

EPSON RC+ 7.0 Option

*RC+ API 7.0*

Rev.21

ENM231S5556F

Original instructions

EPSON RC+ 7.0 Option RC+ API 7.0 Rev.21

EPSON RC+ 7.0 Option

# *RC+ API 7.0*

Rev.21

©Seiko Epson Corporation 2012-2023

# FOREWORD

Thank you for purchasing our robot products.

This manual contains the information necessary for the correct use of the Manipulator.

Please carefully read this manual and other related manuals before installing the robot system.

Keep this manual handy for easy access at all times.

The robot system and its optional parts are shipped to our customers only after being subjected to the strictest quality controls, tests, and inspections to certify its compliance with our high performance standards. Please note that the basic performance of the product will not be exhibited if our robot system is used outside of the usage conditions and product specifications described in the manuals.

This manual describes possible dangers and consequences that we can foresee. Be sure to comply with safety precautions on this manual to use our robot system safely and correctly.

# TRADEMARKS

Microsoft, Windows, and Windows logo are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other brand and product names are trademarks or registered trademarks of the respective holders.

# TRADEMARK NOTATION IN THIS MANUAL

Microsoft® Windows® 8 operating system

Microsoft® Windows® 10 operating system

Microsoft® Windows® 11 operating system

Throughout this manual, Windows 8, Windows 10 and Windows 11 refer to above respective operating systems. In some cases, Windows refers generically to Windows 8, Windows 10 and Windows 11.

# NOTICE

No part of this manual may be copied or reproduced without authorization.

The contents of this manual are subject to change without notice.

Please notify us if you should find any errors in this manual or if you have any comments regarding its contents.

# MANUFACTURER

**SEIKO EPSON CORPORATION**

# CONTACT INFORMATION

Contact information is described in “SUPPLIERS” in the first pages of the following manual:

*Robot System Safety Manual Read this manual first*

---

<b>1. Introduction</b>	<b>1</b>
1.1 Features .....	1
<b>2. Installation</b>	<b>2</b>
2.1 Step by step instructions.....	2
2.2 What's installed .....	3
<b>3. Getting Started</b>	<b>4</b>
3.1 Getting started using Visual Basic .....	4
3.2 Getting started using Visual C# .....	5
3.3 Getting started using Visual C++ .....	6
3.4 Countermeasure for build failure (MSB8036) on Visual C++ 2017 .....	9
<b>4. Environments</b>	<b>11</b>
4.1 Development Environment .....	11
4.1.1 Development Startup .....	11
4.1.2 Spel Class Instance Initialization .....	11
4.1.3 Spel Class Instance Termination.....	11
4.1.4 Development Cycle .....	11
4.2 In Production Facilities .....	12
4.2.1 Opening EPSON RC+ 7.0 at Runtime.....	12
4.2.2 Using EPSON RC+ 7.0 Dialogs and Windows.....	12
4.2.3 Installation on Target System.....	12
<b>5. Executing Methods, Programs, Tasks</b>	<b>13</b>
5.1 Executing Methods .....	13
5.1.1 Using Multiple Threads .....	13
5.2 Executing SPEL+ Programs .....	18
5.3 Executing SPEL+ Tasks.....	18
5.4 Aborting All Tasks.....	19
<b>6. Events</b>	<b>20</b>
6.1 Overview .....	20
6.2 System Events .....	20
6.3 User Events from SPEL+ .....	20

<b>7. Error Handling</b>	<b>22</b>
7.1 Errors for Spel methods.....	22
<b>8. Handling Pause and Continue</b>	<b>24</b>
8.1 Pause state.....	24
8.2 Catching the Pause event.....	24
8.3 Executing Pause.....	25
8.4 Continue after pause.....	26
8.5 Abort after pause.....	26
<b>9. Handling Emergency Stop</b>	<b>27</b>
9.1 Using system EStop events.....	27
<b>10. EPSON RC+ 7.0 Windows and Dialogs</b>	<b>28</b>
10.1 Windows.....	28
10.2 Dialogs.....	29
<b>11. Displaying Video</b>	<b>30</b>
Using multiple video displays.....	31
<b>12. Using AsyncMode</b>	<b>33</b>
<b>13. SPELCom_Event</b>	<b>35</b>
<b>14. RCAPINet Reference</b>	<b>36</b>
14.1 Spel Class.....	36
14.2 Spel Class Properties.....	36
14.3 Spel Class Methods.....	65
14.4 Spel Class Events.....	364
14.5 SPELVideo Control.....	368
14.6 SPELVideo Control Properties.....	368
14.7 SPELVideo Control Methods.....	371
14.8 SPELVideo Control Events.....	372
14.9 SpelConnectionInfo Class.....	372
14.10 SpelControllerInfo Class.....	372
14.11 SpelException Class.....	373
14.12 SpelOptionInfo Class.....	374

---

14.13 SpelPoint Class.....	374
14.13.1 SpelPoint Properties .....	376
14.13.2 SpelPoint Methods.....	377
14.14 SpelRobotInfo Class .....	378
14.15 SpelTaskInfo Class .....	378
14.16 Enumerations.....	379
14.16.1 SpelArmDefMode Enumeration .....	379
14.16.2 SpelArmDefType Enumeration .....	379
14.16.3 SpelAxis Enumeration.....	379
14.16.4 SpelBaseAlignment Enumeration .....	379
14.16.5 SpelCalPlateType Enumeration.....	379
14.16.6 SpelConnectionType Enumeration .....	379
14.16.7 SpelDialogs Enumeration .....	380
14.16.8 SpelElbow Enumeration.....	380
14.16.9 SpelEvents Enumeration .....	380
14.16.10 SpelForceAxis Enumeration .....	381
14.16.11 SpelForceCompareType Enumeration.....	381
14.16.12 SpelForceProps Enumeration.....	381
14.16.13 SpelHand Enumeration.....	383
14.16.14 SpelIOLabelTypes Enumeration .....	383
14.16.15 SpelLocalDefType Enumeration .....	383
14.16.16 SpelOperationMode Enumeration .....	383
14.16.17 SpelOptions Enumeration.....	384
14.16.18 SpelOptionStatus Enumeration .....	384
14.16.19 SpelRobotPosType Enumeration.....	384
14.16.20 SpelRobotType Enumeration.....	384
14.16.21 SpelShutdownMode Enumeration .....	384
14.16.22 SpelSimObjectType Enumeration.....	385
14.16.23 SpelSimProps Enumeration.....	385
14.16.24 SpelStopType Enumeration .....	385
14.16.25 SpelTaskState Enumeration .....	386
14.16.26 SpelTaskType Enumeration .....	386
14.16.27 SpelToolDefType Enumeration .....	386
14.16.28 SpelToolDefType3D Enumeration.....	386
14.16.29 SpelUserRights Enumeration .....	387
14.16.30 SpelVDefShowWarning Enumeration.....	387
14.16.31 SpelVisionImageSize Enumeration .....	388
14.16.32 SpelVisionObjectTypes Enumeration .....	388
14.16.33 SpelVisionProps Enumeration .....	389
14.16.34 SpelWindows Enumeration.....	389
14.16.35 SpelWrist Enumeration .....	389
14.17 Spel Error Numbers and Messages .....	389

<b>15. 32 Bit and 64 Bit Applications</b>	<b>390</b>
<b>16. Using the LabVIEW VI Library</b>	<b>391</b>
16.1 Overview .....	391
16.2 Installation.....	391
16.3 Tool and Control Palettes .....	392
16.4 Getting started .....	394
16.5 Working with Spel+ projects .....	395
16.6 Displaying Video .....	396
16.7 VI Reference.....	397
<b>17. Using LabVIEW with RCNetLib</b>	<b>506</b>
17.1 Overview .....	506
17.2 Initialization .....	506
17.2.1 Add a constructor node for the Spel class.....	506
17.2.2 Initialize the Spel class instance.....	507
17.2.3 Connect to Controller and set project.....	507
17.3 Use Spel properties and methods .....	507
17.4 Shutdown .....	507
17.5 Using Dialogs and Windows.....	507
<b>18. How to Control Multiple Controllers from One PC</b>	<b>508</b>
18.1 Overview .....	508
18.1.1 System Condition.....	508
18.1.2 Connection of PC and Controllers .....	509
18.2 Restrictions on controlling multiple Controllers .....	510
18.2.1 Restrictions on Controller options.....	510
18.2.2 Restrictions on simulator .....	510
18.3 Sample Program for connecting multiple Controllers.....	510
18.3.1 Controller connection setting .....	510
18.3.2 Project setting .....	511
18.3.3 Sample program using Visual Basic.....	511
18.3.4 Sample program using Visual C#.....	513



# 1. Introduction

The EPSON RC+ 7.0 Option RC+ API enables you to use Microsoft Visual Basic or any other language that supports .NET technology to run your robotic applications. This gives you the power to create sophisticated user interfaces, use databases, and use third party products designed for use with .NET.

A LabVIEW library is also included.


## 1.1 Features

The following features are supported in the RC+ API package:

- A .NET library and LabVIEW library.
- Supports 32 bit and 64 bit applications.
- Properties and methods for controlling multiple robots, I/O, and tasks from multiple controllers.
- Methods for executing vision and force sensing\* commands.
  - \* Force sensing and Force Sensor are different.  
Methods and properties for force sensing described in API manual are not available for Force Sensor. To use commands for Force Sensor, use Xpt method to execute SPEL function.  
API does not support EPSON RC+ option Force Guide.
- Supports parallel execution of asynchronous commands by multi-threading.
- Several EPSON RC+ 7.0 windows and dialogs can be used by your .NET application, including:
  - Robot Manager
  - IO monitor
  - Task manager
  - Simulator
  - Controller Tools dialog

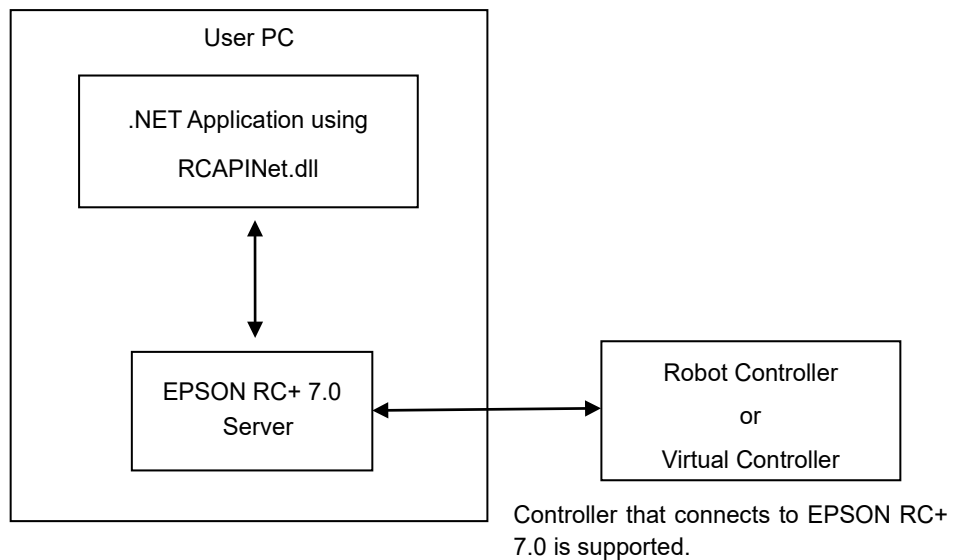
During development, EPSON RC+ 7.0 can be run along with Visual Basic.

In production facilities, EPSON RC+ 7.0 can be run invisibly in the background.

 **NOTE** RC+ API supports “.NET library” and LabVIEW library. “.NET Core”, or “.NET” 5 or later is not supported.

## 2. Installation

The figure below shows the basic structure of a system using the RC+ API.



RC+ API Basic Structure for the .NET library

EPSON RC+ 7.0 is an out-of-process server for the RCAPINet library.

Each instance of RCAPINet Spel class can start an instance of EPSON RC+ 7.0.

## 2. Installation

Please follow the instructions in this chapter to help ensure proper installation of the RC+ API software.

Before starting, ensure that all Windows applications have been closed.

### 2.1 Step by step instructions

- (1) Install either one of the following:  
Visual Studio 2012, 2013, 2015, 2017, 2019  
(Includes Enterprise, Professional, Community, and Express Editions)  
LabVIEW 2009 or later
- (2) Install EPSON RC+ 7.0.
- (3) If you are using LabVIEW, install the LabVIEW VI library.
- (4) Ensure that the software key has been enabled for RC+ API in the Controllers you will be using. Refer to the EPSON RC+ 7.0 User's Guide for information on how to enable options in the Controller.

This completes the RC+ API installation.

## 2.2 What's installed

The directories and files shown in the table below are installed on your PC during installation.

Directories and Files	Description
\EPSONRC70\API\VS20xx\VB\DEMOS	Visual Basic .NET demonstrations
\EPSONRC70\API\VS20xx\VCS\DEMOS	Visual C# .NET demonstrations
\EPSONRC70\API\VS20xx\VC\DEMOS	Visual C++ .NET demonstrations
\EPSONRC70\API\LabVIEW	LabVIEW VI Library installer
\EPSONRC70\PROJECTS\API_Demos	EPSON RC+ 7.0 projects for demos
\EPSONRC70\EXE\RCAPINet.dll	RCAPINet Class library (32 bit or 64 bit)

# 3. Getting Started

This chapter contains information for getting started in the following development environments.

- Visual Basic .NET
- Visual C# .NET
- Visual C++ .NET

Demonstration programs are supplied with the RC+ API. It is recommended that you go through the demonstrations to get more familiar with the product.

LabVIEW users should now refer to chapter 16. *Using the LabVIEW VI Library* for instructions on getting started and using the library.

When you build the demonstration program on Visual C++ 2017 for the first time, the program build may fail. When program build fails, refer to the following section:

### 3.4 Countermeasure for build failure (MSB8036) on Visual C++ 2017

To use .NET application with EPSON RC+ 7.0 version 7.5.0 or later, set .NET Framework version to v4.5 and greater.

## 3.1 Getting started using Visual Basic

To use RCAPINet in a Visual Basic .NET project, declare a Spel Class instance, as shown in the example below. `g_spel` can now be used in your project.

1. In Visual Studio .NET, select File | Project.
2. Create a Visual Basic project as Windows Forms Application.
3. From the Project menu, select Add Reference.
4. In the NET Components tab, browse to the `\EpsonRC70\Exe` directory and select the `RCAPINet.dll` file.
5. From the Project menu, create a new module and add the following code.

```
Module Module1
    Public WithEvents g_spel As RCAPINet.Spel
    Public Sub InitApp()
        g_spel = New RCAPINet.Spel
        With g_spel
            .Initialize
            .Project = "c:\EpsonRC70\projects\API_Demos\Demo1
\demo1.sprj"
        End With
    End Sub

    Public Sub EventReceived( _
        ByVal sender As Object, _
        ByVal e As RCAPINet.SpelEventArgs) _
        Handles g_spel.EventReceived

        MsgBox("received event " & e.Event)
    End Sub
End Module
```



When your application exits, you need to execute Dispose for each Spel class instance. This can be done in your main form's FormClosed event. If Dispose is not executed, the application will not shutdown properly.

```
g_spel.Dispose()
```

## 3.2 Getting started using Visual C#

1. In Visual Studio .NET, select File | Project.
2. Create a Visual C# project as Windows Forms Application.
3. From the Project menu, select Add Reference.
4. Select the Browse tab and browse to the \EpsonRC70\Exe directory and select the RCAPINet.dll file.

5. In the Form1 class, declare a Spel class variable as shown below.

```
private RCAPINet.Spel m_spel;
```

6. In the Form\_Load event, add initialization code, as shown below.

```
private void Form1_Load(object sender, EventArgs e)
{
    m_spel = new RCAPINet.Spel();
    m_spel.Initialize();

    m_spel.Project =

    "c:\\EpsonRC70\\projects\\API_Demos\\Demo1\\demo1.sprj";

    m_spel.EventReceived += new
        RCAPINet.Spel.EventReceivedEventHandler(m_spel_
        EventReceived);
}
```

7. Add the event handler, as shown below.

```
public void m_spel_EventReceived(object sender,
    RCAPINet.SpelEventArgs e)
{
}
}
```

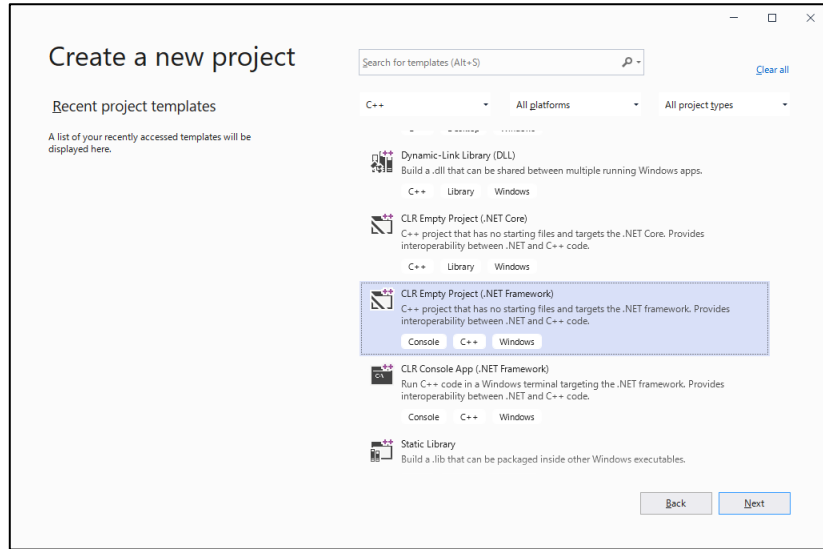


When your application exits, you need to execute Dispose for each Spel class instance. This can be done in your main form's FormClosed event. If Dispose is not executed, the application will not shutdown properly.

```
m_spel.Dispose();
```

### 3.3 Getting started using Visual C++

1. In Visual Studio .NET, select [Create a new project].
2. Select [Visual C++]-[CLR]-[CLR Empty Project (.NET Framework)].



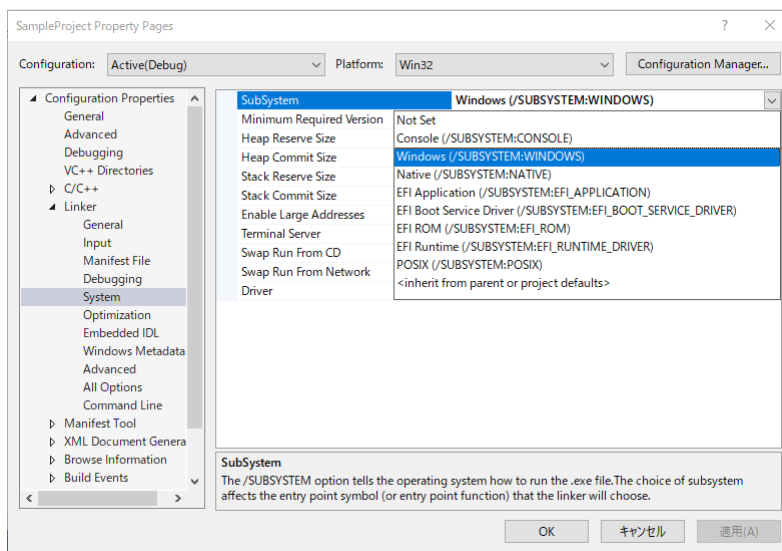
3. Select the menu-[Project]-[Add Reference].
4. Click the <Browse> button to browse “/EpsonRC70/Exe” directory and select “RCAPINet.dll” file.
5. Select the menu-[Project]-[Add New Item]-[UI]-[Windows Form].
6. Open the cpp file (ex: Form1.cpp) of the added form and add the following source code.

```
#include "Form1.h"
using namespace SampleProject; ← Created project name
void main() {
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);

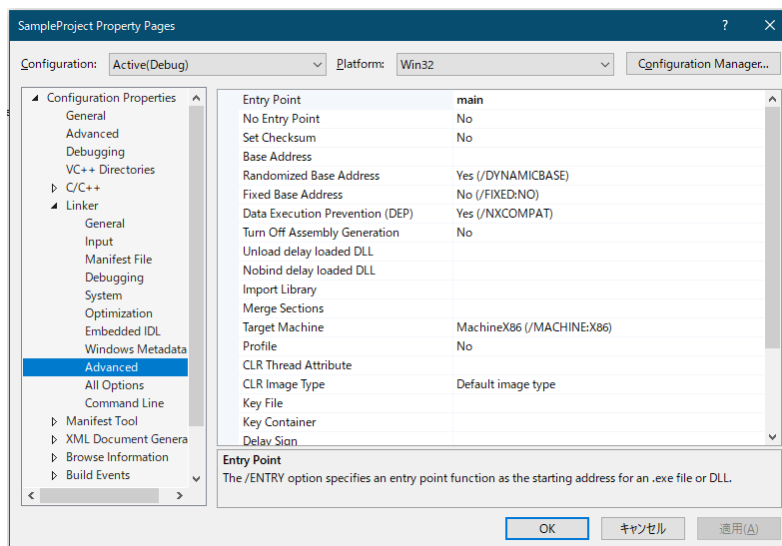
    Form1 frm; ← Added form name
    Application::Run(% frm);
}
```

7. Select the menu-[Project]-[Project Properties].

8. Select [Configuration Properties]-[Linker]-[System] on Property Pages and select “Windows (/SUBSYSTEM:WINDOWS)” on SubSystem.



9. Select [Configuration Properties]-[Linker]-[Advanced] on Property Pages and enter the function name that is added in the step 6 in “Entry Point”. In here, “main” is entered.



10. Click the <OK> button.



After configuring the setting, build the solution once and make sure that no error is occurred. Then, we recommend that you close the solution and reopen it.

