesita usar una tecla, escriba `stEmpty` en lugar del texto del botón.

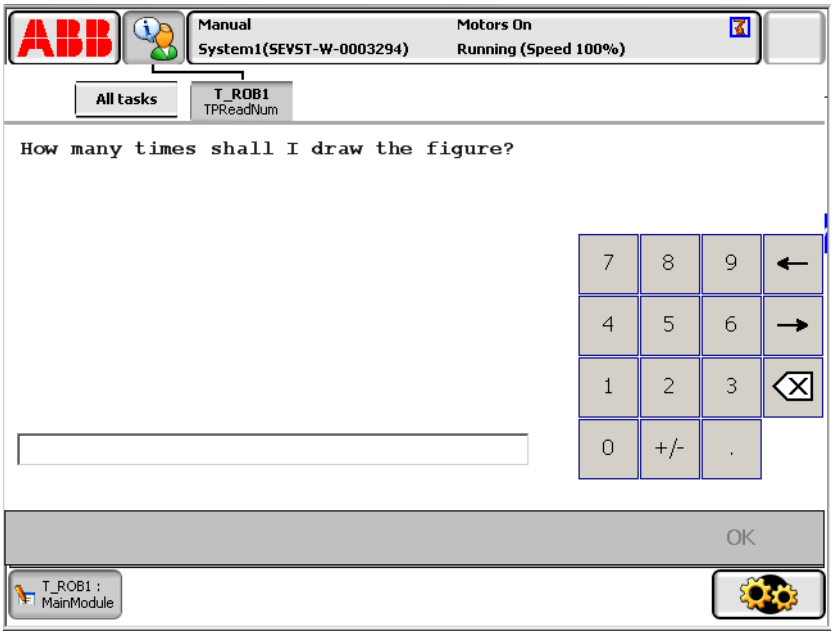
Continúa en la página siguiente

Continuación

TPReadNum

`TPReadNum` permite al operador escribir un número en el FlexPendant, en lugar de elegir una opción.

```
VAR num answer;
TPReadNum answer, "How many times shall I draw the figure?";
FOR i FROM 1 TO answer DO
  ! code to draw figure
ENDFOR
```



xx0700000378

La variable numérica `answer` recibe el valor introducido por el operador.

2.3.1. Comunicación con el FlexPendant

3 Estructura

3.1. Procedimiento de RAPID

Qué es un procedimiento

Hasta el momento, todos los ejemplos de código de RAPID que hemos visto tenían su código ejecutable en el procedimiento `main`. La ejecución comienza automáticamente en el procedimiento llamado `main`, pero es posible disponer de varios procedimientos.

Los procedimientos deben ser declarados con la palabra clave `PROC` seguida del nombre del procedimiento, los argumentos del procedimiento y el código de programa que debe ser ejecutado por el procedimiento. La llamada a un procedimiento se hace desde otro procedimiento (excepto en el caso de `main`, al que se llama directamente cuando se inicia el programa).

Ejemplo

Si queremos trazar cuatro cuadrados de tamaños diferentes, podemos escribir cuatro veces prácticamente el mismo código de programa. El resultado sería un montón de código y una mayor dificultad para comprender el programa. Una forma mucho más eficiente de escribir este programa es crear un procedimiento que traza el cuadrado y hacer que el procedimiento main llame cuatro veces a este procedimiento.

```
PERS tooldata tPen:= ...
CONST robtarget p10:= ...

PROC main()
  ! Call the procedure draw_square
  draw_square 100;
  draw_square 200;
  draw_square 300;
  draw_square 400;
ENDPROC

PROC draw_square(num side_size)
  VAR robtarget p20;
  VAR robtarget p30;
  VAR robtarget p40;

  ! p20 is set to p10 with an offset on the y value
  p20 := Offs(p10, 0, side_size, 0);
  p30 := Offs(p10, side_size, side_size, 0);
  p40 := Offs(p10, side_size, 0, 0);

  MoveL p10, v200, fine, tPen;
  MoveL p20, v200, fine, tPen;
  MoveL p30, v200, fine, tPen;
  MoveL p40, v200, fine, tPen;
  MoveL p10, v200, fine, tPen;
```

Continúa en la página siguiente

Continuación

```
ENDPROC
```

Argumentos de procedimientos

Al declarar un procedimiento, todos los argumentos se declaran entre paréntesis a continuación del nombre del procedimiento. Esta declaración contiene el tipo de dato y el nombre de cada argumento. El argumento recibe su valor de la llamada al procedimiento y actúa como una variable dentro del procedimiento (el argumento no puede usarse fuera de su procedimiento).

```
PROC main()
  my_procedure 14, "Hello", TRUE;
ENDPROC

PROC my_procedure(num nbr_times, string text, bool flag)
  ...
ENDPROC
```

Dentro del procedimiento `my_procedure` que aparece arriba, `nbr_times` tiene el valor 14, `text` tiene el valor "Hello" y `flag` tiene el valor `TRUE`.