11. In the Form1 class, declare a Spel variable as shown below.

```
private RCAPINet::Spel^ m_spel;
```

12. In the Form\_Load event, add initialization code, as shown below.

```
private: System::Void Form1_Load(
    System::Object^ sender, System::EventArgs^ e)
{
    m_spel = gcnew RCAPINet::Spel();
    m_spel->Initialize();
    m_spel->Project =
        "c:\\EpsonRC70\\projects\\ API_Demos\\Demo1\\demo1.sprj";
    m_spel->EventReceived += gcnew
        RCAPINet::Spel::EventReceivedEventHandler(
            this, &Form1::m_spel_EventReceived);
}
```

13. Add the event handler, as shown below.

```
private System::Void m_spel_EventReceived(
    System::Object^ sender, RCAPINet::SpelEventArgs^ e)
{
    MessageBox::Show(e->Message);
}
```



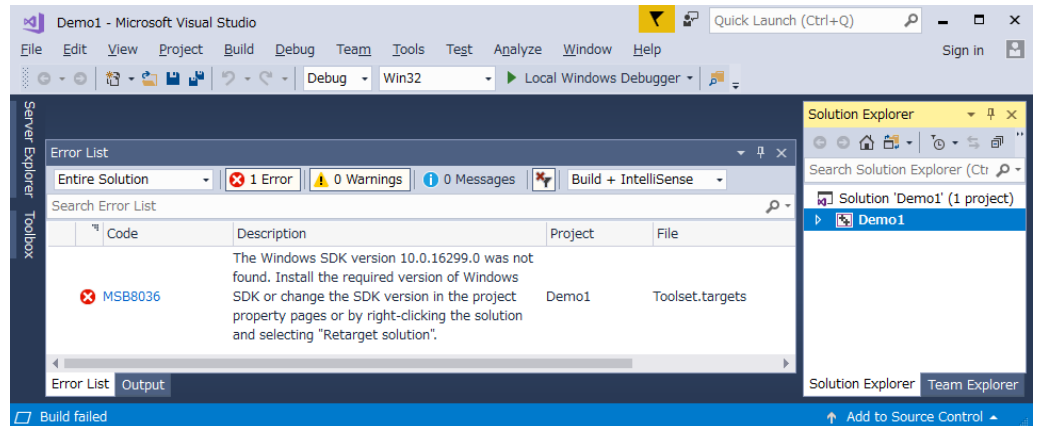
NOTE When your application exits, you need to delete each Spel class instance if it was allocated on the heap (using gcnew). This can be done in your main form's FormClosed event. If the Spel class instances are not deleted, then the application will not shutdown properly.

```
delete m_spel;
```

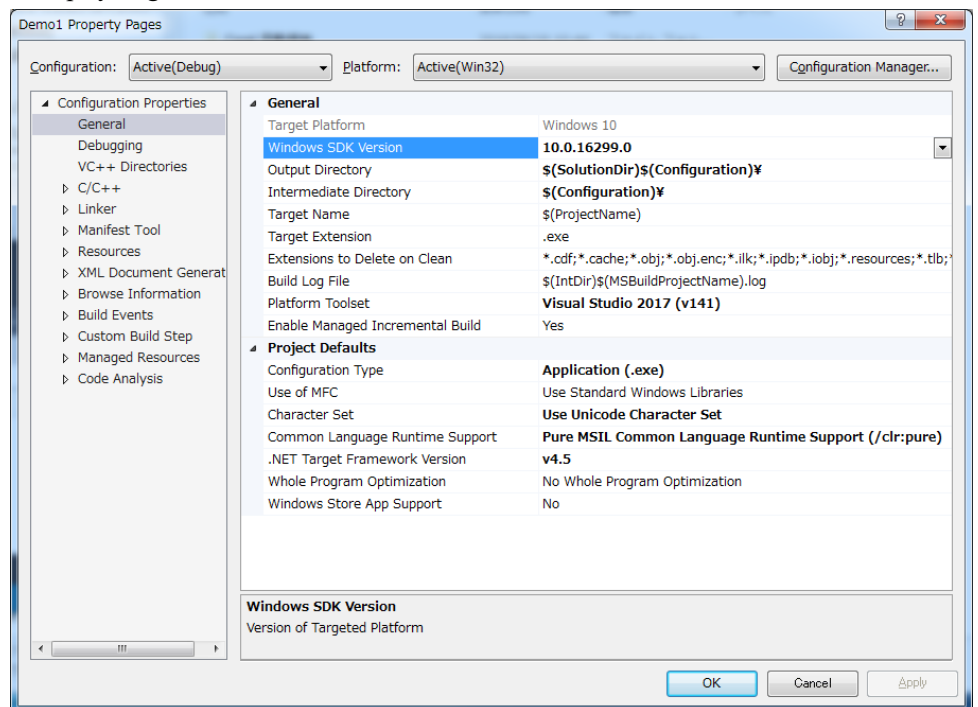


### 3.4 Countermeasure for build failure (MSB8036) on Visual C++ 2017

When you build the demonstration program on Visual C++ 2017 for the first time, and if the build fails due to an error: MSB8036, follow the procedures below:



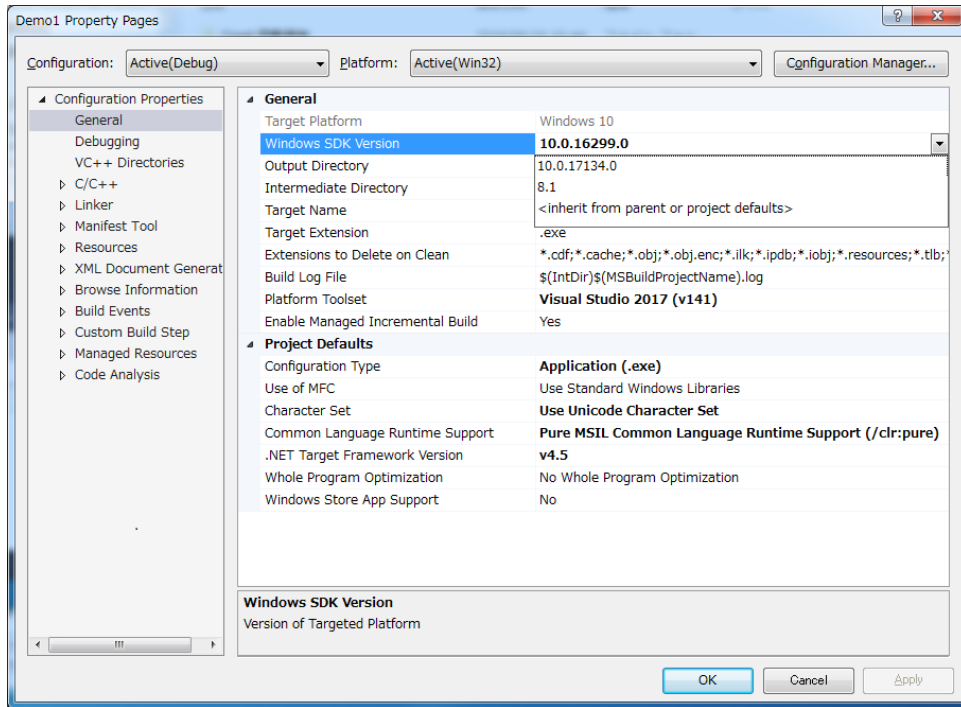
1. Select Visual Studio C++ 2017-Solution Explorer-“Demo1” project.
2. Select Menu-[Project]-[Properties].
3. Select [Configuration Properties]-[General]-[Windows SDK Version] on “Demo1 Property Pages”.



4. Click the pulldown button which is on the right side of “10.0.16299.0”.

### 3. Getting Started

5. Select “Windows SDK Version” which is installed in the developmnet environment.



6. Click the <OK> button.
7. Re-build the demonstration program.

## 4. Environments

### 4.1 Development Environment

#### 4.1.1 Development Startup

Typically, you would perform these steps to start development:

1. Declare a Spel class variable in a module in your .NET project.
2. Start EPSON RC+ 7.0.
3. Open the desired EPSON RC+ 7.0 project or create a new EPSON RC+ 7.0 project.
4. Build the EPSON RC+ 7.0 project.
5. Add initialization code for the SPEL class instance.
6. Run and debug the .NET project.

#### 4.1.2 Spel Class Instance Initialization

After a new instance of the Spel class has been created, it needs to be initialized. When initialization occurs, the underlying EPSON RC+ 7.0 modules are loaded and initialized. Initialization is implicit with the first method call or property access. You can initialize the class by calling the Initialize method.

```
m_spel.Initialize()
```

#### 4.1.3 Spel Class Instance Termination

When your application exits, you need to execute Dispose for each Spel class instance. This can be done in your main form's FormClosed event. If Dispose is not executed, the application will not shutdown properly.

For Visual Basic and Visual C#, use the Dispose method:

```
m_spel.Dispose()
```

For Visual C++, if your Spel class instance was created on the heap (with gcnew), then use delete:

```
delete m_spel;
```

#### 4.1.4 Development Cycle

Follow these basic steps to edit and run your .NET code:

1. Stop the .NET project.
2. Edit the .NET project
3. Open EPSON RC+ 7.0.
4. Make changes in the EPSON RC+ 7.0 project.
5. Build the EPSON RC+ 7.0 project.
6. Close the RC+ 7.0.
7. Switch to Visual Studio.
8. Run the .NET project.

### 4.2 In Production Facilities

#### 4.2.1 Opening EPSON RC+ 7.0 at Runtime

Decide if you want to allow the EPSON RC+ 7.0 environment to be opened from your application. This is especially useful for debugging. Set the `OperationMode` property to `Program` to put EPSON RC+ 7.0 in Program Mode and open the EPSON RC+ 7.0 GUI.

#### 4.2.2 Using EPSON RC+ 7.0 Dialogs and Windows

At runtime, you can open and hide certain EPSON RC+ 7.0 windows from your .NET application. You can also run certain EPSON RC+ 7.0 dialogs. See the chapter *EPSON RC+ 7.0 Windows and Dialogs* for details.

#### 4.2.3 Installation on Target System

You should make an installation program for your .NET project by using a Visual Studio setup project. Then follow these steps to setup a target system for your .NET application:

1. Install EPSON RC+ 7.0.
2. Install your EPSON RC+ 7.0 project.
3. Install your .NET application.

## 5. Executing Methods, Programs, Tasks

### 5.1 Executing Methods

There are several methods in the Spel class. For descriptions of available methods, see the section *14.3 Spel Class Methods*. When you execute a method, the associated internal functions are called in the EPSON RC+ server process, which in turn communicates with the Controller to execute the associated function. There are two types of methods: immediate and asynchronous. For immediate methods, the internal function is executed in the Controller and the reply is returned immediately. Immediate commands include all I/O commands. For asynchronous methods, the associated function is started in the Controller, and then the Spel class instance waits for an event from the EPSON RC+ server process indicating that the function has completed. Asynchronous methods include all robot motion commands. While waiting for command completion, the Spel class instance dispatches Windows events, so that the user GUI is still responsive. For example, when the Go method is called, the robot is moving to a point, and the user may want to stop it by clicking a button. You can disable Windows event dispatching during asynchronous methods by setting `DisableMsgDispatch` to `True`. You can also wait for asynchronous methods to finish in your program by setting `AsyncMode` to `True`.

#### 5.1.1 Using Multiple Threads

You can execute Spel methods in multiple threads in your application. The sections below describe the various scenarios.

##### One Spel class instance used in multiple threads

You can execute methods with the same Spel class instance in multiple threads, but only one asynchronous command at a time. If you attempt to execute an asynchronous command in one thread while another asynchronous command is already executing in another thread, you will get a “command in cycle” error. You can execute an immediate command in one thread while executing an asynchronous command in another thread.

##### Separate Spel class instance used in each thread

For each Controller connection, you can have one or more Spel class instances. The first instance for each Controller initializes an EPSON RC+ 7.0 server process and connects to the specified Controller. To use one or more additional instances in other threads to communicate with the same Controller, you must specify the `ServerInstance` property to be the same value. You call `Initialize` for the first instance before using additional Spel class instances.

### VB Example:

```
' Initialize Spel class instance for thread 1
m_spel_1 = New Spel
m_spel_1.ServerInstance = 1
m_spel_1.Initialize()
m_spel_1.Project =
"c:\EpsonRC70\Projects\MyProject\MyProject.sprj"
m_spel_1.Connect(1)
```

```
' Initialize Spel class instance for thread 2
' This instance uses the same controller as m_spel_1
m_spel_2 = New Spel
m_spel_2.ServerInstance = 1
```

#### Thread 1

```
' Uses instance m_spel_1 for motion
m_spel_1.Robot = 1
Do
  m_spel_1.Go(1)
  m_spel_1.Go(2)
Loop Until m_stop
```

#### Thread 2

```
' Uses instance m_spel_2 for I/O
Do
  m_spel_2.On(1)
  m_spel_2.Delay(500)
  m_spel_2.Off(1)
  m_spel_2.Delay(500)
Loop Until m_stop
```

**C# Example:**

```
// Initialize Spel class instance for thread 1
RCAPINet.Spel m_spel_1 = new RCAPINet.Spel();
m_spel_1.ServerInstance = 1;
m_spel_1.Initialize();
m_spel_1.Project =
@"c:\EpsonRC70\Projects\MyProject\MyProject.sprj";
m_spel_1.Connect(1);
```

```
// Initialize Spel class instance for thread 2
// This instance uses the same controller as m_spel_1
RCAPINet.Spel m_spel_2 = new RCAPINet.Spel();
m_spel_2.ServerInstance = 1;
```

**Thread 1**

```
// Uses instance m_spel_1 for motion
m_spel_1.Robot = 1;
do{
    m_spel_1.Go(1);
    m_spel_1.Go(2);
}while(!m_stop);
```

**Thread 2**

```
// Uses instance m_spel_2 for I/O
do{
    m_spel_2.On(1);
    m_spel_2.Delay(500);
    m_spel_2.Off(1);
    m_spel_2.Delay(500);
}while(!m_stop);
```

**Using API threads in the Controller**

By default, only one API thread is supported in the Controller. In this case, asynchronous methods are executed one at a time in the Controller, even when controlling multiple robots. For most applications that use one robot, or execute robot motion using SPEL+ tasks, this is sufficient, but you can configure the system to use up to 10 API tasks in the Controller to allow parallel processing for your .NET threads, such as when you are controlling more than one robot from the same Controller.

There are two basic steps required to use more than one API task in the Controller.

1. In the EPSON RC+ GUI, connect to the Controller, then open [Setup]-[System Configuration]-[Controller]-[Preferences]. Set "Reserved tasks for API" to the desired number of API tasks. Note that the more tasks you reserve for the API, the fewer tasks will be available for your SPEL+ programs. For example, if you reserve 5 API tasks, then there will be 27 tasks (32 – 5) available for SPEL+.
2. In your application, set the CommandTask property to specify which API task you want to execute methods on.

In the simple example below, there is one thread for each robot in the same Controller. The robot motion commands will execute in parallel, since a different CommandTask is used in each thread, and ServerInstance is set to 1 for both Spel instances.

**VB Example:**

```
' Initialize Spel class instance for thread 1
m_spel_1 = New Spel
m_spel_1.ServerInstance = 1
m_spel_1.CommandTask = 1
m_spel_1.Initialize()
m_spel_1.Project =
"c:\EpsonRC70\Projects\MyProject\MyProject.sprj"
m_spel_1.Connect(1)
```

```
' Initialize Spel class instance for thread 2
' This instance uses the same controller as m_spel_1
' And uses the second CommandTask in the controller.
m_spel_2 = New Spel
m_spel_2.ServerInstance = 1
m_spel_2.CommandTask = 2
```

**Thread 1**

```
' Uses instance m_spel_1 for Robot 1 motion
m_spel_1.Robot = 1
Do
  m_spel_1.Go(1)
  m_spel_1.Go(2)
Loop Until m_stop
```

**Thread 2**

```
' Uses instance m_spel_2 for Robot 2 motion
m_spel_2.Robot = 2
Do
  m_spel_2.Go(1)
  m_spel_2.Go(2)
Loop Until m_stop
```



**C# Example:**

```
// Initialize Spel class instance for thread 1
RCAPINet.Spel m_spel_1 = new RCAPINet.Spel();
m_spel_1.ServerInstance = 1;
m_spel_1.CommandTask = 1;
m_spel_1.Initialize();
m_spel_1.Project =
@"c:\EpsonRC70\Projects\MyProject\MyProject.sprj";
m_spel_1.Connect(1);
```

```
// Initialize Spel class instance for thread 2
// This instance uses the same controller as m_spel_1
// And uses the second CommandTask in the controller.
RCAPINet.Spel m_spel_2 = new RCAPINet.Spel();
m_spel_2.ServerInstance = 1;
m_spel_2.CommandTask = 2;
```

**Thread 1**

```
// Uses instance m_spel_1 for Robot 1 motion
m_spel_1.Robot = 1;
do{
    m_spel_1.Go(1);
    m_spel_1.Go(2);
}while(!m_stop);
```

**Thread 2**

```
// Uses instance m_spel_2 for Robot 2 motion
m_spel_2.Robot = 2;
do{
    m_spel_2.Go(1);
    m_spel_2.Go(2);
}while(!m_stop);
```

## 5.2 Executing SPEL+ Programs

A SPEL+ program contains one or more functions, and the program is run by starting the main function of the program. You can run any of the 64 built-in main functions in the current Controller project by using the *Start* method of the *Spel* class. The main function(s) that you start must be defined in your SPEL+ code. When you start a main function, all global variables and module variables are cleared to default values.

The table below shows the program numbers and their corresponding function names in the SPEL+ project.

Program Number	SPEL+ Function Name
0	main
1	main1
2	main2
3	main3
...	...
63	main63

Here are examples that start function “main”:

### VB Example:

```
Sub btnStart_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnStart.Click

    m_spel.Start(0) ' Starts function main
    btnStart.Enabled = False
    btnStop.Enabled = True
End Sub
```

### C# Example:

```
void btnStart_Click(object sender, EventArgs e)
{
    m_spel.Start(0); //Starts function main
    btnStart.Enabled = false;
    btnStop.Enabled = true;
}
```

## 5.3 Executing SPEL+ Tasks

You can execute functions in your SPEL+ program as a normal task by using the *Xqt* method. When you execute a task, global variables are not cleared to default values, as they are when you use the *Start* method.

To suspend and resume a task, use the *Halt* and *Resume* methods.

To quit a task, use the *Quit* method.

You can also start Controller background tasks using the *StartBGTask* method.

## 5.4 Aborting All Tasks

If you are running tasks and want to abort all tasks at once, you can use the *Stop* method of the *Spel* class. The *Stop* method has an optional parameter that allows you to additionally stop all background tasks.

### VB Example:

```
Sub btnStop_Click( _  
    ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) _  
    Handles btnStop.Click  
    m_spel.Stop()  
    btnStop.Enabled = False  
    btnStart.Enabled = True  
End Sub
```

### C# Example:

```
void btnStop_Click(object sender, EventArgs e)  
{  
    m_spel.Stop();  
    btnStop.Enabled = false;  
    btnStart.Enabled = true;  
}
```

## 6. Events

### 6.1 Overview

The Spel Class supports two types of events: system events and user events. System events are notifications of system status. User defined events are sent from any SPEL+ task to the .NET application.

### 6.2 System Events

There are several system events that are sent to the .NET application. Each system event indicates a change in status. There are events for Pause, Continue, Emergency Stop, etc. For complete details on all system events, see the description for *14.4 Spel Class Events - EventReceived*.

Use the Spel class EnableEvents method to control which system events are sent.

### 6.3 User Events from SPEL+

You can cause events to occur in your .NET application from your SPEL+ programs. For example, you can inform the .NET application about a continuous cycle loop. This is a better method to use than polling for variable values in the Controller from .NET.

To fire an event to .NET from SPEL+, use the SPELCom\_Event command in a SPEL+ program statement. For example:

```
SPELCom_Event 1000, cycNum, lotNum, cycTime
```

The SPELCom\_Event command is similar to a Print command. You can specify one or more pieces of data to be sent to the .NET application. See *13. SPELCom\_Event* for details on SPELCom\_Event.

Before you can receive events, you must declare your Spel class variable using the WithEvents clause.

```
Public WithEvents m_spel As RCAPINet.Spel
```

Catch the event in the EventReceived routine for the Spel class instance. To edit this routine, in the module where the Spel class is declared select "m\_spel" from the class name list and EventReceived from the procedure list.

Here is an example of code in the EventReceived routine that updates some labels when an event occurs.

#### VB Example:

```
Sub m_spel_EventReceived (ByVal sender As Object, _
ByVal e As RCAPINet.SpelEventArgs) _
Handles m_spel.EventReceived
    Dim tokens() As String
    Select Case e.Event
        Case 2000
            tokens = e.Message.Split(New [Char]() {" "c}, _
System.StringSplitOptions.RemoveEmptyEntries)
            lblCycCount.Text = tokens(0)
            lblLotNumber.Text = tokens(1)
            lblCycTime.Text = tokens(2)
    End Select
End Sub
```

**C# Example:**

```
void m_spel_EventReceived(object sender, SpelEventArgs e)
{
    string[] tokens = new string[3];
    switch(e.Event)
    {
        case 2000:
            tokens = e.Message.Split(' ');
            lblCycCount.Text = tokens(0);
            lblLotNumber.Text = tokens(1);
            lblCycTime.Text = tokens(2);
            break;
        default:
            break;
    }
}
```

## 7. Error Handling

### 7.1 Errors for Spel methods

When you execute a Spel class method, an exception is thrown if there are any errors.

When an error occurs, the Spel class instance throws it to the calling routine. You should use error handlers in your application to catch this error. In some cases, you will only want to display an error message.

#### VB Example:

```
Sub btnStart_Click( _  
    ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) _  
    Handles btnStart.Click  
  
    Try  
        m_spel.Start(0)  
    Catch ex As RCAPINet.SpelException  
        MsgBox(ex.Message)  
    End Try  
End Sub
```

You can examine the error number associated with the exception by using the `ErrorNumber` property of `SpelException`.

```
Try  
    m_spel.Start(0)  
Catch ex As RCAPINet.SpelException  
    MsgBox("SPEL Error: " + ex.ErrorNumber.ToString())  
End Try
```

**C# Example:**

```
void btnStart_Click(object sender, EventArgs e)
{
    try{
        m_spel.Start(0);
    }
    catch(SpelException ex){
        MessageBox.Show(ex.Message);
    }
}
```

You can examine the error number associated with the exception by using the `LineNumber` property of `SpelException`.

```
try {
    m_spel.Start(0);
}
catch(SpelException ex) {
    MessageBox.Show("SPEL Error: " +
ex.LineNumber.ToString());
}
```

## 8. Handling Pause and Continue

### 8.1 Pause state

When a pause occurs, the Controller and SPEL<sup>+</sup> tasks are in the pause state.

The Controller is in the pause state after one of the following occurs while tasks are running:

- The Spel class Pause method was executed
- A SPEL<sup>+</sup> task executed Pause.
- The safeguard was opened.

### 8.2 Catching the Pause event

The Spel class will signal your .NET application that a pause has occurred.

You can catch the Pause event in the EventReceived event for the Spel class.

#### VB Example:

```
Sub m_spel_EventReceived (ByVal sender As Object, ByVal e
As RCAPINet.SpelEventArgs) Handles m_spel.EventReceived
    Select Case e.Event
        Case RCAPINet.SpelEvents.Pause
            btnPause.Enabled = False
            btnContinue.Enabled = True
    End Select
End Sub
```

#### C# Example:

```
void m_spel_EventReceived(object sender, SpelEventArgs e)
{
    switch(e.Event)
    {
        case SpelEvents.Pause:
            btnPause.Enabled = false;
            btnContinue.Enabled = true;
            break;
        default:
            break;
    }
}
```



### 8.3 Executing Pause

The following routine shows how to issue a PAUSE from Visual Basic using the *Pause* method.

**VB Example:**

```
Sub btnPause_Click( _  
    ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) _  
    Handles btnPause.Click  
  
    m_spel.Pause()  
    btnPause.Enabled = False  
    btnContinue.Enabled = True  
End Sub
```

**C# Example:**

```
void btnPause_Click(object sender, EventArgs e)  
{  
    m_spel.Pause();  
    btnPause.Enabled = false;  
    btnContinue.Enabled = true;  
}
```

### 8.4 Continue after pause

To continue after a pause has occurred, use the *Continue* method.

#### VB Example:

```
Sub btnContinue_Click( _  
    ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) _  
    Handles btnContinue.Click  
  
    m_spel.Continue()  
    btnContinue.Enabled = False  
    btnPause.Enabled = True  
End Sub
```

#### C# Example:

```
void btnContinue_Click(object sender, EventArgs e)  
{  
    m_spel.Continue();  
    btnContinue.Enabled = false;  
    btnPause.Enabled = true;  
}
```

### 8.5 Abort after pause

You can also execute the *Stop* method if you don't want to continue after a pause.

#### VB Example:

```
Sub btnStop_Click( _  
    ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) _  
    Handles btnStop.Click  
  
    m_spel.Stop()  
    btnContinue.Enabled = False  
    btnPause.Enabled = False  
End Sub
```

#### C# Example:

```
void btnStop_Click(object sender, EventArgs e)  
{  
    m_spel.Stop();  
    btnContinue.Enabled = true;  
    btnPause.Enabled = false;  
}
```

## 9. Handling Emergency Stop

When an Emergency stop occurs, you may want to perform some specific action in your program, such as displaying a dialog, or a message box.

The Spel class issues two standard events for emergency stop status: EStopOn and EStopOff.

### 9.1 Using system EStop events

You can catch the system EStop events in the EventReceived handler in your Visual Basic application.

```
Imports RCAPINet.Spel

Private Sub m_spel_EventReceived(ByVal sender As Object,
ByVal e As SpelEventArgs) Handles m_spel.EventReceived
    Select Case e.Event
        Case RCAPINet.SpelEvens.EstopOn
            MsgBox "E-Stop detected"
            gEStop = True
            lblEStop.BackColor = Color.Red
            lblEStop.Text = "EStop ON"
        Case RCAPINet.SpelEvents.EstopOff
            gEStop = False
            lblEStop.BackColor = Color.Green
            lblEStop.Text = "EStop OFF"
    End Select
End Sub
```

You can catch the system EStop events in the EventReceived handler in your C# application.

```
private void m_spel_EventReceived(object sender,
SpelEventArgs e)
{
    switch(e.Event)
    {
        case SpelEvents.EstopOn:
            MessageBox.Show("E-Stop detected");
            gEStop = true;
            lblEStop.BackColor = Color.Red;
            lblEStop.Text = "EStop ON";
        case SpelEvents.EstopOff:
            gEStop = false;
            lblEStop.BackColor = Color.Green;
            lblEStop.Text = "EStop OFF";
    }
}
```

# 10. EPSON RC+ 7.0 Windows and Dialogs

You can open certain EPSON RC+ 7.0 windows and dialogs from your .NET application using the ShowWindow and RunDialog methods of the Spel class.

## 10.1 Windows

Windows are non-modal, meaning that they can remain open while other elements of your Visual Basic GUI can be used. You can show and hide EPSON RC+ 7.0 windows from your Visual Basic program.

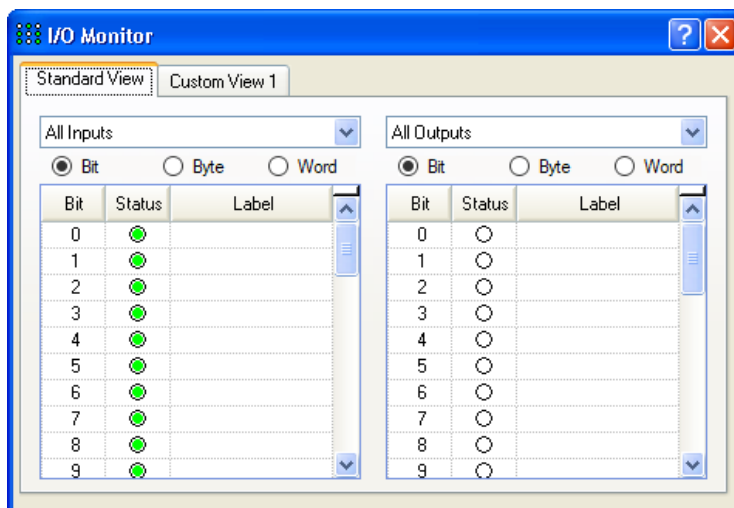
For example, to open and close the I/O Monitor window:

```
m_spel.ShowWindow(RCAPINet.SpelWindows.IOMonitor, Me)
m_spel.HideWindow(RCAPINet.SpelWindows.IOMonitor)
```

Windows are non-modal, meaning that they can remain open while other elements of your C# GUI can be used. You can show and hide EPSON RC+ 7.0 windows from your C# program.

```
m_spel.ShowWindow(RCAPINet.SpelWindows.IOMonitor, this);
m_spel.HideWindow(RCAPINet.SpelWindows.IOMonitor);
```

WindowID	Window
RCAPINet.SpelWindows.IOMonitor	IO Monitor
RCAPINet.SpelWindows.TaskManager	Task Manager
RCAPINet.SpelWindows.ForceMonitor	Force Monitor
RCAPINet.SpelWindows.Simulator	Simulator



*I/O Monitor Window*

## 10.2 Dialogs

Dialogs are modal: when a dialog is opened, other elements of your .NET GUI cannot be used until the dialog is closed.

For example, to open the Robot Manager dialog:

```
m_spel.RunDialog(RCAPINet.SpelDialogs.RobotManager)
```

Once a dialog has been opened, it must be closed by the operator. You cannot close a dialog from within your program. This is for safety reasons.

The following table shows the dialogs that can be opened.

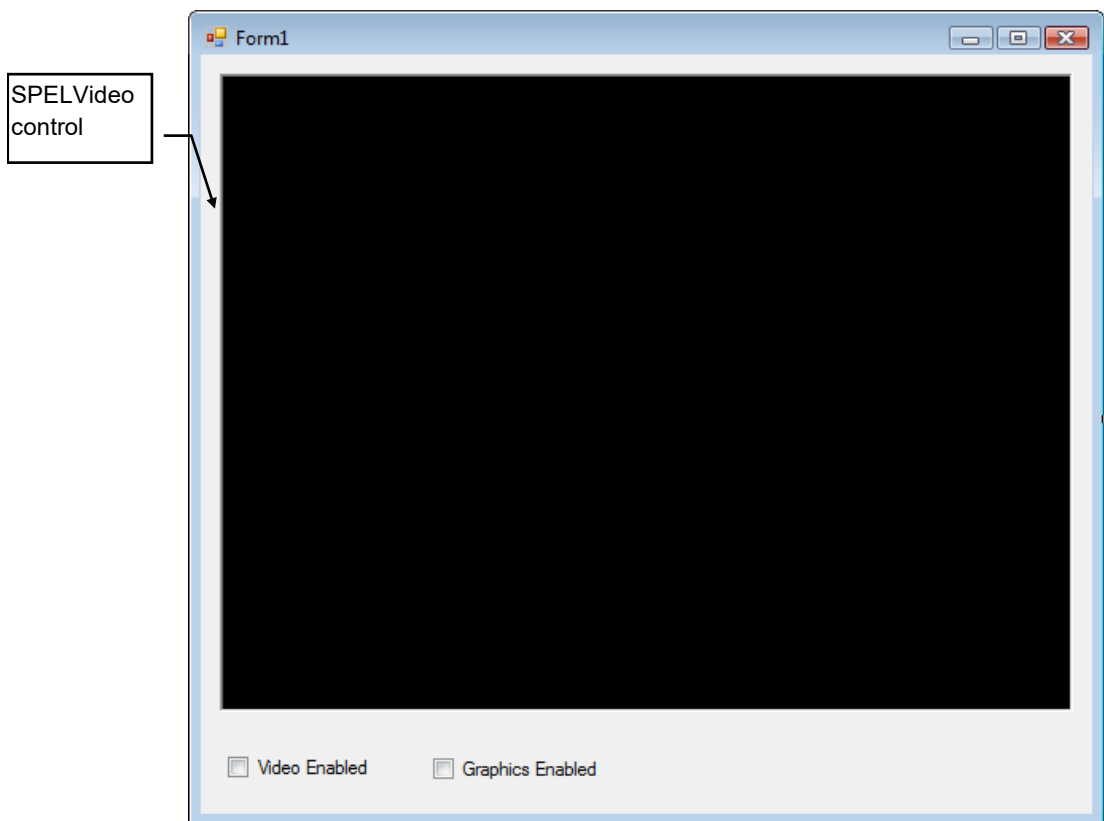
DialogID	Dialog
RCAPINet.SpelDialogs.RobotManager	Robot Manager
RCAPINet.SpelDialogs.ControllerTools	Controller Tools
RCAPINet.SpelDialogs.VisionGuide	Vision Guide
RCAPINet.SpelDialogs.ForceGuide	Force Guide
RCAPINet.SpelDialogs.PartFeeding	Part Feeding

# 11. Displaying Video

You can display video on a form in your application by using the SPELVideo control. When you run a vision sequence, the graphics can also be displayed on the window.

Perform the following steps to create a video display:

1. Add the SPELVideo component to your project. To add the control to your Visual Studio .NET toolbox, right click on the toolbox and select Choose Items. Select the Browse tab and browse to the \EpsonRC70\Exe directory and select the RCAPINet.dll file. The SPELVideo control icon will be added to the toolbox.
2. Place a SPELVideo control on the form you want the video to be displayed. The control size can be changed up to the full size.
3. Set the VideoEnabled property to True.
4. Set the GraphicsEnabled property to True if you want to display vision graphics. You must also attach the SPELVideo control to a Spel class instance using the Spel class SpelVideoControl property.



*SPELVideo control placed on a form*

When the GraphicsEnabled property is True and the control is attached to a Spel class instance, then vision graphics will be displayed whenever the VRun method is executed on the Controller connected to the Spel class instance.

Here is an example showing how to enable video and graphics on a Visual Basic form where a Spel class instance is used and a SPELVideo control have been placed:

```
Private Sub Form_Load(sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    m_spel = New Spel
    m_spel.Initialize()
    m_spel.Project =
"c:\EpsonRC70\projects\test\test.sprj"
    SpelVideo1.VideoEnabled = True
    SpelVideo1.GraphicsEnabled = True
    m_spel.SpelVideoControl = SPELVideo1
End Sub
```

Here is an example showing how to enable video and graphics on a C# form where a Spel class instance is used and a SPELVideo control have been placed:

```
private void Form_Load(object sender, EventArgs e)
{
    RCAPINet.Spel m_spel = new RCAPINet.Spel();
    m_spel.Initialize();
    m_spel.Project = @"c:\EpsonRC70\projects\test\test.sprj";
    SpelVideo1.VideoEnabled = True;
    SpelVideo1.GraphicsEnabled = True;
    m_spel.SpelVideoControl = SPELVideo1;
}
```

## Using multiple video displays

Starting with EPSON RC+ 7.0 version 7.3.0 or later, you can use multiple video displays in your application. For each display, you can select which camera video to display.

To use multiple displays, you must set the SpelVideoControl property for each display.

The example below shows initialization that includes two video displays.

### VB Example:

```
Private Sub Form_Load(sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    m_spel = New Spel
    m_spel.Initialize()
    m_spel.Project =
"c:\EpsonRC70\projects\test\test.sprj"
    SpelVideo1.VideoEnabled = True
    SpelVideo1.GraphicsEnabled = True
    SpelVideo1.Camera = 1
    SpelVideo2.VideoEnabled = True
    SpelVideo2.GraphicsEnabled = True
    SpelVideo2.Camera = 2
    m_spel.SpelVideoControl = SPELVideo1
    m_spel.SpelVideoControl = SPELVideo2
End Sub
```

### C# Example:

```
private void Form_Load(object sender, EventArgs e)
{
    RCAPINet.Spel m_spel = new RCAPINet.Spel();
    m_spel.Initialize();
    m_spel.Project =
@"c:\\EpsonRC70\\project\\test\\test.sprj";
    SpelVideo1.VideoEnabled = true;
    SpelVideo1.GraphicsEnabled = true;
    SpelVideo1.Camera = 1;
    SpelVideo2.VideoEnabled = true;
    SpelVideo2.GraphicsEnabled = true;
    SpelVideo2.Camera = 2;
    m_spel.SpelVideoControl = SPELVideo1;
    m_spel.SpelVideoControl = SPELVideo2;
}
```



## 12. Using AsyncMode

AsyncMode allows you to execute Spel methods while other methods are executing. Only the following Spel class methods are allowed to execute asynchronously:

Arc	Jump3
Arc3	Jump3CP
BGo	MCal
BMove	Move
CVMove	Pass
Go	PTran
Home	Pulse
JTran	TGo
Jump	TMove

To execute a method asynchronously, set the AsyncMode property to True, then execute the method. When the AsyncMode property is true and you execute an asynchronous method, the method will be started and control will return immediately back to the .NET application for further processing.

If you execute another asynchronous method while a previous one is executing, SPEL will wait for the first method to complete, then start the next method and return back to .NET.

To wait for an asynchronous method to complete, you can use one of the following:

- Execute the WaitCommandComplete method.
- Set AsyncMode property to False.

If an asynchronous command cannot be started due to an error (e.g. point does not exist), then an exception will occur immediately. However, if an error occurs during a command running asynchronously, the error exception occurs on the next execution of an asynchronous command or execution of WaitCommandComplete, or AsyncMode is set to False. If the exception occurs on the next command, you do not know which statement caused the error (the previous statement or the current statement). If you need to check if an asynchronous command completed successfully before executing another command, then call WaitCommandComplete before the next command. If an error occurred during the previous asynchronous command, a SpelException exception will occur with the error number and message. See the example below.

**VB Example:**

```
Try
    m_spel.AsyncMode = True
    m_spel.Go(1)
    ' do other things here during motion
    ' When Go(2) executes, an exception occurs if Go(1) had
    ' an error during execution, so we don't know if error
    ' occurred for Go(1) or Go(2)
    m_spel.Go(2)

    m_spel.Go(3)
    ' do other things here during motion
    ' Check if Go(3) was successful
    m_spel.WaitCommandComplete() ' Exception occurs if Go(3) had an
error

    m_spel.Go(4)
Catch ex As SpelException
    ' Handle the error exception

End Try
```

**C# Example:**

```
try {
    m_spel.AyncMode = true;
    m_spel.Go(1);
    //do things here during motion
    //When Go(2) executes, an exception occurs if Go(1) had
    //an error during execution, so we don't know if error
    //occurred for Go(1) or Go(2)
    m_spel.Go(2);

    m_spel.Go(3);
    //do other things here during motion
    //Check if Go(3) was successful
    m_spel.WaitCommandComplete();
    //Exception occurs if Go(3) had an error

    m_spel.Go(4);
}
catch (RCAPINet.SpelException ex) {
    //Handle the error exception
}
```

## 13. SPELCom\_Event

Generates a user event from a Spel class instance.

### Syntax

**SPELCom\_Event** *eventNumber* [, *msgArg1*, *msgArg2*, *msgArg3*,... ]

### Parameters

*eventNumber* An integer expression whose value is from 1000 - 32767.  
*msgArg1*, *msgArg2*, *msgArg3*... Optional. Each message argument can be either a number, string literal, or a variable name.

### Description

This instruction makes it easy to send real time information to an application from a Spel task running in the Controller. For example, you can update parts count, lot number, etc. by sending an event.

### SPELCom\_Event Example

In this example, a SPEL+ task sends cycle data to an application using the RC+ API .

```
Function RunParts
    Integer cycNum
    String lot$
    Double cycTime

    cycNum = 0
    Do
        TmrReset(0)
        ...
        ...
        cycTime = Tmr(0)
        cycNum = cycNum + 1
        Spelcom_Event 3000, cycNum, lot$, cycTime
        Wait 0.01
    Loop
Fend
```

## 14. RCAPINet Reference

### 14.1 Spel Class

#### Description

This class allows you to execute commands and receive events from EPSON RC+ 7.0.

#### File Name

RCAPINet.dll (64-bit and 32-bit)

### 14.2 Spel Class Properties

#### AsyncMode Property, Spel Class

#### Description

Sets / returns asynchronous execution mode.

#### Syntax

Property **AsyncMode** As Boolean

#### Default value

False

#### Return value

A Boolean value that is True if asynchronous mode is active, False if not.

#### See Also

Using AsyncMode, WaitCommandComplete

#### AsyncMode Example

##### VB Example:

```
With m_spel
    .AsyncMode = True
    .Jump("pick")
    .Delay(500)
    .On(1)
    .WaitCommandComplete()
End With
```

##### C# Example:

```
m_spel.AsyncMode = true;
m_spel.Jump("pick");
m_spel.Delay(500);
m_spel.On(1);
m_spel.WaitCommandComplete();
```

**AvoidSingularity Property, Spel Class****Description**

Sets / returns singularity avoidance mode.

**Syntax**

Property **AvoidSingularity** As Boolean

**Default value**

False

**Return value**

A Boolean value that is True if singularity avoidance is active, False if not.

**See Also**

Go, Jump, Move

**AvoidSingularity Example****VB Example:**

```
m_spel.AvoidSingularity = True
```

**C# Example:**

```
m_spel.AvoidSingularity = true;
```

**CommandInCycle Property, Spel Class**

**Description**

Returns whether a method is being executed.

**Syntax**

ReadOnly Property **CommandInCycle** As Boolean

**Return value**

A Boolean value that is True if a method is executing, False if not.

**See Also**

AsyncMode

**CommandInCycle Example**

**VB Example:**

```
If m_spel.CommandInCycle Then
    MsgBox "A SPEL command is executing, operation aborted"
End If
```

**C# Example:**

```
if (m_spel.CommandInCycle)
    MessageBox.Show("SPEL command is executing, operation
aborted");
```

---

**CommandTask Property, Spel Class**
**Description**

Specifies the reserved API task to use in the Controller for executing robot commands.

**Syntax**

Property **CommandTask** As Integer

**Default Value**

The default value is 0 (do not use a reserved API task).

**Remarks**

Use **CommandTask** when you want to execute Spel robot commands on another thread in the Controller. Normally, **CommandTask** is used on a multi-robot system. Before using **CommandTask**, you must first reserve tasks in the Controller to be used by the API from EPSON RC+ menu -[Setup]-[System Configuration]-[Controller]-[Preferences]. You can reserve up to 16 API tasks in the Controller.

**See Also**

ServerInstance

**CommandTask Example****VB Example:**

```
' In Robot1 thread
m_spel.CommandTask = 1
m_spel.Robot = 1
```

```
' In Robot2 thread
m_spel.CommandTask = 2
m_spel.Robot = 2
```

**C# Example:**

```
// In Robot1 thread
m_spel.CommandTask = 1;
m_spel.Robot = 1;
```

```
// In Robot2 thread
m_spel.CommandTask = 2;
m_spel.Robot = 2;
```

DisableMsgDispatch Property, Spel Class

**Description**

Sets / returns whether Windows messages should be processed during Spel method execution.

**Syntax**

**DisableMsgDispatch**

**Type**

Boolean

**Default Value**

False

**Remarks**

This property should normally not be used. It is intended for special applications that do not want keyboard or mouse processing while a Spel method is executing.



**ErrorCode Property, Spel Class****Description**

Returns the current Controller error code.

**Syntax**

ReadOnly Property **ErrorCode** As Integer

**Return Value**

Integer value containing the error code.

**See Also**

ErrorOn

**ErrorCode Example****VB Example:**

```
If m_spel.ErrorOn Then
    lblErrorCode.Text = m_spel.ErrorCode.ToString()
Else
    lblErrorCode.Text = ""
End If
```

**C# Example:**

```
if (m_spel.ErrorOn)
    lblErrorCode.Text = m_spel.ErrorCode.ToString();
else
    lblErrorCode.Text = "";
```

**ErrorOn Property, Spel Class**

**Description**

Returns True if a critical error has occurred in the Controller.

**Syntax**

ReadOnly Property **ErrorOn** As Boolean

**Return Value**

True if the Controller is in the error state, False if not.

**Remarks**

When the Controller is in the error state, the ErrorOn property returns True, and you can retrieve the error code by using the ErrorCode property.

**See Also**

ErrorCode

**ErrorOn Example**

**VB Example:**

```
If m_spel.ErrorOn Then  
    m_spel.Reset  
End If
```

**C# Example:**

```
if (m_spel.ErrorOn)  
    m_spel.Reset ();
```

**EStopOn Property, Spel Class****Description**

Returns the status of the Controller's emergency stop.

**Syntax**

ReadOnly Property **EStopOn** As Boolean

**Return Value**

True if the emergency stop is active, False if not.

**EStopOn Example****VB Example:**

```
If m_spel.EStopOn Then
    lblEStop.Text = "Emergency stop is active"
Else
    lblEStop.Text = ""
EndIf
```

**C# Example:**

```
if (m_spel.EStopOn)
    lblEStop.Text = "Emergency stop is active";
else
    lblEStop.Text = "";
```

**Force\_Sensor Property, Spel Class**

**Description**

Sets and return the current force sensor number.

**Syntax**

Property **Force\_Sensor** As Integer

**Default value**

1

**Return value**

An Integer value that is the current force sensor number

**Remarks**

Before using any force methods, you must set the current force sensor using this property.

**See Also**

Force\_Calibrate, Force\_GetForces, Force\_SetTrigger

**Force\_Sensor Example**

**VB Example:**

```
' Read the Z axis force for sensor 2  
m_spel.Force_Sensor = 2  
f = m_spel.Force_GetForce(3)
```

**C# Example:**

```
// Read the Z axis force for sensor 2  
m_spel.Force_Sensor = 2;  
f = m_spel.Force_GetForce(3);
```

---

**MotorsOn Property, Spel Class****Description**

Sets and return the status of the motor power on or off for the current robot.

**Syntax**

Property **MotorsOn** As Boolean

**Default value**

False

**Return value**

A Boolean value that is True if motors are on, False if not.

**See Also**

PowerHigh, Reset, Robot

**MotorsOn Example****VB Example:**

```
If Not m_spel.MotorsOn Then  
    m_spel.MotorsOn = True  
End If
```

**C# Example:**

```
if (!m_spel.MotorsOn)  
    m_spel.MotorsOn = true;
```

## NoProjectSync Property, Spel Class

### Description

Sets / returns whether the current project in the PC should be synchronized with the Controller project.

### Syntax

**NoProjectSync**

### Type

Boolean

### Default Value

False

### Remarks

When NoProjectSync is set to False (default), then the Spel class ensures that the project on the PC is synchronized with the project on the Controller.

When NoProjectSync is set to True, the Spel class does not check for any project on the PC and does not synchronize the PC project with the Controller. This allows you to run programs in the Controller without any project on the PC.

This property is not persistent. You must set it after creating a Spel class instance if you want to set it to True.

### See Also

Start

### NoProjectSync Examples

#### VB Example:

```
m_spel.Initialize()  
m_spel.NoProjectSync = True
```

#### C# Example:

```
m_spel.Initialize();  
m_spel.NoProjectSync = true;
```

## OperationMode Property, Spel Class

**Description**

Reads or sets the EPSON RC+ 7.0 mode of operation.

**Syntax**

Property **OperationMode** As SpelOperationMode

**Return value**

SpelOperationMode value

**Remarks**

When **OperationMode** is set to Program, the EPSON RC+ 7.0 GUI for the current instance of the Spel class is opened and the Controller operation mode is set to Program. If the user closes the GUI, **OperationMode** is set to Auto. If **OperationMode** is set to Auto from Visual Basic, the GUI also closes.