Al llamar al procedimiento, el orden de los argumentos es importante para asignar el valor correcto al argumento correcto. En la llamada al procedimiento no se usan paréntesis.

Variables declaradas dentro del procedimiento

Las variables declaradas dentro de un procedimiento sólo existen dentro del procedimiento. Es decir, en el ejemplo anterior, `p10` puede usarse en todos los procedimientos de este módulo, pero `p20`, `p30` y `p40` sólo pueden usarse en el procedimiento `draw_square`.

3.2. Módulos

Acerca de los módulos

Los programas de RAPID pueden constar de uno o varios módulos. Cada módulo puede contener uno o varios procedimientos.

Los programas pequeños y sencillos mostrados en este manual sólo utilizan un único módulo. En un entorno de programación más complejo, algunos procedimientos estándar utilizados por muchos programas diferentes pueden ser colocados en un módulo separado.

Ejemplo

El módulo `MainModule` contiene código específico de este programa y especifica qué debe hacer el robot en este programa en concreto. El módulo `figures_module` contiene código estándar que puede usarse en cualquier programa en el que se desee trazar un cuadrado, un triángulo o un círculo.

```
MODULE MainModule
  ...
  draw_square;
  ...
ENDMODULE

MODULE figures_module
  PROC draw_square()
     ...
  ENDPROC
  PROC draw_triangle()
     ...
  ENDPROC
  PROC draw_circle()
     ...
  ENDPROC
ENDMODULE
```

Módulos de programa

Los módulos de programa se guardan con la extensión de archivo .mod, por ejemplo *figures_module.mod*.

Para el controlador de robot no supone ninguna diferencia que el programa esté escrito en distintos módulos. El motivo de utilizar distintos módulos de programa es sólo facilitar a los programadores la comprensión del programa y la reutilización del código.

Sólo puede haber un único programa activo en el controlador de robot. Es decir, sólo uno de los módulos puede contener un procedimiento llamado `main`.

Continúa en la página siguiente

Continuación

Módulos de sistema

Los módulos de sistema se guardan con la extensión de archivo .sys, por ejemplo *system_data_module.sys*.

Los datos y procedimientos que deban permanecer en el sistema incluso al cambiar de programa deben situarse en un módulo de sistema. Por ejemplo, si se declara una variable persistente del tipo `tooldata` en un módulo de sistema, la recalibración de la herramienta se conserva incluso al cargar un nuevo programa.

3.3. Diseño estructurado

Acerca de la estructura

Al enfrentarse por primera vez a un problema que desea resolver con un programa de RAPID, analice con calma el problema y sus componentes. Si empieza a programar sin primero pensar la totalidad del diseño, su programa será irracional. Un programa bien diseñado tiene una menor probabilidad de contener errores y es más fácil de comprender para otras personas. El tiempo dedicado al diseño se recupera muchas veces durante las pruebas y el mantenimiento del programa.

Divida el problema

Siga estos pasos para dividir el problema en fragmentos manejables:

	Acción
1.	Identifique la funcionalidad a mayor escala. Intente dividir el problema en piezas más pequeñas y fáciles de manejar.
2.	Cree una estructura de diseño. Trace un mapa de las funciones y de cómo se relacionarán entre sí.
3.	Observe los distintos bloques de la estructura de diseño. ¿Es posible dividir un bloqueo en fragmentos más pequeños? ¿Qué se necesita para implementar el bloque?

Ejemplo

Descripción del problema

Crear un programa de RAPID capaz de trazar cuadrados o triángulos en un papel. Dejar que el operador decida si lo siguiente que debe trazarse es un cuadrado o un triángulo. Cuando el robot termina de trazar la figura, el usuario debe ser capaz de hacer la misma selección de nuevo hasta que el operador toca un botón Salir.

Cuando el robot ha trazado 10 figuras en el mismo papel, escribir un mensaje para indicar que se debe sustituir el papel y esperar a que el operador toque un botón OK.