**OperationMode Example****VB Example:**

```
Sub btnSpelProgramMode_Click _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnHideIOMonitor.Click
    Try
        m_spel.OperationMode = _
            RCAPINet.SpelOperationMode.Program
        ' If you want to wait for the user to close the RC+ GUI,
        ' you can wait here for OperationMode to change to Auto
    Do
        Application.DoEvents()
        System.Threading.Thread.Sleep(10)
    Loop Until m_spel.OperationMode = _
        RCAPINet.SpelOperationMode.Auto
    Catch ex As RCAPINet.SpelException
        MsgBox(ex.Message)
    End Try
End If
```

**C# Example:**

```
void btnSpelProgramMode_Click(object sender, EventArgs e)
{
    try {
        m_spel.OperationMode = RCAPINet.SpelOperationMode.Auto;
        // If you want to wait for the user to close the RC+ GUI, you can wait here
        Do {
            Application.DoEvents();
            System.Threading.Thread.Sleep(10);
        } while (!m_spel.OperationMode =
RCAPINet.OperationMode.Auto);
    }
    Catch (SpelException ex){
        MessageBox.Show(ex.Message);
    }
}
```



---

**ParentWindowHandle Property, Spel Class****Description**

Sets / returns the handle for the parent window used for dialogs and windows.

**Syntax**

Property **ParentWindowHandle** As Integer

**Return Value**

Integer value containing the window handle.

**Remarks**

Use **ParentWindowHandle** to specify the parent window from applications that do not have .NET forms, such as LabVIEW.

**See Also**

ShowWindow

**ParentWindowHandle Example****VB Example:**

```
m_spel.ParentWindowHandle = Me.Handle  
m_spel.ShowWindow(RCAPINet.SpelWindows.IOMonitor)
```

**C# Example:**

```
m_spel.ParentWindowHandle = (int)this.Handle;  
m_spel.ShowWindow(RCAPINet.SpelWindows.IOMonitor);
```

PauseOn Property, Spel Class

**Description**

Returns status of the Controller pause state.

**Syntax**

ReadOnly Property **PauseOn** As Boolean

**Return Value**

True if the Controller is in the pause state, False if not.

**See Also**

Continue Pause

**PauseOn Example**

**VB Example:**

```
If m_spel.PauseOn Then
    btnPause.Enabled = False
    btnContinue.Enabled = True
End If
```

**C# Example:**

```
if (m_spel.PauseOn) {
    btnPause.Enabled = false;
    btnContinue.Enabled = true;
}
```

---

**PowerHigh Property, Spel Class****Description**

Sets and returns the power state for the current robot.

**Syntax**

Property **PowerHigh** As Boolean

**Default Value**

False

**Return Value**

True if the current robot power is high, False if not.

**See Also**

MotorsOn

**PowerHigh Example****VB Example:**

```
If Not m_spel.PowerHigh Then
    m_spel.PowerHigh = True
End If
```

**C# Example:**

```
if (!m_spel.PowerHigh)
    m_spel.PowerHigh = true;
```

Project Property, Spel Class

**Description**

Sets / returns the current project.

**Syntax**

Property **Project** As String

**Default Value**

Empty string.

**Return Value**

A string containing the project path and file.

**Remarks**

When setting the **Project**, you must supply the full path and name of the EPSON RC+ 7.0 project make file. The make file is the project name with a .SPRJ extension.

**Project Example**

**VB Example:**

```
m_spel.Project = "c:\EpsonRC70\projects\myapp\myapp.sprj"
```

**C# Example:**

```
m_spel.Project = @"c:\EpsonRC70\projects\myapp\myapp.sprj";
```

---

**ProjectBuildComplete Property, Spel Class****Description**

Returns the status of the current project build.

**Syntax**

ReadOnly Property **ProjectBuildComplete** As Boolean

**Return Value**

True if the project build is complete, False if not.

**See Also**

BuildProject

**ProjectBuildComplete Example****VB Example:**

```
If m_spel.ProjectBuildComplete Then
    lblBuild.Text = "Project build is Complete"
Else
    lblBuild.Text = "Project build is not Complete"
End If
```

**C# Example:**

```
if (m_spel.ProjectBuildComplete)
    lblBuild.Text = "Project build is Complete";
else
    lblBuild.Text = "Project build is not Complete";
```

## ProjectOverwriteWarningEnabled Property, Spel Class

**Description**

Sets / returns whether the project overwrite warning will be displayed.

**Syntax**

Property **ProjectOverwriteWarningEnabled** As Boolean

**Default Value**

True

**Return Value**

True if the project overwrite warning is enabled, False if not.

**See Also**

BuildProject

**Remarks**

By default, when the current project is not the same as the project in the controller, then a project overwrite warning message is displayed when the project is built and sent to the controller. Set **ProjectOverwriteWarningEnabled** to False when you don't want the overwrite warning message to be displayed. This is useful for when your application needs to switch projects used in the controller.

**ProjectOverwriteWarningEnabled Example****VB Example:**

```
' Disable the project overwrite warning
m_spel.ProjectOverwriteWarningEnabled = False
m_spel.Project =
"c:\EpsonRC70\Projects\Project1\Project1.sprj"
```

**C# Example:**

```
// Disable the project overwrite warning
m_spel.ProjectOverwriteWarningEnabled = false;
m_spel.Project =
@"c:\EpsonRC70\Projects\Project1\Project1.sprj";
```

**ResetAbortEnabled Property, Spel Class****Description**

Sets / returns whether ResetAbort method should be enabled or not.

**Syntax**

Property **ResetAbortEnabled** As Boolean

**Default Value**

True

**Return Value**

True if ResetAbort is enabled, False if not.

**See Also**

ResetAbort

**ResetAbortEnabled Example****VB Example:**

```
' Enable reset abort  
m_spel.ResetAbortEnabled = True
```

**C# Example:**

```
// Enable reset abort  
m_spel.ResetAbortEnabled = true;
```

Robot Property, Spel Class

**Description**

Sets / returns the current robot number.

**Syntax**

Property **Robot** As Integer

**Default Value**

If one or more robots exists, the default value for the first Spel instance is 1, otherwise it is 0. For all other Spel instances, the default value is 0.

**Return Value**

Integer value that contains the current robot number.

**Remarks**

On a system using multiple robots, use the **Robot** property to set the robot for subsequent robot related commands, such as motion commands.

**See Also**

RobotModel, RobotType

**Robot Example**

**VB Example:**

```
m_spel.Robot = 2
If Not m_spel.MotorsOn Then
    m_spel.MotorsOn = True
End If
```

**C# Example:**

```
m_spel.Robot = 2;
if (!m_spel.MotorsOn)
    m_spel.MotorsOn = true;
```



---

**RobotModel Property, Spel Class****Description**

Returns the model name for the current robot.

**Syntax**

ReadOnly Property **RobotModel** As String

**Return Value**

String that contains the current robot's model name.

**See Also**

Robot, RobotType

**RobotModel Example****VB Example:**

```
lblRobotModel.Text = m_spel.RobotModel
```

**C# Example:**

```
lblRobotModel.Text = m_spel.RobotModel;
```

## RobotType Property, Spel Class

**Description**

Returns the type of the current robot.

**Syntax**

ReadOnly Property **RobotType** As SpelRobotType

**Return Value**

SpelRobotType value

**See Also**

Robot, RobotModel

**RobotType Example****VB Example:**

```
Select Case m_spel.RobotType
    Case RCAPINet.SpelRobotType.Scara
        lblRobotType.Text = "Scara"
    Case RCAPINet.SpelRobotType.Cartesian
        lblRobotType.Text = "Cartesian"
End Select
```

**C# Example:**

```
switch (m_spel.RobotType)
{
    case SpelRobotType.Scara:
        lblRobotType.Text = "Scara";
        break;
    case SpelRobotType.Cartesian:
        lblRobotType.Text = "Cartesian";
        break;
    default:
        break;
}
```

---

**SafetyOn Property, Spel Class****Description**

Returns status of the Controller's safeguard input.

**Syntax**

ReadOnly Property **SafetyOn** As Boolean

**Return Value**

True if the safeguard is open, False if not.

**Remarks**

Use the SafetyOn property to obtain the safeguard status when your application starts, then use the SafeguardOpen and SafeguardClose events to update the status.

**SafetyOn Example****VB Example:**

```
If m_spel.SafetyOn Then
    lblSafeguard.Text = "Safe guard is active"
Else
    lblSafeguard.Text = ""
End If
```

**C# Example:**

```
if (m_spel.SafetyOn)
    lblSafeguard.Text = "Safe guard is active";
else
    lblSafeguard.Text = "";
```

## ServerInstance Property, Spel Class

**Description**

Specifies which instance of EPSON RC+ server to use.

**Syntax**

Property **ServerInstance** As Integer

**Default Value**

The default value is the next available server instance.

**Remarks**

The API communicates with an RC+ server process. **ServerInstance** specifies which server to use. Each server instance corresponds with one controller and one project.

By default, when you create a new Spel class instance, the **ServerInstance** is automatically set to 1.

Sometimes you may want multiple instances of the Spel class for the same Controller, such as for multithreading in your application. In that case, you must set the **ServerInstance** property to the same value for each Spel class instance using the same controller.

**ServerInstance** must be between 1 and 10 and it must be set before executing **Initialize** or any other methods.

**See Also**

CommandTask, Initialize

**ServerInstance Example****VB Example:**

```
' Controller 1
spell = New Spel
spell.ServerInstance = 1
spell.Initialize()
spell.Connect(1)

' Controller 2
spel2 = New Spel
spel2.ServerInstance = 2
spel2.Initialize()
spel2.Connect(2)
```

**C# Example:**

```
// Controller 1
RCAPINet.Spel spell = new RCAPINet.Spel();
spell.ServerInstance = 1;
spell.Initialize();
spell.Connect(1);

// Controller 2
RCAPINet.Spel spel2 = new RCAPINet.Spel();
spel2.ServerInstance = 2;
spel2.Initialize();
spel2.Connect(2);
```

---

**SPELVideoControl Property, Spel Class****Description**

Used to connect a SPELVideo control to the Spel class instance so that video and graphics can be displayed.

**Syntax**

Property **SpelVideoControl** As SpelVideo

**See Also**

Graphics Enabled, VideoEnabled, Camera

**SpelVideoControl Example****VB Example:**

```
m_spel.SpelVideoControl = SpelVideo1
```

**C# Example:**

```
m_spel.SpelVideoControl = SpelVideo1;
```

Version Property, Spel Class

**Description**

Returns the current EPSON RC+ 7.0 software version.

**Syntax**

ReadOnly Property **Version** As String

**Return Value**

String that contains the current EPSON RC+ 7.0 software version.

**Version Example**

**VB Example:**

```
' Get version of software  
curVer = m_spel.Version
```

**C# Example:**

```
// Get version of software  
curVer = m_spel.Version;
```

**WarningCode Property, Spel Class****Description**

Returns Controller warning code.

**Syntax**

ReadOnly Property **WarningCode** As Integer

**Return Value**

Integer value that contains the current controller warning code.

**See Also**

WarningOn

**WarningCode Example****VB Example:**

```
If m_spel.WarningOn Then
    lblWarningCode.Text = m_spel.WarningCode.ToString()
Else
    lblWarningCode.Text = ""
End If
```

**C# Example:**

```
if (m_spel.WarningOn)
    lblWarningCode.Text = m_spel.WarningCode.ToString();
else
    lblWarningCode.Text = "";
```

## WarningOn Property, Spel Class

### Description

Returns status of the Controller warning state.

### Syntax

ReadOnly Property **WarningOn** As Boolean

### Return Value

True if the Controller is in the warning state, False if not.

### See Also

WarningCode

### WarningOn Example

#### VB Example:

```
If m_spel.WarningOn Then
    lblWarningStatus.Text = "ON"
Else
    lblWarningStatus.Text = "OFF"
End If
```

#### C# Example:

```
if (m_spel.WarningOn)
    lblWarningStatus.Text = "ON";
else
    lblWarningStatus.Text = "OFF";
```



## 14.3 Spel Class Methods

### Accel Method, Spel Class

#### Description

Sets acceleration and deceleration for point to point motion commands Go, Jump, and Pulse.

#### Syntax

Sub **Accel** (*PointToPointAccel* As Integer, *PointToPointDecel* As Integer, \_  
           [*JumpDepartAccel* As Integer], [*JumpDepartDecel* As Integer], \_  
           [*JumpApproAccel* As Integer], [*JumpApproDecel* As Integer])

#### Parameters

<i>PointToPointAccel</i>	Integer expression between 1-100 representing a percentage of maximum acceleration rate.
<i>PointToPointDecel</i>	Integer expression between 1-100 representing a percentage of maximum deceleration rate.
<i>JumpDepartAccel</i>	Integer expression between 1-100 representing a percentage of maximum acceleration rate for Jump command Z Axis upward motion.
<i>JumpDepartDecel</i>	Integer expression between 1-100 representing a percentage of maximum deceleration rate for Jump command Z Axis upward motion.
<i>JumpApproAccel</i>	Integer expression between 1-100 representing a percentage of maximum acceleration rate for Jump command Z Axis downward motion.
<i>JumpApproDecel</i>	Integer expression between 1-100 representing a percentage of maximum deceleration rate for Jump command Z Axis downward motion.

#### See Also

Accels, Speed

#### Accel Example

##### VB Example:

```
m_spel.Accel (50, 50)
m_spel.Go ("pick")
```

##### C# Example:

```
m_spel.Accel (50, 50);
m_spel.Go ("pick");
```

## AccelR Method, Spel Class

**Description**

Sets acceleration and deceleration for tool rotation motion.

**Syntax**

Sub **AccelR** (*Accel* As Single, [*Decel* As Single])

**Parameters**

*Accel* Single expression from 0.1 to 5000 deg/sec<sup>2</sup> to define tool rotation acceleration when ROT is used in motion commands. If Decel is omitted, this value is used for both the Acceleration and Deceleration rates.

*Decel* Optional. Single expression from 0.1 to 5000 deg/sec<sup>2</sup> to define tool rotation deceleration when ROT is used in motion commands.

**See Also**

Arc, Arc3, BMove, Jump3CP, Power, SpeedR, TMove

**AccelR Example****VB Example:**

```
Sub MoveToPlace ()
    m_spel.AccelR(100)
    m_spel.Move("place ROT")
End Sub
```

**C# Example:**

```
void MoveToPlace ()
{
    m_spel.AccelR(100);
    m_spel.Move("place ROT");
}
```

---

**AccelS Method, Spel Class**
**Description**

Sets acceleration and deceleration for linear interpolator (straight line) motion commands Jump3CP, Move, TMove.

**Syntax**

Sub **AccelS** (*Accel* As Single, *Decel* As Single, [*JumpDepartAccel* As Single], [*JumpDepartDecel* As Single], \_ [*JumpApproAccel* As Single], [*JumpApproDecel* As Single] )

**Parameters**

<i>Accel</i>	Single expression between 1-5000 represented in mm/sec <sup>2</sup> units to define acceleration and deceleration values for Straight Line and Continuous Path motion. If Decel is omitted, this value is used for both the Acceleration and Deceleration rates.
<i>Decel</i>	Single expression between 1-5000 represented in mm/sec <sup>2</sup> units to define deceleration values for Straight Line and Continuous Path motion. One parameter is used for representing both the Acceleration and Deceleration rates.
<i>JumpDepartAccel</i>	Single expression between 1-5000 representing a percentage of maximum acceleration rate for Jump3CP command Z Axis upward motion.
<i>JumpDepartDecel</i>	Single expression between 1-5000 representing a percentage of maximum deceleration rate for Jump3CP command Z Axis upward motion.
<i>JumpApproAccel</i>	Single expression between 1-5000 representing a percentage of maximum acceleration rate for Jump3CP command Z Axis downward motion.
<i>JumpApproDecel</i>	Single expression between 1-5000 representing a percentage of maximum deceleration rate for Jump3CP command Z Axis downward motion.

**See Also**

Accel, SpeedS, Jump3CP, Move, TMove

**AccelS Example****VB Example:**

```
Sub MoveToPlace ()
    m_spel.AccelS(500)
    m_spel.Move(pick)
    m_spel.AccelS(500, 300)
    m_spel.Move(place)
End Sub
```

**C# Example:**

```
void MoveToPlace ()
{
    m_spel.AccelS(500);
    m_spel.Move(pick);
    m_spel.AccelS(500, 300);
    m_spel.Move(place);
}
```

Agl Method, Spel Class

**Description**

Returns the joint angle for the selected rotational axis, or position for the selected linear axis.

**Syntax**

Function **Agl** (*JointNumber* As Integer) As Single

**Parameters**

*JointNumber* Integer expression from 1-9 representing the joint number.

**See Also**

Pls, CX - CT

**Agl Example**

**VB Example:**

```
Dim j1Angle As Single  
j1Angle = m_spel.Agl(1)
```

**C# Example:**

```
float j1Angle;  
j1Angle = m_spel.Agl(1);
```

## AIO\_In Method, Spel Class

**Description**

Reads analog value form optional analog I/O input channel.

**Syntax**

Function **AIO\_In** (*Channel* As Integer) As Single

**Parameters**

*Channel* Specify the channel number of the analog I/O.

**Return Value**

Return the analog input value of the analog I/O channel which specified in channel number in real number.

Return value range differs depending on the input range configuration of the analog I/O board.

**See Also**

AIO\_InW, AIO\_Out, AIO\_OutW

**AIO\_In Example****VB Example:**

```
Dim val As Single  
val = m_spel.AIO_In(2)
```

**C# Example:**

```
float val;  
val = m_spel.AIO_In(2);
```

## AIO\_InW Method, Spel Class

**Description**

Reads analog value from optional analog I/O input channel.

**Syntax**

Function **AIO\_InW** (*Channel As Integer*) As Integer

**Parameters**

*Channel* Specify the channel number of the analog I/O.

**Return Value**

Returns the input states (Integer expression from 0 to 65535) of specified analog I/O channel.

The following table shows input voltage (current) and return value of each input channel according to input range configuration of analog I/O board.

Input Data		Input Range Configuration				
Hexadecimal	Decimal	$\pm 10.24(\text{V})$	$\pm 5.12(\text{V})$	0-5.12(V)	0-10.24(V)	0-24(mA)
0xFFFF	65535	10.23969	5.11984	5.12000	10.24000	24.00000
0x8001	32769	0.00031	0.00016	2.56008	5.12016	12.00037
0x8000	32768	0.00000	0.00000	2.56000	5.12000	12.00000
0x0000	0	-10.24000	-5.12000	0.00000	0.00000	0.00000

**See Also**

AIO\_In, AIO\_Out, AIO\_OutW

**AIO\_InW Example****VB Example:**

```
Dim val As Integer
val = m_spel.AIO_InW(2)
```

**C# Example:**

```
int val;
val = m_spel.AIO_InW(2);
```

## AIO\_Out Method, Spel Class

**Description**

Reads or set analog value from the optional analog I/O output channel.

**Syntax**

Function **AIO\_Out** (*Channel* As Integer) As Single

Sub **AIO\_Out** (*Channel* As Integer, *Value* As Single)

**Parameters**

*Channel* Specify the channel number of the analog I/O.

*Value* Specify the real number which indicates output voltage [V] or current.

**Return Value**

Returns specified analog I/O channel voltage and current output state in real number. Unit of voltage output is [V] and current output is [mA].

For Function **AIO\_Out** (*Channel* As Integer) As Single:

When outputting the speed information of the robot on specified channel, a return value can be acquired by this method.

**Remarks**

Output the real value indicating specified voltage [V] or current [mA] to analog output port which specified on channel port. Set the voltage output range of analog output port or selection of voltage and current output by the switch on the board. If setting a value which out of range of analog I/O port, output the border value (maximum and minimum value) which is not out of the range.

For Sub **AIO\_Out** (*Channel* As Integer, *Value* As Single):

The output setting of this method becomes an error if outputting the speed information by specified channel. Stop the speed information output and execute this method.

**See Also**

AIO\_In, AOI\_InW, AIO\_OutW

**AIO\_Out Example****VB Example:**

```
Dim val As Single
val = m_spel.AIO_Out(1)
```

**C# Example:**

```
float val;
val = m_spel.AIO_Out(1);
```

**AIO\_OutW Method, Spel Class**

**Description**

Reads or set analog value from the optional analog I/O output channel.

**Syntax**

Function **AIO\_OutW** (*Channel As Integer*) As Integer

Sub **AIO\_OutW** (*Channel As Integer, OutputData As Integer*)

**Parameters**

*Channel* Specify the channel number of the analog I/O.

*OutputData* Specify the output data (Integer expression from 0 to 65535) in formula or value.

**Return Value**

Returns the output state of specified analog I/O channel in Long integers from 0 to 65535.

The following table shows output voltage (current) and return value of each output channel according to output range configuration of analog I/O board.

Output Data		Output Range Configuration					
Hexadecimal	Decimal	±10(V)	±5(V)	0-5(V)	0-10(V)	4-20(mA)	0-20(mA)
0xFFFF	65535	9.99970	4.99985	5.00000	10.00000	20.00000	20.00000
0x8001	32769	0.00031	0.00015	2.50008	5.00015	12.00024	10.00031
0x8000	32768	0.00000	0.00000	2.50000	5.00000	12.00000	10.00000
0x0000	0	-10.00000	-5.00000	0.00000	0.00000	4.00000	0.00000

This method is available when outputting the speed information of the robot on specified channel.

**See Also**

AIO\_In, AOI\_InW, AIO\_Out

**AIO\_OutW Example**

**VB Example:**

```
Dim val As Integer
val = m_spel.AIO_OutW(1)
```

**C# Example:**

```
int val;
val = m_spel.AIO_OutW(1);
```



## Arc Method, Spel Class

**Description**

Arc moves the arm to the specified point using circular interpolation in the XY plane.

**Syntax**

Sub **Arc** (*MidPoint* As Integer, *EndPoint* As Integer)

Sub **Arc** (*MidPoint* As SpelPoint, *EndPoint* As SpelPoint)

Sub **Arc** (*MidPoint* As String, *EndPoint* As String)

**Parameters**

Each syntax has two parameters that specify the mid point and end point of the arc.

*MidPoint* Specifies the mid point by using an integer, SpelPoint or string expression.

*EndPoint* Specifies the end point by using an integer, SpelPoint or string expression. When using a string expression, you can include ROT, CP, SYNC, a search expression for Till, and a parallel processing statement.

**See Also**

AccelR, AccelS, SpeedR, SpeedS

Arc3, CVMove, Go, Jump, Jump3, Jump3CP, Move

BGo, BMove, TGo, TMove

CP, Till

**Arc Example****VB Example:**

' Points specified using SpelPoint

```
Dim midPoint, endPoint As SpelPoint
midPoint = m_spel.GetPoint("P1")
endPoint = m_spel.GetPoint("P2")
m_spel.Arc(midPoint, endPoint)
```

' Points specified using expressions

```
m_spel.Arc("P1", "P2")
m_spel.Arc("P1", "P2 CP")
```

' Using parallel processing

```
m_spel.Arc("P1", "P2 !D50; On 1; D90; Off 1!")
```

**C# Example:**

// Points specified using SpelPoint

```
SpelPoint midPoint, endPoint;
midPoint = m_spel.GetPoint("P1");
endPoint = m_spel.GetPoint("P2");
m_spel.Arc(midPoint, endPoint);
```

// Points specified using expressions

```
m_spel.Arc("P1", "P2");
m_spel.Arc("P1", "P2 CP");
```

// Using parallel processing

```
m_spel.Arc("P1", "P2 !D50; On 1; D90; Off 1!");
```

## Arc3 Method, Spel Class

**Description**

Arc3 moves the arm to the specified point using circular interpolation in 3 dimensions.