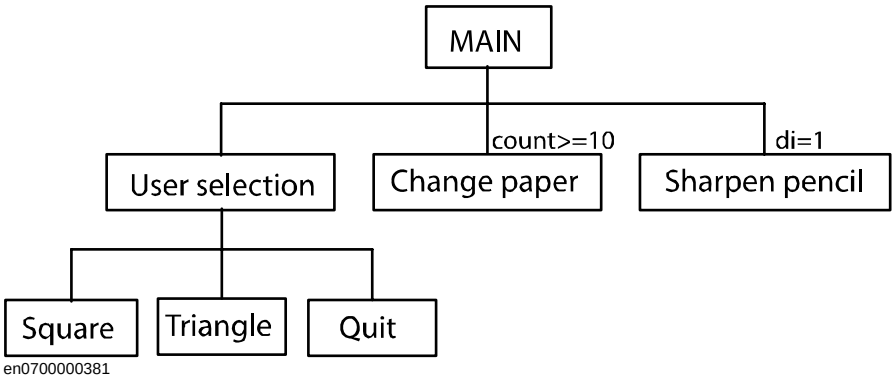
Entre un dibujo y el siguiente, comprobar si `di1` tiene el valor 1. Si es así, moverse hasta un sacapuntas y cambiar `do1` a 1 para poner en marcha el sacapuntas e introducir lentamente el lápiz en el sacapuntas. Normalmente, necesitamos redefinir la herramienta dado que se acorta al afilarla, pero en este ejemplo omitiremos ese paso.

Continúa en la página siguiente

3.3. Diseño estructurado

Continuación

Estructura de diseño



en0700000381

Código del programa

```
MODULE MainModule
  PERS tooldata tPen := [ TRUE, [[200, 0, 30], [1, 0, 0 ,0]], [0.8,
       [62, 0, 17], [1, 0, 0, 0], 0, 0, 0]];
  CONST robtarget p10 := [ [600, -100, 800], [0.707170, 0,
       0.707170, 0], [0, 0, 0, 0], [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9,
       9E9] ];
  CONST robtarget pSharp1 := [ [200, 500, 850], [1, 0, 0, 0], [0,
       0, 0, 0], [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9, 9E9] ];
  PERS num count := 0;

  PROC main()
     user_selection;
     IF count >= 10 THEN
        change_paper;
        ! Reset count
        count := 0;
     ENDIF
     IF di=1 THEN
        sharpen_pencil;
     ENDIF
  ENDPROC

  PROC user_selection()
     VAR num answer;
     TPReadFK answer, "Select which figure to draw", "Square",
          "Triangle", "Quit", stEmpty, stEmpty;
     IF answer = 1 THEN
        draw_square;
        count := count + 1;
     ELSEIF answer = 2 THEN
        draw_triangle;
        count := count + 1;
     ELSE
        quit;
     ENDIF
```

Continúa en la página siguiente

Continuación

```
ENDPROC

PROC draw_square()
   VAR robtarget p20;
   VAR robtarget p30;
   VAR robtarget p40;

   ! Define points that give a square with the side 200 mm
   p20 := Offs(p10, 0, 200, 0);
   p30 := Offs(p10, 200, 200, 0);
   p40 := Offs(p10, 200, 0, 0);

   MoveL p10, v200, fine, tPen;
   MoveL p20, v200, fine, tPen;
   MoveL p30, v200, fine, tPen;
   MoveL p40, v200, fine, tPen;
   MoveL p10, v200, fine, tPen;
ENDPROC

PROC draw_triangle()
   VAR robtarget p20;
   VAR robtarget p30;

   ! Define points for the triangle
   p20 := Offs(p10, 0, 200, 0);
   p30 := Offs(p10, 200, 100, 0);

   MoveL p10, v200, fine, tPen;
   MoveL p20, v200, fine, tPen;
   MoveL p30, v200, fine, tPen;
   MoveL p10, v200, fine, tPen;
ENDPROC

PROC quit()
   TPWrite "Good bye!"
   ! Terminate the program
   EXIT;
ENDPROC

PROC change_paper()
   VAR num answer;
   TPReadFK answer, "Change the paper", "OK", stEmpty, stEmpty,
        stEmpty, stEmpty;
ENDPROC
```