**Syntax**

Sub **Arc3** (*MidPoint* As Integer, *EndPoint* As Integer)

Sub **Arc3** (*MidPoint* As SpelPoint, *EndPoint* As SpelPoint)

Sub **Arc3**(*MidPoint* As String, *EndPoint* As String)

**Parameters**

Each syntax has two parameters that specify the mid point and end point of the arc.

<i>MidPoint</i>	Specifies the mid point by using an integer, SpelPoint or string expression.
<i>EndPoint</i>	Specifies the end point by using an integer, SpelPoint or string expression. When using a string expression, you can include ROT, ECP, CP, SYNC, a search expression for Till, and a parallel processing statement.

**See Also**

AccelR, AccelS, SpeedR, SpeedS  
 Arc, CVMove, Go, Jump, Jump3, Jump3CP, Move  
 BGo, BMove, TGo, TMove  
 CP, Till

**Arc3 Example****VB Example:**

```
' Points specified using SpelPoint
Dim midPoint, endPoint As SpelPoint
midPoint = m_spel.GetPoint("P1")
endPoint = m_spel.GetPoint("P2")
m_spel.Arc3(midPoint, endPoint)

' Points specified using expressions
m_spel.Arc3("P1", "P2")
m_spel.Arc3("P1", "P2 CP")

' Using parallel processing
m_spel.Arc3("P1", "P2 !D50; On 1; D90; Off 1!")
```

**C# Example:**

```
// Points specified using SpelPoint
SpelPoint midPoint, endPoint;
midPoint = m_spel.GetPoint("P1");
endPoint = m_spel.GetPoint("P2");
m_spel.Arc3(midPoint, endPoint);

// Points specified using expressions
m_spel.Arc3("P1", "P2");
m_spel.Arc3("P1", "P2 CP");

// Using parallel processing
m_spel.Arc3("P1", "P2 !D50; On 1; D90; Off 1!");
```

## Arch Method, Spel Class

**Description**

Defines ARCH parameters (Z height to move before beginning horizontal motion) for use with the JUMP instructions.

**Syntax**

Sub **Arch** (*ArchNumber* As Integer, *DepartDist* As Integer, *ApproDist* As Integer)

**Parameters**

<i>ArchNumber</i>	The Arch number to define. Valid Arch numbers are (0-6) making a total of 7 entries into the Arch table.
<i>DepartDist</i>	The depart distance in millimeters moved at the beginning of the Jump instruction before starting horizontal motion.
<i>ApproDist</i>	The approach distance in millimeters above the target position of the Jump instruction.

**See Also**

Jump, Jump3, Jump3CP

**Arch Example****VB Example:**

```
Sub SetArchs ()
    With m_spel
        .Arch(1, 30, 30)
        .Arch(2, 60, 60)
        .Jump("P1 C1")
        .Jump("P2 C2")
    End With
End Sub
```

**C# Example:**

```
void SetArchs ()
{
    m_spel.Arch(1, 30, 30);
    m_spel.Arch(2, 60, 60);
    m_spel.Jump("P1 C1");
    m_spel.Jump("P2 C2");
}
```

Arm Method, Spel Class

**Description**

Selects the current robot arm.

**Syntax**

Sub **Arm** (*ArmNumber* As Integer)

**Parameters**

*ArmNumber* Integer expression from 0-15. The user may select up to 16 different arms. Arm 0 is the standard (default) robot arm. Arm(s) 1-15 are auxiliary arms defined by the ArmSet instruction.

**See Also**

ArmSet, GetArm, Tool

**Arm Example**

**VB Example:**

```
m_spel.Arm(1)
```

**C# Example:**

```
m_spel.Arm(1);
```

---

**ArmClr Method, Spel Class****Description**

Clears (undefines) an arm for the current robot.

**Syntax**

Sub **ArmClr** (*ArmNumber* As Integer)

**Parameters**

*ArmNumber* Integer expression from 1-15. Arm 0 is the standard (default) robot arm and cannot be cleared. Arm(s) 1-15 are auxiliary arms defined by the ArmSet instruction.

**See Also**

ArmSet, GetArm, Tool

**ArmClr Example****VB Example:**

```
m_spel.ArmClr(1)
```

**C# Example:**

```
m_spel.ArmClr(1);
```

ArmDef Method, Spel Class

**Description**

Returns whether a robot arm is defined or not.

**Syntax**

Function **ArmDef** (*ArmNumber* As Integer) As Boolean

**Parameters**

*ArmNumber* Integer expression from 1-15. Arm 0 is the standard (default) robot arm and is always defined. Arm(s) 1-15 are auxiliary arms defined by using the ArmSet method.

**Return Value**

True if the specified arm is defined, False if not.

**See Also**

ArmSet, GetArm, Tool

**ArmDef Example**

**VB Example:**

```
x = m_spel.ArmDef(1)
```

**C# Example:**

```
x = m_spel.ArmDef(1);
```

## ArmSet Method, Spel Class

**Description**

Defines an auxiliary robot arm.

**Syntax**

Sub **ArmSet** ( *ArmNumber* As Integer, *Param1* As Single, *Param2* As Single, *Param3* As Single, *Param4* As Single, *Param5* As Single )

**Parameters**

<i>ArmNumber</i>	Integer number: Valid range from 1-15.
<i>Param1</i>	(For SCARA Robots) The horizontal distance from the center line of the elbow joint to the center line of the new orientation axis. (I.E. the position where the new auxiliary arm's orientation axis center line is located.) (For Cartesian Robots) X axis direction position offset from the original X position specified in mm.
<i>Param2</i>	(For SCARA Robots) The offset (in degrees) between the line formed between the normal Elbow center line and the normal orientation Axis center line and the line formed between the new auxiliary arm elbow center line and the new orientation axis center line. (These 2 lines should intersect at the elbow center line and the angle formed is the <i>Param2</i> .) (For Cartesian Robots) Y axis direction position offset from the original Y position specified in mm.
<i>Param3</i>	(For SCARA & Cartesian Robots) The Z height offset difference between the new orientation axis center and the old orientation axis center. (This is a distance.)
<i>Param4</i>	(For SCARA Robots) The distance from the shoulder center line to the elbow center line of the elbow orientation of the new auxiliary axis. (For Cartesian Robots) This is a dummy parameter (Specify 0)
<i>Param5</i>	(For SCARA & Cartesian Robots) The angular offset (in degrees) for the new orientation axis vs. the old orientation axis.

**See Also**

Arm, Tool, TLSet

**ArmSet Example****VB Example:**

```
Sub SetArms()
    With m_spel
        .ArmSet(1, 1.5, 0, 0, 0, 0)
        .ArmSet(2, 3.2, 0, 0, 0, 0)
    End With
End Sub
```

**C# Example:**

```
void SetArms()
{
    m_spel.ArmSet(1, 1.5, 0, 0, 0, 0);
    m_spel.ArmSet(2, 3.2, 0, 0, 0, 0);
}
```

Atan Method, Spel Class

**Description**

Returns the arc tangent of a numeric expression.

**Syntax**

Function **Atan** (*number* As Double) As Double

**Parameters**

*number* Numeric expression representing the tangent of an angular value.

**Return Value**

Arc tangent of the specified value

**See Also**

Atan2

**Atan Example**

**VB Example:**

```
Dim angle As Double
```

```
angle = m_spel.Atan(.7)
```

**C# Example:**

```
double angle;
```

```
angle = m_spel.Atan(.7);
```



---

**Atan2 Method, Spel Class****Description**

Returns the angle of the imaginary line connecting points (0,0) and (X, Y) in radians.

**Syntax**

Function **Atan2** (*Dx* As Double, *Dy* as Double) As Double

**Parameters**

*Dx* Numeric expression representing the X coordinate.

*Dy* Numeric expression representing the Y coordinate.

**Return value**

A double value containing the angle.

**See Also**

Atan

**Atan2 Example****VB Example:**

```
Dim angle As Double
```

```
angle = m_spel.Atan2(-25, 50)
```

**C# Example:**

```
double angle;
```

```
angle = m_spel.Atan2(-25, 50);
```

**ATCLR Method, Spel Class**

**Description**

Clears and initializes the average torque for one or more joints.

**Syntax**

Sub **ATCLR** ()

**See Also**

ATRQ, PTCLR, PTRQ

**ATCLR Example**

**VB Example:**

```
m_spel.ATCLR ()
```

**C# Example:**

```
m_spel.ATCLR () ;
```

**AtHome Method, Spel Class****Description**

Returns True if the current robot is at the home position.

**Syntax**

Function **AtHome** () As Boolean

**Return Value**

True if the current robot is at it's home position, False if not.

**See Also**

Home

**AtHome Example****VB Example:**

```
If m_spel.AtHome () Then
    lblCurPos.Text = "Robot is at home position"
Else
    lblCurPos.Text = "Robot is not at home position"
End If
```

**C# Example:**

```
if(m_spel.AtHome ())
    lblCurPos.Text = "Robot is at home position";
else
    lblCurPos.Text = "Robot is not at home position";
```

ATRQ Method, Spel Class

**Description**

Returns the average torque for the specified joint.

**Syntax**

Function **ATRQ** (*JointNumber* As Integer) As Single

**Parameters**

*JointNumber* Integer expression from 1-9 representing the joint number.

**Return Value**

Real value from 0 to 1.

**See Also**

ATCLR, PTCLR, PTRQ

**ATRQ Example**

**VB Example:**

```
Dim avgTorque As Single
Dim i As Integer
For i = 1 To 4
    avgTorque = m_spel.ATRQ(i)
Next i
```

**C# Example:**

```
float avgTorque;
for(int i = 1; i <= 4; i++)
    avgTorque = m_spel.ATRQ(i);
```

## AvgSpeed Method, Spel Class

**Description**

Returns the average value of the absolute speed values for the specified joints.

**Syntax**

Function **AvgSpeed** (*JointNumber* As Integer) As Single

**Parameters**

*JointNumber* Integer expression from 1-9 representing the joint number.

**Return Value**

Real value from 0 to 1.

**Remarks**

This method returns the average value of the absolute speed values for the specified joints. The loading state of the motor can be obtained by this method. The result is a real value from 0 to 1 with 1 being the maximum average speed value.

You must execute this method before AvgSpeedClear method is executed.

This method is time restricted. You must execute this method within 60 seconds after AvgSpeedClear method is executed. When this time is exceeded, error 4088 occurs.

When using the virtual controller or conducting dry-run, the average of the absolute speed values is calculated from the commanded speed instead of the actual speed. This method does not support the PG additional axes.

**See Also**

AvgSpeedClear, PeakSpeed, PeakSpeedClear

**AvgSpeed Example****VB Example:**

```
Dim avgSpeed As Single
Dim i As Integer
For i = 1 To 4
    avgSpeed = m_spel.AvgSpeed(i)
Next i
```

**C# Example:**

```
float avgSpeed;
for(int i = 0; i <=4; i++)
    avgSpeed = m_spel.AvgSpeed(i);
```

**AvgSpeedClear Method, Spel Class**

**Description**

Clears and initializes the average of the absolute speed values for one or more joints.

**Syntax**

Sub **AvgSpeedClear** ()

**Remarks**

This method clears the average of the absolute speed values for the specified joints.

You must execute this method before executing AvgSpeed method.

This method does not support the PG additional axes.

**See Also**

AvgSpeed, PeakSpeed, PeakSpeedClear

**AvgSpeedClear Example**

**VB Example:**

```
m_spel.AvgSpeedClear ()
```

**C# Example:**

```
m_spel.AvgSpeedClear ();
```

## AxisLocked Method, Spel Class

**Description**

Returns True when specified axis release free joint state.

**Syntax**

Function **AxisLocked** (*AxisNumber* As Integer) As Boolean

**Parameters**

*AxisNumber* Numeric expression representing the axis number. The value can be from 1 - 9.

**Return Value**

True if the specified axis is under servo control.

**See Also**

SLock, SFree

**AxisLocked Example****VB Example:**

```
If m_spel.AxisLocked(1) Then
    lblAxis1.Text = "Robot axis #1 is locked"
Else
    lblAxis1.Text = "Robot axis #1 is free"
End If
```

**C# Example:**

```
if (m_spel.AxisLocked(1))
    lblAxis1.Text = "Robot axis #1 is locked";
else
    lblAxis1.Text = "Robot axis #1 is free";
```

**Base Method, Spel Class****Description**

Defines the base coordinate system.

**Syntax**

```
Sub Base (OriginPoint As SpelPoint [, XAxisPoint As SpelPoint] [, YAxisPoint As  
SpelPoint]  
[, Alignment As SpelBaseAlignment] )
```

**Parameters**

<i>OriginPoint</i>	A SpelPoint representing the origin of the base coordinate system.
<i>XAxisPoint</i>	Optional. A SpelPoint located anywhere on the X axis of the base coordinate system.
<i>YAxisPoint</i>	Optional. A SpelPoint located anywhere on the Y axis of the base coordinate system.
<i>Alignment</i>	Optional. When supplying the <i>XAxisPoint</i> and <i>YAxisPoint</i> parameters, use the Alignment parameter to specify which axis to align the base with.

**See Also**

Local

**Base Example****VB Example:**

```
Dim originPoint As New SpelPoint  
originPoint.X = 50  
originPoint.Y = 50  
m_spel.Base(originPoint)
```

**C# Example:**

```
SpelPoint originPoint = new SpelPoint();  
originPoint.X = 50;  
originPoint.Y = 50;  
m_spel.Base(originPoint);
```



## BGo Method, Spel Class

**Description**

Executes Point to Point relative motion in the selected local coordinate system.

**Syntax**

Sub **BGo** (*PointNumber* As Integer)

Sub **BGo** (*Point* As SpelPoint)

Sub **BGo** (*Point* As SpelPoint, *AttribExpr* As String)

Sub **BGo** (*PointExpr* As String)

**Parameters**

Each syntax has one parameter that specifies the end point which the arm travels to during the BGo motion. This is the final position at the end of the point to point motion.

*PointNumber* Specifies the end point by using the point number for a previously taught point in the Controller's point memory for the current robot.

*Point* Specifies the end point by using a SpelPoint data type.

*AttribExpr* Specifies the end point attributes by using a string expression.

*PointExpr* Specifies the end point by using a string expression.

**See Also**

Accel, Speed

Arc, Arc3, CVMove, Go, Jump, Jump3, Jump3CP, Move

BMove, TGo, TMove

CP, Till

**BGo Example****VB Example:**

' Using a point number

```
m_spel.Tool(1)
```

```
m_spel.BGo(100)
```

' Using a SpelPoint

```
Dim pt As SpelPoint
```

```
pt = m_spel.GetPoint("P*")
```

```
pt.X = 125.5
```

```
m_spel.BGo(pt)
```

' Using an attribute expression

```
m_spel.BGo(pt, "ROT")
```

' Using a point expression

```
m_spel.BGo("P0 /L /2")
```

```
m_spel.BGo("P1 :Z(-20)")
```

' Using a parallel processing

```
m_spel.BGo("P1 !D50; On 1; D90; Off 1!")
```

' Using point label

```
m_spel.BGo("pick")
```

### C# Example:

```
// Using a point number
m_spel.Tool(1);
m_spel.BGo(100);

// Using a SpelPoint
SpelPoint pt;
pt = m_spel.GetPoint("P*");
pt.X = 125.5;
m_spel.BGo(pt);

// Using an attribute expression
m_spel.BGo(pt, "ROT");

// Using a point expression
m_spel.BGo("P0 /L /2");
m_spel.BGo("P1 :Z(-20)");

// Using a parallel processing
m_spel.BGo("P1 !D50; On 1; D90; Off 1!");

// Using point label
m_spel.BGo("pick");
```

## BMove Method, Spel Class

**Description**

Executes linear interpolated relative motion in the selected local coordinate system

**Syntax**

Sub **BMove** (*PointNumber* As Integer)

Sub **BMove** (*Point* As SpelPoint)

Sub **BMove** (*Point* As SpelPoint, *AttribExpr* As String)

Sub **BMove** (*PointExpr* As String)

**Parameters**

Each syntax has one parameter that specifies the end point which the arm travels to during the BMove motion. This is the final position at the end of the linear interpolated motion.

*PointNumber* Specifies the end point by using the point number for a previously taught point in the Controller's point memory for the current robot.

*Point* Specifies the end point by using a SpelPoint data type.

*AttribExpr* Specifies the end point attributes by using a string expression.

*PointExpr* Specifies the end point by using a string expression.

**See Also**

AccelR, AccelS, SpeedR, SpeedS

Arc, Arc3, CVMove, Go, Jump, Jump3, Jump3CP, Move

BGo, TGo, TMove

CP, Till

**BMove Example****VB Example:**

```
' Using a point number
m_spel.Tool(1)
m_spel.BMove(100)

' Using a SpelPoint
Dim pt As SpelPoint
pt = m_spel.GetPoint("P*")
pt.X = 125.5
m_spel.BMove(pt)

' Using a point expression
m_spel.BMove("P0 /L /2")

' Using a parallel processing
m_spel.BMove("P1 !D50; On 1; D90; Off 1!")

' Using point label
m_spel.BMove("pick")
```

### C# Example:

```
// Using a point number
m_spel.Tool(1);
m_spel.BMove(100);

// Using a SpelPoint
SpelPoint pt;
pt = m_spel.GetPoint("P*");
pt.X = 125.5;
m_spel.BMove(pt);

// Using a point expression
m_spel.BMove("P0 /L /2");

// Using a parallel processing
m_spel.BMove("P1 !D50; On 1; D90; Off 1!");

// Using point label
m_spel.BMove("pick");
```

## Box Method, Spel Class

**Description**

Specifies an approach check area defined within a box.

**Syntax**

Sub **Box** (*AreaNumber* As Integer, *MinX* As Single, *MaxX* As Single, *MinY* As Single, *MaxY* As Single, *MinZ* As Single, *MaxZ* As Single)

Sub **Box** (*AreaNumber* As Integer, *MinX* As Single, *MaxX* As Single, *MinY* As Single, *MaxY* As Single, *MinZ* As Single, *MaxZ* As Single, *PolarityOn* As Boolean)

**Parameters**

<i>AreaNumber</i>	Integer number from 1-15 representing which of the 15 boxes to define.
<i>MinX</i>	The minimum X coordinate position of the approach check area.
<i>MaxX</i>	The maximum X coordinate position of the approach check area.
<i>MinY</i>	The minimum Y coordinate position of the approach check area.
<i>MaxY</i>	The maximum Y coordinate position of the approach check area.
<i>MinZ</i>	The minimum Z coordinate position of the approach check area.
<i>MaxZ</i>	The maximum Z coordinate position of the approach check area.
<i>PolarityOn</i>	Optional. Sets the remote output logic when the corresponding remote output is used. To set I/O output to on when the end effector is in the box area, use True. To set I/O output to off when the end effector is in the box area, use False.

**See Also**

BoxClr, BoxDef, Plane

**Box Example****VB Example:**

```
m_spel.Box(1, -5, 5, -10, 10, -20, 20)
```

**C# Example:**

```
m_spel.Box(1, -5, 5, -10, 10, -20, 20);
```

**BoxClr Method, Spel Class**

**Description**

Clears the definition of a box (approach check area).

**Syntax**

Sub **BoxClr** (*BoxNumber* As Integer)

**Parameters**

*BoxNumber* Integer expression representing the area number from 1 to 15.

**See Also**

Box, BoxDef

**BoxClr Example**

**VB Example:**

```
m_spel.BoxClr(1)
```

**C# Example:**

```
m_spel.BoxClr(1);
```

**BoxDef Method, Spel Class****Description**

Returns whether Box has been defined or not.

**Syntax**

Function **BoxDef** (*BoxNumber* As Integer) As Boolean

**Parameters**

*BoxNumber* Integer expression representing the area number from 1 to 15.

**Return Value**

True if the specified box is defined, False if not.

**See Also**

Box, BoxClr

**BoxDef Example****VB Example:**

```
x = m_spel.BoxDef(1)
```

**C# Example:**

```
x = m_spel.BoxDef(1);
```

## Brake Method, Spel Class

**Description**

Reads or sets brake status for specified joint.

**Syntax**

Sub **Brake** (*JointNumber* As Integer, *State* As Boolean)

Function **Brake** (*JointNumber* As Integer) As Boolean

**Parameters**

*JointNumber* Integer expression from 1-9 representing the joint number.

*State* The keyword On is used to turn the brake on.  
The keyword Off is used to turn the brake off.

**Return Value**

0 = Brake Off

1 = Brake On

**Remarks**

This method is used to turn brakes on or off for one joint of the 6-axis robot (including N series). This method is intended for use by maintenance personnel only.

When this method is executed, the robot control parameter is initialized.



- Use extreme caution when turning off a brake. Ensure that the joint is properly supported, otherwise the joint can fall and cause damage to the robot and personnel.

**Note****Before releasing the brake, be ready to use the emergency stop switch**

When the controller is in emergency stop status, the motor brakes are locked. Be aware that the robot arm may fall by its own weight when the brake is turned off with Brake command.

**See Also**

Reset, SFree, SLock

**Brake Example****VB Example:**

```
Dim state As Boolean
state = m_spel.Brake(1)
```

**C# Example:**

```
bool state;
state = m_spel.Brake(1);
```



**BTst Method, Spel Class****Description**

Returns the status of 1 bit in a number.

**Syntax**

Function **BTst** (*Number* As Integer, *BitNumber* As Integer) As Boolean

**Parameters**

*Number* Specifies the number for the bit test with an expression or numeric value.

*BitNumber* Specifies the bit (integer from 0 to 31) to be tested.

**Return Value**

True if the specified bit is set, False if not.

**See Also**

On, Off

**BTst Example****VB Example:**

```
x = m_spel.BTst(data, 2)
```

**C# Example:**

```
x = m_spel.BTst(data, 2);
```

**BuildProject Method, Spel Class**

**Description**

Builds the EPSON RC+ 7.0 project specified by the Project property.

**Syntax**

Sub **BuildProject** ()

**See Also**

Project, ProjectBuildComplete, ProjectOverwriteWarningEnabled

**BuildProject Example**

**VB Example:**

```
With m_spel
    .Project = "c:\EpsonRC70\projects\myproj\myproj.sprj"
    If Not .ProjectBuildComplete() Then
        .BuildProject()
    End If
End With
```

**C# Example:**

```
m_spel.Project =
@"c:\EpsonRC70\projects\myproj\myproj.sprj";
if(!m_spel.ProjectBuildComplete())
    m_spel.BuildProject();
```

---

**Call Method, Spel Class**

---

**Description**

Calls (executes) a SPEL<sup>+</sup> function which can optionally return a value.

**Syntax**

Function **Call** (*FuncName* As String [, *Parameters* As String]) As Object

**Parameters**

*FuncName*      The name of a function which has already been defined in the current project.

*Parameters*    Optional.  
Specify a list of arguments. Use arguments separated by commas (.).

**Return Value**

The return value of the SPEL<sup>+</sup> function. The data type matches the data type of the function.

**Remarks**

Use the Call method to call a SPEL<sup>+</sup> function and retrieve the return value. When assigning the result of Call to a variable, ensure that the correct data type is used, otherwise a type mismatch error will occur.

You can also call DLL functions declared in your SPEL<sup>+</sup> code from your Visual Basic application.

**Note**

---

**The function executed by Call method cannot be stopped/paused by Stop, Pause, Halt, or Quit method.**

If you need to stop or pause, use Xqt Method.

---

**See Also**

Xqt

### Call Example

' Visual Basic Code

```
Dim errCode As Integer  
errCode = m_spel.Call("GetPart", ""Test"",2")
```

// C# Code

```
int errCode;  
errCode = m_spel.Call("GetPart", ""Test"",2");
```

' SPEL+ function

```
Function GetPart(Info$ As String, Timeout As Integer) As  
Integer  
    Long errNum  
    OnErr GoTo GPErr  
    Print Info$  
    errNum = 0  
    Jump P1  
    On vacuum  
    Wait Sw(vacOn) = On, Timeout  
    If TW = True Then  
        errNum = VAC_TIMEOUT  
    EndIf  
    GetPart = errNum  
    Exit Function  
GPErr:  
    GetPart = Err  
Fend
```

## CalPIs Method, Spel Class

**Description**

Reads or sets the position and orientation pulse values for calibration.

**Syntax**

Function CalPIs (JointNumber As Integer) As Integer

Sub CalPIs (J1Pulses As Integer, J2Pulses As Integer, J3Pulses As Integer, J4Pulses As Integer, [J5Pulses As Integer], [J6Pulses As Integer], [J7Pulses As Integer], [J8Pulses As Integer], [J9Pulses As Integer])

**Parameters**

*J1Pulses – J9Pulses* Integer number representing the pulse count of J1 to J9. J5Pulses to J9Pulses can be omitted.

**Return Value**

When parameters are omitted, displays the current CalPIs values.

**Remarks**

Specifies and maintains the correct position pulse value(s) for calibration.

CalPIs is intended to be used for maintenance, such as after changing motors or when motor zero position needs to be matched to the corresponding arm mechanical zero position. This matching of motor zero position to corresponding arm mechanical zero position is called calibration.

Normally, the calibration position Pulse values match the CalPIs pulse values. However, after performing maintenance operations such as changing motors, these two sets of values no longer match, and therefore calibration becomes necessary.

Calibration may be accomplished by moving the arm to a certain calibration position and then executing Calib. By executing Calib, the calibration position pulse value is changed to the CalPIs value (the correct pulse value for the calibration position.)

Hofs values must be determined to execute calibration. To have Hofs values automatically calculated, move the arm to the desired calibration position, and execute Calib. The controller automatically.

**Note****CalPIs Values Cannot be Changed by cycling power**

CalPIs values are not initialized by turning main power to the controller off and then on again. The only method to modify the CalPIs values is to execute the Calib command.

**See Also**

Hofs

### **CalPls Example**

#### **VB Example:**

```
Dim val As Single
Dim i As Integer
For i = 1 To 4
    val = m_spel.CalPls(i)
Next i
```

#### **C# Example:**

```
float val;
for(int i = 1; i <= 4; i++)
    val = m_spel.CalPls(i);
```

---

**ClearPoints Method, Spel Class****Description**

Clears the points in memory for the current robot.

**Syntax**

Sub **ClearPoints** ()

**See Also**

LoadPoints, Robot, SavePoints, SetPoint

**ClearPoints Example****VB Example:**

```
With m_spel
    .ClearPoints ()
    .SetPoint(1, 100, 200, -20, 0, 0, 0)
    .Jump(1)
End With
```

**C# Example:**

```
m_spel.ClearPoints ();
m_spel.SetPoint(1, 100, 200, -20, 0, 0, 0);
m_spel.Jump(1);
```

## Connect Method, Spel Class

**Description**

Connects the Spel class instance with a Controller.

**Syntax**

Sub **Connect** (ConnectionName As String)

Sub **Connect** (ConnectionName As String, ConnectionPassword As String)

Sub **Connect** (*ConnectionNumber* As Integer)

Sub **Connect** (*ConnectionNumber* As Integer, ConnectionPassword As String)

**Parameters**

*ConnectionName* String expression for the connection name.

*ConnectionNumber* Integer expression for the connection number. The connection number is the number of the connection in the RC+ Setup | PC to Controller Communications dialog. Specify a value of -1 to use the last successful connection.

*ConnectionPassword* String expression for the connection password. If the controller has a password for its connection and the password was not configured in the RC+ Setup | PC to Controller Communications dialog, then you must specify a password in order to connect with the controller.

**Remarks**

When a Spel class instance needs to communicate with the Controller, it automatically connects. If you want to explicitly connect to the Controller, use the Connect method.

**See Also**

Disconnect, Initialize

**Connect Example****VB Example:**

```
Try
    m_spel.Connect (1)
Catch ex As RCAPINet.SpelException
    MsgBox (ex.Message)
End Try
```

**C# Example:**

```
try{
    m_spel.Connect (1);
}
catch(RCAPINet.SpelException ex){
    MessageBox.Show(ex.Message);
}
```



## Continue Method, Spel Class

**Description**

Causes all tasks in the Controller to resume if a pause has occurred.

**Syntax**

Sub **Continue** ()

**Remarks**

Use **Continue** to resume all tasks that have been paused by the Pause method or by safeguard open.

When the safeguard is open while tasks are running, the robot will decelerate to a stop and the robot motors will be turned off. After the safeguard has been closed, you can use **Continue** to resume the cycle.

**See Also**

Pause, Start, Stop

**Continue Example****VB Example:**

```
Sub btnContinue_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnContinue.Click

    btnPause.Enabled = True
    btnContinue.Enabled = False

    Try
        m_spel.Continue()
    Catch ex As RCAPINet.SpelException
        MsgBox(ex.Message)
    End Try
End Sub
```

**C# Example:**

```
void btnContinue_Click(object sender, EventArgs e)
{
    btnPause.Enabled = true;
    btnContinue.Enabled = true;

    try{
        m_spel.Continue ();
    }
    catch(RCAPINet.SpelException ex){
        MessageBox.Show(ex.Message);
    }
}
```

Ctr Method, Spel Class

**Description**

Returns the counter value of the specified input counter.

**Syntax**

Function **Ctr** (*BitNumber* As Integer) As Integer

**Parameters**

*BitNumber* Number of the input bit set as a counter.  
Only 16 counters can be active at the same time.

**Return Value**

Returns the counter value.( integer from 0 to 65535)

**See Also**

CtReset

**Ctr Example**

**VB Example:**

```
lblCounter.Text = m_spel.Ctr(1).ToString()
```

**C# Example:**

```
lblCounter.Text = m_spel.Ctr(1).ToString();
```

**CtReset Method, Spel Class****Description**

Resets the counter value of the specified input counter. Also defines the input as a counter Input.

**Syntax**

Sub **CtReset** (*BitNumber* As Integer)

**Parameters**

*BitNumber* Number of the input bit set as a counter.  
Only 16 counters can be active at the same time.

**See Also**

Ctr

**CtReset Example****VB Example:**

```
m_spel.CtReset(2)
```

**C# Example:**

```
m_spel.CtReset(2);
```

## Curve Method, Spel Class

**Description**

Defines the data and points required to move the arm along a curved path. Many data points can be defined in the path to improve precision of the path.

**Syntax**

Sub **Curve** (*FileName* As String, *Closure* As Boolean, *Mode* As Integer, *NumOfAxis* As Integer, *PointList* As String)

**Parameters**

*FileName* A string expression for the path and name of the file in which the point data is stored. The specified *fileName* will have the extension CRV appended to the end so no extension is to be specified by the user. When the **Curve** instruction is executed, *fileName* will be created.

*Closure* A Boolean expression that specifies whether to connect the last point of the path to the first point.

*Mode* Specifies whether or not the arm is automatically interpolated in the tangential direction of the U-Axis.

Mode Setting	Tangential Correction
0	No
2	Yes

*NumOfAxis* Integer expression between 2 - 4 which specifies the number of Axes controlled during the curved motion as follows:  
 2: Generate a curve in the XY plane with no Z-Axis movement or U-Axis rotation.  
 3: Generate a curve in the XYZ plane with no U-Axis rotation. (Theta 1, Theta2, and Z)  
 4: Generate a curve in the XYZ plane with U-Axis rotation. (Controls all 4 Axes)

*PointList* { point expression | P(*start:finish*) } [, *output command*] ...  
 This parameter is actually a series of Point Numbers and optional output statements either separated by commas or an ascended range of points separated by a colon.

Normally the series of points are separated by commas as shown below:

```
Curve MyFile, 0, 0, 4, P1, P2, P3, P4
```

Or use a colon to specify as shown below:

```
Curve MyFile, 0, 0, 4, P(1:4)
```

**Remarks**

Use Curve to define a spline path to be executed with the CVMove method. See the SPEL+ command Curve for more details.

**See Also**

Curve (SPEL+ Statement), CVMove Method

**Curve Example****VB Example:**

```
m_spel.Curve("mycurveFile", True, 0, 4, "P(1:3), On 1, P(4:7) ")
m_spel.CVMove("mycurveFile")
```

**C# Example:**

```
m_spel.Curve("mycurveFile", True, 0, 4, "P(1:3), On 1, P(4:7) ");
m_spel.CVMove("mycurveFile");
```

## CVMove Method, Spel Class

**Description**

Performs the continuous spline path motion defined by the **Curve** instruction.

**Syntax**

Sub **CVMove** (*FileName* As String [, *OptionList* As String])

**Parameters**

*FileName* String expression for the path and name of the file to use for the continuous path motion data. This file must be previously created by the Curve instruction.

*OptionList* Optional. String expression containing Till specification.

**Remarks**

Use CVMove to execute a path defined with the Curve method. See the SPEL<sup>+</sup> command **CVMove** for more details.

If you need to execute CVMove with CP, it is recommended that you execute CVMove from a SPEL<sup>+</sup> task rather than from your application. The reason for this is that for CP motion to perform properly, the system needs to know ahead of time where the next motion target is. Since the RC<sup>+</sup> API commands are executed one at a time, the system does not know ahead of time where the next target is.

**See Also**

Curve, CVMove (SPEL<sup>+</sup> Command)

**CVMove Example****VB Example:**

```
m_spel.Curve("mycurveFile", True, 0, 4, "P(1:3), On 1, P(4:7)")
m_spel.CVMove("mycurveFile", "CP Till Sw(1) = 1")
m_spel.CVMove("mycurveFile")
```

**C# Example:**

```
m_spel.Curve("mycurveFile", True, 0, 4, "P(1:3), On 1, P(4:7)");
m_spel.CVMove("mycurveFile", "CP Till Sw(1) = 1");
m_spel.CVMove("mycurveFile");
```

## CX, CY, CZ, CU, CV, CW, CR, CS, CT Methods, Spel Class

**Description**

Retrieves a coordinate value from a point

CV and CW are for the 6-axis robot  
 CS and CT are for the additional axis  
 CR is for the Joint 7-axis robot

**Syntax**

Function **CX** (*PointExpr* As String) As Single  
 Function **CY** (*PointExpr* As String) As Single  
 Function **CZ** (*PointExpr* As String) As Single  
 Function **CU** (*PointExpr* As String) As Single  
 Function **CV** (*PointExpr* As String) As Single  
 Function **CW** (*PointExpr* As String) As Single  
 Function **CR** (*PointExpr* As String) As Single  
 Function **CS** (*PointExpr* As String) As Single  
 Function **CT** (*PointExpr* As String) As Single

**Parameters**

*PointExpr* A string expression specifying the point from which to retrieve the specified coordinate. Any valid point expression can be used. P\* can also be used to retrieve the coordinate from the current position.

**Return Value**

The specified coordinate value.  
 Return value of CX, CY, CZ : Real value (mm)  
 Return value of CU, CV, CW : Real value (deg)  
 Return value of CR, CS, CT : Real value

**See Also**

GetPoint, SetPoint

**CX, CY, CZ, CU, CV, CW, CR, CS, CT Example****VB Example:**

```
Dim x As Single, y As Single
x = m_spel.CX("P1")
y = m_spel.CY("P*")
```

**C# Example:**

```
float x, y;
x = m_spel.CX("P1");
y = m_spel.CY("P*");
```

## Delay Method, Spel Class

**Description**

Delays for a specified number of milliseconds.

**Syntax**

Sub **Delay** (*Milliseconds* As Integer)

**Parameters**

*Milliseconds* Integer value containing the number of milliseconds to delay.

**Delay Example****VB Example:**

```
m_spel.Delay(500)
```

**C# Example:**

```
m_spel.Delay(500);
```

DegToRad Method, Spel Class

**Description**

Converts Degrees into Radians.

**Syntax**

Function **DegToRad** (*degrees* As Double) As Double

**Parameters**

*degrees*      The number of degrees to convert into Radians.

**Return value**

A double value containing radians.

**See Also**

RadToDeg

**DegToRad Example**

**VB Example:**

```
Dim rad As Double

rad = m_spel.DegToRad(45)
```

**C# Example:**

```
double rad;

rad = m_spel.DegToRad(45);
```



## Disconnect Method, Spel Class

**Description**

Disconnects the Spel class instance from the current connection.

**Syntax**

Sub **Disconnect** ()

**Remarks**

Use **Disconnect** to disconnect from the current Controller connection.

**See Also**

Connect, Initialize

**Disconnect Example****VB Example:**

```
Try
    m_spel.Disconnect()
Catch ex As RCAPINet.SpelException
    MsgBox(ex.Message)
End Try
```

**C# Example:**

```
try{
    m_spel.Disconnect();
}
catch(RCAPINet.SpelException ex){
    MessageBox.Show(ex.Message);
}
```

ECP Method, Spel Class

**Description**

Selects the current ECP definition.

**Syntax**

Sub **ECP** (*ECPNumber* As Integer)

**Parameters**

*ECPNumber* Integer number from 0-15 representing which of 16 ECP definitions to use with the next motion instructions.

**See Also**

ECPSet

**ECP Example**

**VB Example:**

```
m_spel.ECP(1)
m_spel.Move("P1 ECP")
```

**C# Example:**

```
m_spel.ECP(1);
m_spel.Move("P1 ECP");
```

---

**ECPClr Method, Spel Class****Description**

Clears (undefines) an external control point for the current robot.

**Syntax**

Sub **ECPClr** (*ECPNumber* As Integer)

**Parameters**

*ECPNumber* Integer expression representing which one of the 15 external control points to clear (undefine).  
(ECP 0 is the default and cannot be cleared.)

**See Also**

ECP, ECPDef

**ECPClr Example****VB Example:**

```
m_spel.ECPClr(1)
```

**C# Example:**

```
m_spel.ECPClr(1);
```

ECPDef Method, Spel Class

**Description**

Returns ECP definition status.

**Syntax**

Function **ECPDef** (*ECPNumber* As Integer) As Boolean

**Parameters**

*ECPNumber* Integer value representing which ECP to return status for.

**Return Value**

True if the specified ECP is defined, False if not.

**See Also**

ECP, ECPClr

**ECPDef Example**

**VB Example:**

```
x = m_spel.ECPDef(1)
```

**C# Example:**

```
x = m_spel.ECPDef(1);
```

## ECPSet Method, Spel Class

**Description**

Defines an ECP (external control point).

**Syntax**

Sub **ECPSet** (*ECPNumber* As Integer, *Point* As *SpelPoint*)

Sub **ECPSet** (*ECPNumber* As Integer, *XCoord* as Double, *YCoord* as Double, *ZCoord* as Double, *UCoord* as Double [, *VCoord* As Double] [, *WCoord* as Double])

**Parameters**

<i>ECPNumber</i>	Integer number from 1-15 representing which of 15 external control points to define.
<i>Point</i>	A <i>SpelPoint</i> containing the point data.
<i>XCoord</i>	The external control point X coordinate.
<i>YCoord</i>	The external control point Y coordinate.
<i>ZCoord</i>	The external control point Z coordinate.
<i>UCoord</i>	The external control point U coordinate.
<i>VCoord</i>	Optional. The external control point V coordinate.
<i>WCoord</i>	Optional. The external control point W coordinate.

**See Also**

ArmSet, ECP, GetECP, TLSet

**ECPSet Example****VB Example:**

```
m_spel.ECPSet(1, 100.5, 99.3, 0, 0)
```

**C# Example:**

```
m_spel.ECPSet(1, 100.5, 99.3, 0, 0);
```

EnableEvent Method, Spel Class

**Description**

Enables certain system events for the EventReceived event.

**Syntax**

Sub **EnableEvent** (EventNumber As RCAPINet.SpelEvents, Enabled as Boolean)

**Parameters**

*Event*                The event to enable or disable.

*Enabled*             Set to True to enable the event and False to disable it.

**See Also**

EventReceived

**EnableEvent Example**

**VB Example:**

```
With m_spel
    .EnableEvent (RCAPINet.SpelEvents.ProjectBuildStatus, True)
    .BuildProject ()
End With
```

**C# Example:**

```
m_spel.EnableEvent (RCAPINet.SpelEvents.ProjectBuildStatus,
true);
m_spel.BuildProject ();
```

## ExecuteCommand Method, Spel Class

**Description**

Sends a command to EPSON RC+ 7.0 and waits for it to complete

**Syntax**

Sub **ExecuteCommand** (*Command* As String , [ByRef *Reply* As String])

**Parameters**

*Command*       String containing SPEL<sup>+</sup> command.

*Reply*           Optional reply returned.

**Remarks**

Normally, **ExecuteCommand** is not required. Most operations can be performed by executing Spel methods. However, sometimes it is desirable to execute SPEL<sup>+</sup> multi-statements. Multi-statements are one line commands that contain more than one statement separated by semi-colons. Use **ExecuteCommand** to execute multi-statements.

For example:

```
m_spel.ExecuteCommand("JUMP pick; ON tipvac")
```

The maximum command line length is 200 characters.

**See Also**

Pause

**ExecuteCommand Example****VB Example:**

```
m_spel.ExecuteCommand("JUMP P1!D50; ON 1!")
```

**C# Example:**

```
m_spel.ExecuteCommand("JUMP P1!D50; ON 1!");
```

**FbusIO\_GetBusStatus Method, Spel Class**

**Description**

Returns the status of the specified Fieldbus.

**Syntax**

Function FbusIO\_GetBusStatus (*BusNumber* As Integer) As Integer

**Parameters**

*BusNumber* Integer expression representing the Fieldbus system number.  
This number must be 16. This is the ID for the bus connected to the Fieldbus master board on the PC side of the controller.  
Integer number from 1-15 representing which of 15 external control points to define.

**Return Value**

0 – OK  
1 – Disconnected  
2 – Power off

**Remarks**

**Note**

---

**This method will only work if the Fieldbus Master option is active.**

---

**See Also**

FbusIO\_GetDeviceStatus, FbusIO\_SendMsg, IsOptionActive

**FbusIO\_GetBusStatus Example**

**VB Example:**

```
Dim busStatus As Integer  
busStatus = m_spel.FbusIO_GetBusStatus(16)
```

**C# Example:**

```
int busStatus;  
busStatus = m_spel.FbusIO_GetBusStatus(16);
```



## FbusIO\_GetDeviceStatus Method, Spel Class

**Description**

Returns the status of the specified Fieldbus device.

**Syntax**

Function FbusIO\_GetDeviceStatus (*BusNumber* As Integer, *DeviceID* As Integer) As Integer

**Parameters**

*BusNumber* Integer expression representing the Fieldbus system number. This number must be 16. This is the ID for the bus connected to the Fieldbus master board on the PC side of the controller. Integer number from 1-15 representing which of 15 external control points to define.

*DeviceID* Integer expression representing the Fieldbus ID of the device.

**Return Value**

0 – OK

1 – Disconnected

2 – Power off

3 - Synchronization error. Device is booting, or has incorrect baud rate.

**Remarks****Note**


---

**This method will only work if the Fieldbus Master option is active.**

---

**See Also**

FbusIO\_GetBusStatus, FbusIO\_SendMsg, IsOptionActive

**FbusIO\_GetDeviceStatus Example****VB Example:**

```
Dim deviceStatus As Integer
deviceStatus = m_spel.FbusIO_GetDeviceStatus(16, 10)
```

**C# Example:**

```
int deviceStatus;
devideStatus = m_spel.FbusIO_GetDeviceStatus(16, 10);
```

**FbusIO\_SendMsg Method, Spel Class****Description**

Sends an explicit message to a Fieldbus device and returns the reply.

**Syntax**

Sub FBusIO\_SendMsg (*BusNumber* As Integer, *DeviceID* As Integer, *MsgParam* As Integer, *SendData* As Byte(), *ByRef RecvData* As Byte())

**Parameters**

- BusNumber* Integer expression representing the Fieldbus system number. This number must be 16. This is the ID for the bus connected to the Fieldbus master board on the PC side of the controller. Integer number from 1-15 representing which of 15 external control points to define.
- DeviceID* Integer expression representing the Fieldbus ID of the device.
- MsgParam* Integer expression representing the message parameter. This is not available for DeviceNet.
- SendData* Array of type Byte containing data that is sent to the device. This array must be dimensioned to the number of bytes to send. If there are no bytes to send, specify 0.
- RecvData* Array of type Byte that contains the data received from the device. This array will automatically be redimensioned to the number of bytes received.

**Remarks****Note**

**This method will only work if the Fieldbus Master option is active.**

**See Also**

FbusIO\_GetBusStatus, FbusIO\_GetDeviceStatus, IsOptionActive

**FbusIO\_SendMsg Examples****VB Examples:**

```
' Send explicit message to DeviceNet device
Dim recvData() as Byte
Dim sendData(6) as Byte
Array.Clear(sendData, 0, sendData.Length)
sendData(0) = 14 'Command
sendData(1) = 1 'Class
sendData(3) = 1 'Instance
sendData(5) = 7 'Attribute
' MsgParam is 0 for DeviceNet
m_spel.FbusIO_SendMsg(16, 1, 0, sendData, recvData)

' Send explicit message to Profibus device
Dim recvData() As Byte;
m_spel.FbusIO_SendMsg(16, 1, 56, Nothing, recvData);
```

**C# Examples:**

```
// Send explicit message to DeviceNet device
byte[] sendData, recvData;
byte[] sendData = new byte[6];
Array.Clear(sendData, 0, sendData.Length);
sendData[0] = 14; //Command
sendData[1] = 1; //Class
sendData[3] = 1; //Instance
sendData[5] = 7; //Attribute
//MsgParam is 0 for DeviceNet
m_spel.FbusIO_SendMsg(16, 201, 0, sendData, out recvData);

// Send explicit message to Profibus device
byte[] recvData;
m_spel.FbusIO_SendMsg(16, 1, 56, null, out recvData);
```

## FGGet Method, Spel Class

**Description**

Acquires a result of a Force Guide sequence or Force Guide object.