Continúa en la página siguiente

Continuación

```
PROC sharpen_pencil()
   VAR robtarget pSharp2;
   VAR robtarget pSharp3;

   pSharp2 := Offs(pSharp1, 100, 0, 0);
   pSharp3 := Offs(pSharp1, 120, 0, 0);
   ! Move quickly to position in front of sharpener
   MoveJ pSharp1, vmax, z10, tPen;
   ! Place pencil in sharpener
   MoveL pSharp2, v500, fine, tPen;
   ! Start the sharpener
   SetDO do1, 1;
   ! Slowly move into the sharpener
   MoveL pSharp3, v5, fine, tPen;
   ! Turn off sharpener
   SetDO do1, 0;
   ! Move out of sharpener
   MoveL pSharp1, v500, fine, tPen;
 ENDPROC
ENDMODULE
```

Recuerde que en una producción el programa se ejecuta normalmente en el modo continuo, de forma que cuando la ejecución llega al final del procedimiento, el procedimiento `main` comienza de nuevo desde el principio. Si no es así, es posible utilizar un bucle `WHILE` para repetir todo el contenido del procedimiento `main`.

4 Datos con varios valores

4.1. Matrices

Qué es una matriz

Una matriz es una variable que contiene más de un valor. Para indicar uno de los valores, se usa un número de índice.

Declaración de una matriz

La declaración de una matriz se parece a la de cualquier otra variable, si bien la longitud de la matriz se especifica con { }.

```
VAR num my_array{3};
```

Asignación de valores

Es posible asignar a una matriz todos sus valores de una vez. Al asignar la totalidad de la matriz, los valores se encierran entre [ ] y se separan con comas.

```
my_array := [5, 10, 7];
```

También es posible asignar un valor a uno de los elementos de una matriz. El elemento al que se asigna el valor se especifica entre { }.

```
my_array{3} := 8;
```

Ejemplo

En este ejemplo se usan un bucle `FOR` y matrices para preguntar al operador el tiempo de producción estimado de cada pieza. Se trata de una forma muy eficiente de escribir código, en lugar de tener una variable para cada pieza y no poder usar el bucle `FOR`.

```
VAR num time{3};
VAR string part{3} := ["Shaft", "Pipe", "Cylinder"];
VAR num answer;

PROC main()
  FOR i FROM 1 TO 3 DO
     TPReadNum answer, "Estimated time for " + part{i} + "?";
     time{i} := answer;
  ENDFOR
ENDPROC
```

4.2. Tipos de datos compuestos

Qué es un tipo de dato compuesto

Un tipo de dato compuesto es un tipo de dato que contiene más de un valor. Se declara como una variable normal, pero contiene un número predefinido de valores.

pos

Un ejemplo sencillo de tipo de dato compuesto es del tipo de dato `pos`. Contiene tres valores numéricos (X, Y y Z).

La declaración tiene el aspecto de una variable simple:

```
VAR pos pos1;
```

La asignación de todos los valores se realiza como con las matrices:

```
pos1 := [600, 100, 800];
```

Los distintos componentes tienen nombres en lugar de números. Los componentes de `pos` reciben los nombres `x`, `y` y `z`. El valor de un componente se identifica con el nombre de la variable, un punto y el nombre del componente:

```
pos1.z := 850;
```

orient

El tipo de dato `orient` especifica la orientación de la herramienta. La orientación se especifica con cuatro valores numéricos, con los nombres `q1`, `q2`, `q3` y `q4`.

```
VAR orient orient1 := [1, 0, 0, 0];
TPWrite "The value of q1 is " \Num:=orient1.q1;
```

pose

Un tipo de dato puede componerse de otros tipos de datos compuestos. Un ejemplo de ello es el tipo de dato `pose`, que se compone de un dato `pos` con el nombre `trans` y un dato `orient` con el nombre `rot`.

```
VAR pose pose1 := [[600, 100, 800], [1, 0, 0, 0]];
VAR pos pos1 := [650, 100, 850];
VAR orient orient1;
pose1.pos := pos1;
orient1 := pose1.rot;
pose1.pos.z := 875;
```

robtarget

`robtarget` es un tipo de dato demasiado complejo como para explicarlo en detalle aquí, de forma que nos conformaremos con una explicación breve.

`robtarget` se compone de cuatro partes:

Tipo de dato	Nombre	Descripción
pos	trans	Coordenadas X, Y y Z
orient	rot	Orientación
confdata	robconf	Especifica ángulos de ejes de robot.
extjoint	extax	Especifica posiciones de hasta 6 ejes adicionales. Se usa el valor 9E9 si no hay ningún eje adicional.