**Syntax**

Sub FGGet (*Sequence* As String, [*Object* As String], *Property* As SpelForceProps, ByRef *Result* As Boolean)

Sub FGGet (*Sequence* As String, [*Object* As String], *Property* As SpelForceProps, ByRef *Result* As Double)

Sub FGGet (*Sequence* As String, [*Object* As String], *Property* As SpelForceProps, ByRef *Result* As Integer)

Sub FGGet (*Sequence* As String, [*Object* As String], *Property* As SpelForceProps, ByRef *Result* As String)

**Parameters**

*Sequence* Force guide sequence name or string variable representing Force Guide sequence name

*Object* Force guide object name or string variable representing Force Guide object name.  
Omitted when a result of a Force Guide sequence is acquired.

*Property* Name of result to acquire a value

*Result* Variable that shows a returned value  
The number and types vary according to results.

**See Also**

FGRun

**FGGet Example****VB Example:**

```
Dim errCode As Integer
m_spel.MotorsOn = True

m_spel.FGRun("Sequence1")
m_spel.FGGet("Sequence1","", SpelForceProps.EndStatus, val)
```

**C# Example:**

```
int errCode;
m_spel.MotorsOn = true;

m_spel.FGRun("Sequence1");
m_spel.FGGet("Sequence1","", SpelForceProps.EndStatus, val);
```

## FGRUN Method, Spel Class

**Description**

Executes a Force Guide sequence.

**Syntax**

Sub FGRUN (*Sequence As String*)

**Parameters**

*Sequence* Force guide sequence name or string variable representing Force Guide sequence name

**Remarks**

Executes a specified Force Guide sequence. The Force Guide sequence starts from the position where the FGRUN statement was executed. Execute after moving to the assumed start position by the Go statement, Move statement, or other motion commands.

When the specified Force Guide sequence ends, the program proceeds to the next statement.

To acquire the results of sequences executed by FGRUN, use FGGet.

When path motion is enabled by the CP parameter or CP statement, the program waits until the robot stops and then executes a Force Guide sequence.

When any of the following conditions is fulfilled at the time of execution start, an error occurs.

- The robot specified in the program differs from the robot specified by the RobotNumber property.  
Specify the correct robot by the Robot statement.
- The robot type specified in the program differs from the robot type specified by the RobotType property.  
Specify the correct robot by the Robot statement.
- The tool number specified in the program differs from the tool number specified by the RobotTool property.  
Specify the correct tool number by the Tool statement.
- Motor is in OFF state.  
Switch to ON state by the Motor statement.
- Force control function is currently being executed.  
Stop force control by the FCEnd statement.
- Conveyor tracking is currently being executed.  
Stop conveyor tracking by the Cnv\_AbortTrack statement.
- Currently in the torque control mode.  
Disable the torque control mode by the TC statement.

FGRUN, when executed, automatically overwrites the following properties; therefore, it cannot be used together with the following properties:

FM object

AvgForceClear property

PeakForceClear property

This method cannot be executed while the program is running.

**See Also**

FGGet

### **FGRun Example**

#### **VB Example:**

```
Dim errCode As Integer
m_spel.MotorsOn = True

m_spel.FGRun("Sequence1")
errCode = m_spel.FGGet("Sequence1",
SpelForceProps.EndStatus, val)
```

#### **C# Example:**

```
int errCode;
m_spel.MotorsOn = true;

m_spel.FGRun("Sequence1");
errCode = m_spel.FGGet("Sequence1",
SpelForceProps.EndStatus, val);
```

## Find Method, Spel Class

**Description**

Sets the condition to save the coordinate during motion instructions.

**Syntax**

Sub **Find** (*Condition* As String)

**Parameters**

*Condition* Specifies the input status as a trigger.

**See Also**

Go, Jump

**Find Example****VB Example:**

```
m_spel.Find("Sw(5) = On")
```

**C# Example:**

```
m_spel.Find("Sw(5) = On");
```

Fine Method, Spel Class

**Description**

Specifies and displays the positioning accuracy for target points.

**Syntax**

Sub **Fine** ( *J1MaxErr* As Integer, *J2MaxErr* As Integer, *J3MaxErr* As Integer,  
*J4MaxErr* As Integer , *J5MaxErr* As Integer, *J6MaxErr* As Integer  
[, *J7MaxErr* As Integer] [, *J8MaxErr* As Integer] [, *J9MaxErr* As Integer] )

**Parameters**

*J1MaxErr – J9MaxErr* Integer number ranging from (0-32767) which represents the allowable positioning error for the each joint.  
The values for joints 7, 8, and 9 are optional.

**See Also**

Weight

**Fine Example**

**VB Example:**

```
m_spel.Fine(1000, 1000, 1000, 1000, 0, 0)
```

**C# Example:**

```
m_spel.Fine(1000, 1000, 1000, 1000, 0, 0);
```



---

**Force\_Calibrate Method, Spel Class**

---

**Description**

Sets zero offsets for all axes for the current force sensor.

**Syntax**

Sub **Force\_Calibrate()**

**Remarks**

You should call **Force\_Calibrate** for each sensor when your application starts. This will account for the weight of the components mounted on the sensor.

**Note**

---

**This method is only available when the Force Sensing Option (ATI Force Sensor) is installed.**

**When using an Epson Force Sensor, the FGGet method and FGRun method can be used.**

---

**See Also**

**Force\_Sensor**, **Force\_GetForces**, **Force\_SetTrigger**

**Force\_Calibrate Example****VB Example:**

```
m_spel.ForceSensor = 1  
m_spel.Force_Calibrate()
```

**C# Example:**

```
m_spel.ForceSensor = 1;  
m_spel.Force_Calibrate();
```

Force\_ClearTrigger Method, Spel Class

**Description**

Clears all trigger conditions for the current force sensor.

**Syntax**

Sub Force\_ClearTrigger()

**Remarks**

Use Force\_ClearTrigger to clear all conditions for the current force sensor's trigger.

**Note**

---

**This method is only available when the Force Sensing Option (ATI Force Sensor) is installed.**

**When using an Epson Force Sensor, the FGGet method and FGRun method can be used.**

---

**See Also**

Force\_Sensor, Force\_GetForces, Force\_SetTrigger

**Force\_ClearTrigger Example**

**VB Example:**

```
m_spel.ForceSensor = 1  
m_spel.Force_ClearTrigger()
```

**C# Example:**

```
m_spel.ForceSensor = 1;  
m_spel.Force_ClearTrigger();
```

## Force\_GetForce Method, Spel Class

**Description**

Returns the force for a specified force sensor axis.

**Syntax**

Function **Force\_GetForce**( *Axis* As SpelForceAxis) As Single

**Parameters**

*Axis*                    The axis value to retrieve, as shown below:

SpelForceAxis	Value
XForce	1
YForce	2
ZForce	3
XTorque	4
YTorque	5
ZTorque	6

**Remarks**

Use Force\_GetForce to read the current force setting for one axis. The units are determined by the force sensor configuration.

**Note**

**This method is only available when the Force Sensing Option (ATI Force Sensor) is installed.**

**When using an Epson Force Sensor, the FGGet method and FGRun method can be used.**

**See Also**

Force\_Sensor, Force\_GetForces, Force\_SetTrigger

**Force\_GetForce Example****VB Example:**

```
m_spel.ForceSensor = 1
zForce = m_spel.Force_GetForce (SpelForceAxis.ZForce)
```

**C# Example:**

```
m_spel.ForceSensor = 1;
zForce = m_spel.Force_GetForce (SpelForceAxis.ZForce);
```

Force\_GetForces Method, Spel Class

**Description**

Returns the forces and torques for all force sensor axes in an array.

**Syntax**

Sub **Force\_GetForces**( *Values*() As Single)

**Parameters**

*Values*                Single array that will be returned with 6 elements.

**Remarks**

Use Force\_GetForces to read all force and torque values at once.

**Note**

---

**This method is only available when the Force Sensing Option (ATI Force Sensor) is installed.**

**When using an Epson Force Sensor, the FGGet method and FGRun method can be used.**

---

**See Also**

Force\_Sensor, Force\_GetForces, Force\_SetTrigger

**Force\_GetForces Example**

**VB Example:**

```
Dim values() as Single = Nothing
m_spel.ForceSensor = 1
m_spel.Force_GetForces(values)
```

**C# Example:**

```
float[] values = new float[6];

m_spel.ForceSensor(1);
m_spel.Force_GetForces(values);
```

## Force\_SetTrigger Method, Spel Class

**Description**

Sets the force trigger for the Till command.

**Syntax**

Sub **Force\_SetTrigger**( *Axis* As SpelForceAxis, *Threshold* As Single, *CompareType* As SpelForceCompareType )

**Parameters**

*Axis* The axis to use for the trigger, as shown below:

SpelForceAxis	Value
XForce	1
YForce	2
ZForce	3
XTorque	4
YTorque	5
ZTorque	6

*Threshold* Single expression representing the threshold value.

*CompareType* LessOrEqual, or GreaterOrEqual.

**Remarks**

To stop motion with a force sensor, you must set the trigger for the sensor, then use Till Force in your motion statement.

You can set the trigger with multiple axes. Call Force\_SetTrigger for each axis.

To clear all trigger conditions, use Force\_ClearTrigger.

**Note**

**This method is only available when the Force Sensing Option (ATI Force Sensor) is installed.**

**When using an Epson Force Sensor, the FGGet method and FGRun method can be used.**

**See Also**

Force\_ClearTrigger, Force\_Sensor, Till

**Force\_SetTrigger Example****VB Example:**

```
m_spel.ForceSensor = 1
m_spel.Force_SetTrigger(SpelForceAxis.ZForce, -2.0, _
    SpelForceCompareType.GreaterOrEqual)
m_spel.Till("Force")
m_spel.Move("P1 Till")
```

**C# Example:**

```
m_spel.ForceSensor = 1;
m_spel.Force_SetTrigger(SpelForceAxis.ZForce, -2.0,
    SpelForceCompareType.GreaterOrEqual);
m_spel.Till("Force");
m_spel.Move("P1 Till");
```

**GetAccel Method, Spel Class****Description**

Returns specified acceleration/deceleration value.

**Syntax**

Function **GetAccel** (*ParamNumber* As Integer) As Integer

**Parameters**

*ParamNumber* Integer expression which can have the following values:  
1: acceleration specification value  
2: deceleration specification value  
3: depart acceleration specification value for Jump  
4: depart deceleration specification value for Jump  
5: approach acceleration specification value for Jump  
6: approach deceleration specification value for Jump

**Return Value**

Integer containing the specified acceleration/deceleration value.

**See Also**

Accel

**GetAccel Example****VB Example:**

```
Dim x As Integer  
x = m_spel.GetAccel(1)
```

**C# Example:**

```
int x;  
x = m_spel.GetAccel(1);
```

**GetArm Method, Spel Class****Description**

Returns the current Arm number for the current robot.

**Syntax**

Function **GetArm** () As Integer

**Return Value**

Integer containing the current arm number.

**See Also**

Arm, ArmSet, Robot, Tool

**GetArm Example****VB Example:**

```
saveArm = m_spel.GetArm()  
m_spel.Arm(2)
```

**C# Example:**

```
saveArm = m_spel.GetArm();  
m_spel.Arm(2);
```

GetConnectionInfo Method, Spel Class

**Description**

Returns information about the Controller connections.

**Syntax**

Function **GetConnectionInfo()** As SpelConnectionInfo()

**Return Value**

An array of SpelConnectionInfo.

**See Also**

GetControllerInfo

**Remarks**

**GetConnectionInfo** returns an array of SpelConnectionInfo. The connection information is configured in EPSON RC+ from the [Setup]-[PC to Controller Communication] dialog.

**GetConnectionInfo Example**

**VB Example:**

```
Dim info() As SpelConnectionInfo
```

```
info = m_spel.GetConnectionInfo()
```

**C# Example:**

```
SpelConnectionInfo[] info = m_spel.GetConnectionInfo();
```



## GetControllerInfo Method, Spel Class

**Description**

Returns information about the current Controller.

**Syntax**

Function **GetControllerInfo()** As SpelControllerInfo

**Return Value**

A SpelControllerInfo instance.

**See Also**

GetErrorMessage, GetRobotInfo, GetTaskInfo

**Remarks**

**GetControllerInfo** returns a new instance of the SpelControllerInfo class, which contains Controller information properties.

**GetControllerInfo Example****VB Example:**

```
Dim info As SpelControllerInfo
Dim msg As String

info = m_spel.GetControllerInfo()
msg = "Project Name: " & info.ProjectName & vbCrLf _
    & "Project ID: " & info.ProjectID
MsgBox(msg)
```

**C# Example:**

```
SpelControllerInfo info;
string msg;

info = m_spel.GetControllerInfo();
msg = "Project Name:" + info.ProjectName + "\r\n"
    + "ProjectID :" +
    "info.ProjectID";
MessageBox.Show(msg);
```

**GetCurrentConnectionInfo Method, Spel Class**

**Description**

Returns the information of the current controller connection.

**Syntax**

Function **GetCurrentConnectionInfo** () As SpelConnectionInfo

**Return Value**

SpelConnectionInfo

**See Also**

GetConnectionInfo, GetControllerInfo

**GetCurrentConnectionInfo Example**

**VB Example:**

```
Dim info As SpelConnectionInfo  
  
info = m_spel.GetCurrentConnectionInfo()
```

**C# Example:**

```
SpelConnectionInfo info;  
  
info = m_spel.GetCurrentConnectionInfo();
```

**GetCurrentUser Method, Spel Class****Description**

Returns the current EPSON RC+ 7.0 user.

**Syntax**

Function **GetCurrentUser** () As String

**Return Value**

String variable containing the current user.

**See Also**

Login

**GetCurrentUser Example****VB Example:**

```
Dim currentUser As String
```

```
currentUser = m_spel.GetCurrentUser()
```

**C# Example:**

```
string currentUser;
```

```
currentUser = m_spel.GetCurrentUser();
```

**GetECP Method, Spel Class**

**Description**

Returns the current ECP number for the current robot.

**Syntax**

Function **GetECP** () As Integer

**Return Value**

Integer containing the current ECP number.

**See Also**

ECP, ECPSet

**GetECP Example**

**VB Example:**

```
saveECP = m_spel.GetECP()  
m_spel.ECP(2)
```

**C# Example:**

```
saveECP = m_spel.GetECP();  
m_spel.ECP(2);
```

**GetErrorMessage Method, Spel Class****Description**

Returns the error message for the specified error or warning code.

**Syntax**

Function **GetErrorMessage** (*ErrorCode* As Integer) As String

**Parameters**

*ErrorCode*      The error code for which to return the associated error message.

**Return Value**

String containing the error message.

**See Also**

ErrorCode

**GetErrorMessage Example****VB Example:**

```
Dim msg As String

If m_spel.ErrorOn Then
    msg = m_spel.GetErrorMessage(m_spel.ErrorCode)
    MsgBox(msg)
End If
```

**C# Example:**

```
string msg;

if (m_spel.ErrorOn) {
    msg = m_spel.GetErrorMessage(m_spel.ErrorCode);
    MessageBox.Show(msg);
}
```

## GetIODef Method, Spel Class

**Description**

Gets the definition information for an input, output, or memory I/O bit, byte, or word.

**Syntax**

Sub **GetIODef**(*Type* As SpelIOLabelTypes, *Index* As Integer, *ByRef Label* as String, *ByRef Description* As String)

**Parameters**

<i>Type</i>	Specifies the I/O type as shown below: InputBit = 1    InputByte = 2    InputWord = 3 OutputBit = 4    OutputByte = 5    OutputWord = 6 MemoryBit = 7    MemoryByte = 8    MemoryWord = 9 InputReal = 10    OutputReal = 11
<i>Index</i>	Specifies the bit or port number.
<i>Label</i>	Returns the label.
<i>Description</i>	Returns the description.

**Return Value**

The values are returned in the Label and Description parameters.

**Remarks**

Use GetIODef to get the labels and descriptions used for all I/O in the current project.

**See Also**

SetIODef

**GetIODef Example****VB Example:**

```
Dim label As String
Dim desc As String
m_spel.GetIODef(SpelIOLabelTypes.InputBit, 0, label, desc)
```

**C# Example:**

```
string label, desc;

m_spel.GetIODef(SpelIOLabelTypes.InputBit, 0, out label, out desc);
```

## GetJRange Method, Spel Class

**Description**

Gets the permissible working range of the specified joint in pulses.

**Syntax**

Function **GetJRange** (*JointNumber* As Integer, *Bound* As Integer) As Integer

**Parameters**

*JointNumber* Integer expression from 1-9 representing the joint number.  
*Bound* Integer expression containing one of two values:  
1: Specifies lower limit value.  
2: Specifies upper limit value.

**Return Value**

Range configuration (integer value, pulses) of the specified joint.

**See Also**

JRange

**GetJRange Example****VB Example:**

```
Dim minRange As Integer
Dim maxRange As Integer
minRange = m_spel.GetJRange(1, 1)
maxRange = m_spel.GetJRange(1, 2)
```

**C# Example:**

```
int minRange, maxRange;
minRange = m_spel.GetJRange(1, 1);
maxRange = m_spel.GetJRange(1, 2);
```

**GetLimitTorque Method, Spel Class**

**Description**

Returns the limit torque for the specified joint for the current robot.

**Syntax**

Function **GetLimitTorque** (*JointNumber* As Integer) As Integer

**Parameters**

*JointNumber* Integer expression for the desired joint.

**Return Value**

Integer value between 1 and 9 which represents the limit torque setting for the specified joint.

**See Also**

GetRealTorque, GetRobotPos, LimitTorque

**GetLimitTorque Example**

**VB Example:**

```
Dim j1LimitTorque As Integer  
j1LimitTorque = m_spel.GetLimitTorque(1)
```

**C# Example:**

```
int j1LimitTorque;  
j1LimitTorque = m_spel.GetLimitTorque(1);
```



---

**GetLimZ Method, Spel Class****Description**

Returns the current LimZ setting.

**Syntax**

Function **GetLimZ** () As Single

**Return Value**

Real value containing the LimZ value.

**See Also**

LimZ, Jump

**GetLimZ Example****VB Example:**

```
saveLimZ = m_spel.GetLimZ()  
m_spel.LimZ(-22)
```

**C# Example:**

```
saveLimZ = m_spel.GetLimZ();  
m_spel.LimZ(-22);
```

**GetPoint Method, Spel Class****Description**

Retrieves coordinate data for a robot point.

**Syntax**

Function **GetPoint** (*PointNumber* As Integer) As SpelPoint

Function **GetPoint** (*PointName* As String) As SpelPoint

**Parameters**

*PointNumber* Integer expression for a point in the Controller's point memory for the current robot.

*PointName* String expression. This can be a point label, "Pxxx", "P\*" or "\*".

**See Also**

SetPoint

**GetPoint Example****VB Example:**

```
Dim pt As SpelPoint
pt = m_spel.GetPoint("P*")
pt.X = 25.0
m_spel.Go(pt)
```

**C# Example:**

```
SpelPoint pt;
pt = m_spel.GetPoint("P0");
pt.X = 25.0;
m_spel.Go(pt);
```

**GetRealTorque Method, Spel Class****Description**

Returns the torque for the specified joint for the current robot.

**Syntax**

Function **GetRealTorque** (*JointNumber* As Integer) As Double

**Parameters**

*JointNumber* Integer expression for the desired joint.

**Return Value**

Double value between 0 and 1 which represents the portion of maximum torque for the current power mode and for the specified joint.

**See Also**

GetLimitTorque, GetRobotPos

**GetRealTorque Example****VB Example:**

```
Dim j1Torque As Double
j1Torque = m_spel.GetRealTorque (1)
```

**C# Example:**

```
double j1Torque;
j1Torque = m_spel.GetRealTorque (1);
```

**GetRobotInfo Method, Spel Class****Description**

Returns robot information.

**Syntax**

Function **GetRobotInfo** (*RobotNumber* As Integer) As SpelRobotInfo

**Parameters**

*RobotNumber* Specifies a robot number

**Return Value**

SpelRobotInfo

**See Also**

GetControllerInfo, GetTaskInfo

**GetRobotInfo Example****VB Example:**

```
Dim info As SpelRobotInfo
Dim msg As String

info = m_spel.GetRobotInfo(1)
msg = "Robot Model: " & info.RobotModel & vbCrLf _
      & "Robot Serial: " & info.RobotSerial
MsgBox(msg)
```

**C# Example:**

```
SpelRobotInfo info;
string msg;

info = m_spel.GetRobotInfo(1);
msg = "Robot Model: " + info.RobotModel +
      "\r\n Robot Serial: " + info.RobotSerial;
MessageBox.Show(msg);
```

## GetRobotPos Method, Spel Class

**Description**

Returns the current robot position.

**Syntax**

Function **GetRobotPos**( *PosType* As SpelRobotPosType, *Arm* As Integer, *Tool* As Integer, *Local* As Integer) As **Single**()

**Parameters**

<i>PosType</i>	Specifies the type of position data to return.
<i>Arm</i>	Integer expression that specifies the robot arm.
<i>Tool</i>	Integer expression that specifies the robot tool.
<i>Local</i>	Integer expression that specifies the robot local.

**Return Value**

Single data type array containing 9 elements. The data returned depends on the specified *PosType*.

World X, Y, Z, U, V, W, R, S, T

Joint J1, J2, J3, J4, J5, J6, J7, J8, J9

Pulse Pls1, Pls2, Pls3, Pls4, Pls5, Pls6, Pls7, Pls8, Pls9

**See Also**

GetPoint

**GetRobotPos Example****VB Example:**

```
Dim values() As Single
values = m_spel.GetRobotPos(SpelRobotPosType.World, 0, 0, 0)
```

**C# Example:**

```
float[] values;
values = m_spel.GetRobotPos(SpelRobotPosType.World, 0, 0, 0);
```

GetSpeed Method, Spel Class

**Description**

Returns one of the three speed settings for the current robot.

**Syntax**

Function **GetSpeed** (*ParamNumber* As Integer) As Integer

**Parameters**

*ParamNumber* Integer expression which evaluates to one of the values shown below.  
1: PTP motion speed  
2: Jump depart speed  
3: Jump approach speed

**Return Value**

Integer expression from 1-100.

**See Also**

Speed

**GetSpeed Example**

**VB Example:**

```
Dim ptpSpeed As Integer  
ptpSpeed = m_spel.GetSpeed(1)
```

**C# Example:**

```
int ptpSpeed;  
ptpSpeed = m_spel.GetSpeed(1) ;
```

## GetTaskInfo Method, Spel Class

**Description**

Returns task information.

**Syntax**

Function **GetTaskInfo** (*TaskName* As String) As SpelTaskInfo

Function **GetTaskInfo** (*TaskNumber* As Integer) As SpelTaskInfo

**Parameters**

*TaskName* Specifies a task name

*TaskNumber* Specifies a task numbers

**Return Value**

SpelTaskInfo

**See Also**

GetControllerInfo, GetRobotInfo

**GetTaskInfo Example****VB Example:**

```
Dim info As SpelTaskInfo
```

```
Dim msg As String
```

```
info = m_spel.GetTaskInfo(1)
```

```
msg = "Task Name: " & info.TaskName & vbCrLf _
```

```
    & "Task State: " & info.TaskState
```

```
MsgBox(msg)
```

**C# Example:**

```
SpelTaskInfo info;
```

```
string msg;
```

```
info = m_spel.GetTaskInfo(1);
```

```
msg= "Task Name: " + info.TaskName +
```

```
    "\r\n Task State: " + info.TaskState;
```

```
MessageBox.Show(msg);
```

**GetTool Method, Spel Class**

**Description**

Returns the current Tool number for the current robot.