Continúa en la página siguiente

Continuación

```
VAR robtarget p10 := [ [600, -100, 800], [0.707170, 0, 0.707170,
     0], [0, 0, 0, 0], [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9] ];

! Increase the x coordinate with 50
p10.trans.x := p10.trans.x + 50;
```

Descripciones detalladas

Encontrará descripciones detalladas de estos tipos de datos y muchos más en el *Manual de referencia técnica - RAPID Instrucciones, funciones y tipos de datos*, sección *Tipos de datos*.

4.2. Tipos de datos compuestos

5 Instrucciones y funciones de RAPID

5.1. Instrucciones

Qué es una instrucción

Una instrucción de RAPID actúa como un procedimiento prefabricado. La llamada a una instrucción se parece a la llamada del procedimiento, al usar el nombre de la instrucción seguido por los valores de los argumentos.

Algunas instrucciones de RAPID son sencillas y podrían haberse escrito fácilmente como un procedimiento en RAPID. Por ejemplo la instrucción `Add`.

```
Add reg1, 3;
! The same functionality could be written:
reg1 := reg1 + 3;
```

Otras instrucciones de RAPID realizan procesos complicados que no podrían haberse programado sin estas instrucciones prefabricadas. Por ejemplo `MoveL`, que puede parecer una instrucción sencilla pero que encierra cálculos de cuánto debe moverse cada eje de robot y cuánta intensidad debe recibir cada motor. Dado que el código de programa de estos cálculos ya se suministra hecho, lo único que usted tiene que hacer es escribir una sencilla llamada a una instrucción.

```
MoveL p10, v1000, fine, tool0;
```

Descripciones detalladas

Encontrará descripciones detalladas de las instrucciones en el *Manual de referencia técnica - RAPID Instrucciones, funciones y tipos de datos*, sección *Instrucciones*.

5.2. Funciones

Qué es una función

Una función de RAPID es similar a una instrucción pero devuelve un valor.

```
! Calculate the cosine of reg2
reg1 : = Cos(reg2);
```

Dado que la función devuelve un valor, el resultado de la función puede asignarse a una variable.

Los argumentos de la llamada a una función se escriben entre paréntesis y se separan con comas.

Inclusión de una llamada a función en una sentencia

En todos los lugares en los que puede usarse un valor, también puede usarse una función que devuelva un valor del mismo tipo de dato.

```
! Perform something if reg1 is smaller than -2 or greater than 2
IF Abs(reg1) > 2 THEN
...

! Convert the num time to string and concatenate with other strings
string1 := name + "'s time was " + NumToStr(time);
```

Simplificación de cálculos complejos

Con frecuencia, una sola llamada a una función puede sustituir a varias sentencias complejas.

Por ejemplo:

```
p20 := Offs(p10, 100, 200, 300);
```

Puede reemplazar al código siguiente:

```
p20 := p10;
p20.trans.x := p20.trans.x + 100;
p20.trans.y := p20.trans.y + 200;
p20.trans.z := p20.trans.z + 300;
```

Descripciones detalladas

Encontrará descripciones detalladas de las funciones en el *Manual de referencia técnica - RAPID Instrucciones, funciones y tipos de datos*, sección *Funciones*.

6 Qué debe leer a continuación

6.1. Dónde buscar más información

Qué encontrará en cada manual

Lo que desea saber	Dónde obtener información
• Cómo escribir programas en el FlexPendant • Cómo cargar programas en el controlador de robot • Cómo probar el programa	*Manual del operador - IRC5 con FlexPendant*, sección *Programación y testing*
• Información más detallada acerca de la funcionalidad mencionada en este manual • Qué instrucciones existen para una categoría determinada (por ejemplo instrucciones de movimiento o funciones de reloj) • Descripciones de funciones más avanzadas (por ejemplo interrupciones o gestión de errores)	*Manual de referencia técnica - Descripción general de RAPID*
• Información acerca de una instrucción, una función o un tipo de dato en concreto	*Manual de referencia técnica - RAPID Instrucciones, funciones y tipos de datos*
• Detalles acerca de cómo el controlador de robot trata las distintas partes de RAPID	*Technical reference manual - RAPID Kernel*

6.1. Dónde buscar más información

3HAC029364-005, Revisión -, es

ABB Robotics
S-721 68 VÄSTERÅS
SWEDEN
Telephone: +46 (0) 21 344000
Telefax: +46 (0) 21 132592