**Syntax**

Function **GetTool** () As Integer

**Return Value**

Integer containing the current tool number.

**See Also**

Arm, TLSet, Tool

**GetTool Example**

**VB Example:**

```
saveTool = m_spel.GetTool()  
m_spel.Tool(2)
```

**C# Example:**

```
saveTool = m_spel.GetTool();  
m_spel.Tool(2);
```



## GetVar Method, Spel Class

**Description**

Returns the value of a SPEL<sup>+</sup> global preserve variable in the Controller.

**Syntax**

Function **GetVar**(*VarName* As String) As Object

**Parameters**

*VarName*            The name of the SPEL<sup>+</sup> global preserve variable.  
For an array, the entire array can be returned or just one element.

**Return Value**

Returns the value whose data type is determined by the type of the SPEL<sup>+</sup> variable.

**Remarks**

You can use *GetVar* to retrieve values of any global preserve variables in the Controller's current project. Before you can retrieve values, the project must be successfully built.

If you want to retrieve an entire array, then supply the array name in *VarName*. To retrieve one element of an array, supply the subscript in *VarName*.

**See Also**

SetVar

**GetVar Example**

In the SPEL<sup>+</sup> project, the variable is declared:

```
Global Preserve Integer g_myIntVar
Global Preserve Real g_myRealArray(10)
Global Preserve String g_myStringVar$
Function main
    ...
End
```

In the Visual Basic project:

Since *g\_myIntVar* is declared as an integer, the Visual Basic variable used to retrieve the value of *g\_myIntVar* must be declared as an Integer. For *g\_myRealArray*, the Visual Basic variable must be declared as a Single array.

```
Dim myIntVar As Integer
Dim myRealArray() As Single
Dim myStringVar As String

myIntVar = m_spel.GetVar("g_myIntVar")
myRealArray = m_spel.GetVar("g_myRealArray")
myStringVar = m_spel.GetVar("g_myStringVar$")
```

In the C# project:

Since `g_myIntVar` is declared as an integer, the C# variable used to retrieve the value of `g_myIntVar` must be declared as an `int`. For `g_myRealArray`, the C# variable must be declared as a float array.

```
int myIntVar;
float[] myRealArray;
string myStringVar;

myIntVar = m_spel.GetVar("g_myIntVar");
myRealArray = m_spel.GetVar("g_myRealArray");
myStringVar = m_spel.GetVar("g_myStringVar$");
```

## Go Method, Spel Class

**Description**

Moves the arm in a Point to Point fashion from the current position to the specified point or XY position. The **GO** instruction can move any combination of the robot axes at the same time.

**Syntax**

```
Sub Go (PointNumber As Integer)
Sub Go (Point As SpelPoint)
Sub Go (Point As SpelPoint, AttribExpr As String)
Sub Go (PointExpr As String)
```

**Parameters**

Each syntax has one parameter that specifies the end point which the arm travels to during the Go motion. This is the final position at the end of the point to point motion.

<i>PointNumber</i>	Specifies the end point by using the point number for a previously taught point in the Controller's point memory for the current robot.
<i>Point</i>	Specifies the end point by using a SpelPoint data type.
<i>AttribExpr</i>	Specifies the end point attributes by using a string expression.
<i>PointExpr</i>	Specifies the end point by using a string expression.

**See Also**

Accel, Speed  
Arc, Arc3, CVMove, Jump, Jump3, Jump3CP, Move  
BGo, BMove, TGo, TMove  
Arch, CP, Sense, Till

**Go Example****VB Example:**

```
' Point specified using point number
m_spel.Tool(1)
m_spel.Go(100)

' Point specified using SpelPoint
Dim pt As SpelPoint
pt = m_spel.GetPoint("P*")
pt.X = 125.5
m_spel.Go(pt)

' Point specified using expression
m_spel.Go("P0 /L /2")
m_spel.Go("P1 :Z(-20)")

' Using parallel processing
m_spel.Go("P1 !D50; On 1; D90; Off 1!")

' Point specified using label
m_spel.Go("pick")
```

### C# Example:

```
// Point specified using point number
m_spel.Tool(1);
m_spel.Go(100);

// Point specified using SpelPoint
SpelPoint pt;
pt = m_spel.GetPoint("P0");
pt.X = 125.5;
m_spel.Go(pt);

// Point specified using expression
m_spel.Go("P0 /L /2");
m_spel.Go("P1 :Z(-20)");

// Using parallel processing
m_spel.Go("P1 !D50; On 1; D90; Off 1!");

// Point specified using label
m_spel.Go("pick");
```

---

**Halt Method, Spel Class****Description**

Suspends execution of the specified task.

**Syntax**

Sub **Halt** (*TaskNumber* As Integer)

Sub **Halt** (*TaskName* As String)

**Parameters**

*TaskNumber*     The task number of the task to be suspended.  
The range of the task number is 1 to 32.

*TaskName*     A string expression containing the name of the task to be suspended.

**See Also**

Resume, Xqt

**Halt Example****VB Example:**

```
m_spel.Halt(3)
```

**C# Example:**

```
m_spel.Halt(3);
```

Here Method, Spel Class

**Description**

Teaches a point at the current position.

**Syntax**

Sub **Here** (*PointNumber* As Integer)

Sub **Here** (*PointName* As String)

**Parameters**

*PointNumber* Integer expression for a point in the point memory for the current robot.  
Any valid point number can be used starting with 0.

*PointName* A string expression for a point label.

**See Also**

SetPoint

**Here Example**

**VB Example:**

```
m_spel.Here ("P20")
```

**C# Example:**

```
m_spel.Here ("P20");
```

## HideWindow Method, Spel Class

**Description**

Hides an EPSON RC+ 7.0 window that was previously displayed with ShowWindow.

**Syntax**

Sub **HideWindow** (WindowID As SpelWindows)

**Parameters**

*WindowID*      The ID of the EPSON RC+ 7.0 window to hide.

**See Also**

RunDialog, ShowWindow

**HideWindow Example****VB Example:**

```
Sub btnHideIOMonitor_Click _  
    ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) _  
    Handles btnHideIOMonitor.Click  
  
    m_spel.HideWindow(RCAPINet.SpelWindows.IOMonitor)  
End Sub
```

**C# Example:**

```
void btnHideIOMonitor_Click(object sender, EventArgs e)  
{  
    m_spel.HideWindow(RCAPINet.SpelWindows.IOMonitor);  
}
```

Home Method, Spel Class

**Description**

Moves the robot arm to the user defined home position that is set with the HomeSet method.

**Syntax**

Sub **Home** ()

**See Also**

HomeSet, MCal

**Home Example**

**VB Example:**

```
With m_spel
    .MotorsOn = True
    .Home ()
End With
```

**C# Example:**

```
m_spel.MotorsOn = true;
m_spel.Home ();
```



## Hofs Method, Spel Class

**Description**

Reads or sets the offset pulses used for software zero point correction.

**Syntax**

Function Hofs (JointNumber As Integer) As Integer

Sub Hofs (J1Pulses As Integer, J2Pulses As Integer, J3Pulses As Integer, J4Pulses As Integer, [J5Pulses As Integer], [J6Pulses As Integer], [ J7Pulses As Integer], [J8Pulses As Integer] , [J9Pulses As Integer])

**Parameters**

*J1Pulses – J9Pulses* Integer number representing the pulse for Joints #1 to #9.  
*J5Pulses – J9PulsesI* can be omitted.

**Return Value**

The offset pulse value (integer value, in pulses).

**Remarks**

This method displays or sets the home position offset pulses. This method specifies the offset from the encoder 0 point (Z phase) to the mechanical 0 point.)

Although the robot motion control is based on the zero point of the encoder mounted on each joint motor, the encoder zero point may not necessarily match the robot mechanical zero point.

To make the encoder position that matches the robot mechanical zero point as the zero point on the software, set the correction pulse amount by this method.

**Note****Hofs Values SHOULD NOT be Changed unless Absolutely Necessary**

The Hofs values are correctly specified prior to delivery. There is a danger that unnecessarily changing the Hofs value may result in position errors and unpredictable motion. Therefore, it is strongly recommended that Hofs values not be changed unless absolutely necessary.

**To Automatically Calculate Hofs Values**

To have Hofs values automatically calculated, move the arm to the desired calibration position, and execute Calib. The controller then automatically calculates Hofs values based on the CalPls pulse values and calibration position pulse values.

**Saving and Restoring Hofs**

Hofs can be saved and restored using the Save and Load commands in the [System Configuration] dialog-[Robot]-[Calibration] from the System Configuration menu.

**See Also**

CalPls, Home, Hordr, MCal

**Hofs Example****VB Example:**

```
Dim val As Integer
val = m_spel.Hofs(1)
```

**C# Example:**

```
int val;
val = m_spel.Hofs(1);
```

## HomeSet Method, Spel Class

**Description**

Specifies the position used by the Home method.

**Syntax**

```
Sub HomeSet ( J1Pulses As Integer, J2Pulses As Integer, J3Pulses As Integer,
              J4Pulses As Integer, J5Pulses As Integer, J6Pulses As Integer
              [, J7Pulses As Integer] [, J8Pulses As Integer] [, J9Pulses As Integer] )
```

**Parameters**

*J1Pulses* – *J9Pulses* The Home position encoder pulse value for each joint.  
Joints 7, 8, and 9 are optional.

**See Also**

Home, MCal

**HomeSet Example****VB Example:**

```
' Set the home position at the current position
With m_spel
    .HomeSet(.Pls(1), .Pls(2), .Pls(3), .Pls(4), 0, 0)
End With
```

**C# Example:**

```
//Set the home position at the current position
m_spel.HomeSet(m_spel.Pls(1), m_spel.Pls(2), m_spel.Pls(3),
               m_spel.Pls(4), 0 ,0);
```

## Hordr Method, Spel Class

**Description**

Specifies the order of the axes returning to their HOME positions.

**Syntax**

Sub **Hordr** ( *Home1* As Integer, *Home2* As Integer, *Home3* As Integer, *Home4* As Integer, *Home5* As Integer, *Home6* As Integer [, *Home7* As Integer] [, *Home8* As Integer] [, *Home9* As Integer] )

**Parameters**

*Home 1 - 9* Bit pattern that tells which axes should home during each step of the Home process.  
Any number of axes between 0 to all axes may home during the 1st step.  
*Home 7 - 9* can be specified when R, S, or T axis is specified.

**See Also**

Home, HomeSet, Mcordr

**Hordr Example****VB Example:**

```
m_spel.Hordr (2, 13, 0, 0, 0, 0)
```

**C# Example:**

```
m_spel.Hordr (2, 13, 0, 0, 0, 0);
```

Hour Method, Spel Class

**Description**

Returns the accumulated system operating time in hours.

**Syntax**

Function **Hour** () As Single

**Return Value**

Integer expression representing time.

**Hour Example**

**VB Example:**

```
Dim hoursRunning As Single  
hoursRunning = m_spel.Hour()
```

**C# Example:**

```
float hoursRunning;  
hoursRunning = m_spel.Hour();
```

## ImportPoints Method, Spel Class

**Description**

Imports a point file into the current project for the current robot.

**Syntax**

```
Sub ImportPoints ( SourcePath As String, ProjectFileName As String [, RobotNumber
As Integer] )
```

**Parameters**

<i>SourcePath</i>	String expression containing the specific path and file to import into the current project. The extension must be .pts.
<i>ProjectFileName</i>	String expression containing the specific file to be imported to in the current project for the current robot or specified robot if <i>RobotNumber</i> is supplied. The extension must be .pts.
<i>RobotNumber</i>	Optional. Integer expression for the robot that the point file will be used for. Specify 0 to make it a common point file.

**See Also**

SavePoints

**ImportPoints Example****VB Example:**

```
With m_spel
    .ImportPoints ("c:\mypoints\modell.pts", "robot1.pts")
End With
```

**C# Example:**

```
m_spel.ImportPoints (@"c:\mypoints\modell.pts",
"robot1.pts");
```

In Method, Spel Class

**Description**

Returns the status of the specified input port. Each port contains 8 input bits (one byte).

**Syntax**

Function **In** (*PortNumber* As Integer) As Integer

Function **In** (*Label* As String) As Integer

**Parameters**

*PortNumber* Integer expression representing one of the input ports.  
Each port contains 8 input bits (one byte).

*Label* String expression containing an input byte label.

**Return Value**

Integer from 0 to 255 representing the status of the input port.

**See Also**

InBCD, Out, OpBCD, Sw

**In Example**

**VB Example:**

```
Dim port1Value As Integer  
port1Value = m_spel.In(1)
```

**C# Example:**

```
int port1Value;  
port1Value = m_spel.In(1);
```

**InBCD Method, Spel Class****Description**

Returns the input status of 8 inputs using BCD format. (Binary Coded Decimal)

**Syntax**

Function **InBCD** (*PortNumber* As Integer) As Integer

Function **InBCD** (*Label* As String) As Integer

**Parameters**

*PortNumber* Integer expression representing one of the input ports.

*Label* String expression containing an input byte label.

**Return Value**

Integer from 0 to 9 representing the status of the input port.

**See Also**

In, Out, OpBCD, Sw

**InBCD Example****VB Example:**

```
Dim port1Value As Integer
port1Value = m_spel.InBCD(1)
```

**C# Example:**

```
int port1Value;
port1Value = m_spel.InBCD(1);
```

Inertia Method, Spel Class

**Description**

Specifies the load inertia and eccentricity for the current robot.

**Syntax**

Sub **Inertia** (*LoadInertia* As Single, *Eccentricity* As Single)

**Parameters**

<i>LoadInertia</i>	Real expression that specifies total moment of inertia in kgm <sup>2</sup> around the center of the end effector joint, including end effector and part.
<i>Eccentricity</i>	Real expression that specifies eccentricity in mm around the center of the end effector joint, including end effector and part.

**See Also**

Weight

**Inertia Example**

**VB Example:**

```
m_spel.Inertia(0.02, 1.0)
```

**C# Example:**

```
m_spel.Inertia(0.02, 1.0);
```



---

**Initialize Method, Spel Class****Description**

Initializes the Spel class instance.

**Syntax**

Sub **Initialize** ()

**Remarks**

Normally, the Spel class instance is automatically initialized when the first method has been executed. Initialization can take several seconds as EPSON RC+ 7.0 loads into memory. So in some cases, you may want to call initialize first in your application during startup.

Initialize starts RC+ as a server process according to ServerInstance. Each ServerInstance corresponds to one controller and one project.

If using the ServerInstance property, it must be set before executing Initialize.

**See Also**

Connect, Disconnect, ServerInstance

**Initialize Example****VB Example:**

```
m_spel.Initialize ()
```

**C# Example:**

```
m_spel.Initialize ();
```

InReal Method, Spel Class

**Description**

Returns the input data of 2 words (32 bits) as the floating-point data (IEEE754 compliant) of 32 bits.

**Syntax**

Function **InReal** (*PortNumber* As Integer) As Single

**Parameters**

*PortNumber* Integer expression representing the I/O Input port.

**Return Value**

Returns the input port status in 32 bits floating-point data (IEEE754 compliant).

**See Also**

In, InBCD, InW, Out, OutW, OutReal

**InReal Example**

**VB Example:**

```
Dim val As Single  
val = m_spel.InReal(32)
```

**C# Example:**

```
float val;  
val = m_spel.InReal(32);
```

## InsideBox Method, Spel Class

**Description**

Returns the check status of the approach check area.

**Syntax**

Function **InsideBox** (*BoxNumber* As Integer) As Boolean

**Parameters**

*BoxNumber* Integer expression from 1 to 15 representing which approach check area to return status for.

**Return Value**

True if the robot end effector is inside the specified box, False if not.

**See Also**

Box, InsidePlane

**InsideBox Example****VB Example:**

```
Dim isInside As Boolean
isInside = m_spel.InsideBox(1)
```

**C# Example:**

```
bool isInside;
isInside = m_spel.InsideBox(1);
```

**InsidePlane Method, Spel Class****Description**

Returns the check status of the approach check plane.

**Syntax**

Function **InsidePlane** (*PlaneNumber* As Integer) As Boolean

**Parameters**

*PlaneNumber* Integer expression from 1 to 15 representing which approach check plane to return status for.

**Return Value**

True if the robot end effector is inside the specified box, False if not.

**See Also**

InsideBox, Plane

**InsidePlane Example****VB Example:**

```
Dim isInside As Boolean  
isInside = m_spel.InsidePlane(1)
```

**C# Example:**

```
bool isInside;  
isInside = m_spel.InsidePlane(1);
```

**InW Method, Spel Class****Description**

Returns the status of the specified input word port. Each word port contains 16 input bits.

**Syntax**

Function **InW** (*PortNumber* As Integer) As Integer

Function **InW** (*Label* As String) As Integer

**Parameters**

*PortNumber* Integer number representing an input port.

*Label* String expression containing an input word label.

**Return Value**

Integer value from 0 to 65535 representing the input port

**See Also**

In, InBCD, Out, OpBCD, Sw

**InW Example****VB Example:**

```
Dim data As Integer
data = m_spel.InW(0)
```

**C# Example:**

```
int data;
data = m_spel.InW(0);
```

**IsOptionActive Method, Spel Class**

**Description**

Returns the status of software options.

**Syntax**

Function **IsOptionActive** (*option* As SpelOptions) As Boolean

**Parameters**

*option* Integer number representing the option number.

**Return Value**

False – Disabled

True – Enabled

**See Also**

GetControllerInfo

**IsOptionActive Example**

**VB Example:**

```
Dim ret As Boolean  
ret = m_spel.IsOptionActive(SpelOptions.FieldbusMaster)
```

**C# Example:**

```
bool ret;  
ret = m_spel.IsOptionActive(SpelOptions.FieldbusMaster);
```

## JRange Method, Spel Class

**Description**

Defines the permissible working range of the specified robot joint in pulses.

**Syntax**

Sub **JRange** (*JointNumber* As Integer, *LowerLimitPulses* As Integer, *UpperLimitPulses* As Integer)

**Parameters**

<i>JointNumber</i>	Integer number between 1 - 9 representing the joint for which JRange will be specified.
<i>LowerLimitPulses</i>	Integer number representing the encoder pulse count position for the lower limit range of the specified joint.
<i>UpperLimitPulses</i>	Integer number representing the encoder pulse count position for the upper limit range of the specified joint

**See Also**

XYLim

**JRange Example****VB Example:**

```
m_spel.JRange(1, -30000, 30000)
```

**C# Example:**

```
m_spel.JRange(1, -30000, 30000);
```

JS Method, Spel Class

**Description**

Jump Sense detects whether the arm stopped prior to completing a JUMP instruction (which used a SENSE input) or if the arm completed the JUMP move.

**Syntax**

Function **JS** () As Boolean

**Return Value**

True if the SENSE input was detected during motion, False if not.

**See Also**

Jump, Jump3, Jump3CP, Sense, Till

**JS Example**

**VB Example:**

```
With m_spel
    .Sense("Sw(1) = On")
    .Jump("P1 Sense")
    stoppedOnSense = .JS()
End With
```

**C# Example:**

```
m_spel.Sense("Sw(1) = On");
m_spel.Jump("P1 Sense");
stoppedOnSense = m_spel.JS();
```



## JTran Method, Spel Class

**Description**

Executes a relative joint move.

**Syntax**

Sub **JTran** (*JointNumber* As Integer, *Distance* As Single)

**Parameters**

*JointNumber*     The specific joint to move.

*Distance*        The distance to move. Units are in degrees for rotary joints and millimeters for linear joints.

**See Also**

PTran, Pulse

**JTran Example****VB Example:**

```
' Move joint 1 45 degrees in the plus direction.  
m_spel.JTran(1, 45.0)
```

**C# Example:**

```
// Move joint 1 45 degrees in the plus direction.  
m_spel.JTran(1, 45.0);
```

## Jump Method, Spel Class

**Description**

Moves the arm from the current position to the specified point using point to point motion while first moving in a vertical direction up, then horizontally and then finally vertically downward to arrive on the final destination point.

**Syntax**

Sub **Jump** (*PointNumber* As Integer)

Sub **Jump** (*Point* As SpelPoint)

Sub **Jump** (*Point* As SpelPoint, *AttribExpr* As String)

Sub **Jump** (*PointExpr* As String)

**Parameters**

Each syntax has one parameter that specifies the end point which the arm travels to during the Jump motion. This is the final position at the end of the point to point motion.

*PointNumber* Specifies the end point by using the point number for a previously taught point in the Controller's point memory for the current robot.

*Point* Specifies the end point by using a SpelPoint data type.

*AttribExpr* Specifies the end point attributes by using a string expression.

*PointExpr* Specifies the end point by using a string expression.

**See Also**

Accel, Speed

Arc, Arc3, CVMove, Go, Jump3, Jump3CP, Move

BGo, BMove, TGo, TMove

Arch, CP, Sense, Till

**Jump Example****VB Example:**

```
' Point specified using point number
m_spel.Tool(1)
m_spel.Jump(100)

' Point specified using SpelPoint
Dim pt As SpelPoint
pt = m_spel.GetPoint("P*")
pt.X = 125.5
m_spel.Jump(pt)

' Point specified using expression
m_spel.Jump("P0 /L /2")
m_spel.Jump("P1 :Z(-20)")
m_spel.Jump("P1 C0")
m_spel.Jump("P1 C0 LimZ -10")
m_spel.Jump("P1 C0 Sense Sw(0)=On")

' Using parallel processing
m_spel.Jump("P1 !D50; On 1; D90; Off 1!")

' Point specified using label
m_spel.Jump("pick")
```

**C# Example:**

```
// Point specified using point number
m_spel.Tool(1);
m_spel.Jump(100);

// Point specified using SpelPoint
SpelPoint pt;
pt = m_spel.GetPoint("P0");
pt.X = 125.5;
m_spel.Jump(pt);

// Point specified using expression
m_spel.Jump("P0 /L /2");
m_spel.Jump("P1 :Z(-20)");
m_spel.Jump("P1 C0");
m_spel.Jump("P1 C0 LimZ -10");
m_spel.Jump("P1 C0 Sense Sw(0)=On");

// Using parallel processing
m_spel.Jump("P1 !D50; On 1; D90; Off 1!");

// Point specified using label
m_spel.Jump("pick");
```

## Jump3 Method, Spel Class

**Description**

Motion with 3D gate using a combination of two CP motions and one PTP motion. The robot moves to the depart point, then the approach point, and finally the destination point.

**Syntax**

Sub **Jump3** (*DepartPoint* As Integer, *ApproPoint* As Integer, *DestPoint* As Integer)

Sub **Jump3** (*DepartPoint* As SpelPoint, *ApproPoint* As SpelPoint, *DestPoint* As SpelPoint)

Sub **Jump3** (*DepartPoint* As String, *ApproPoint* As String, *DestPoint* As String)

**Parameters**

<i>DepartPoint</i>	The departure point above the current position using a point number or string point expression.
<i>ApproPoint</i>	The approach point above the destination position using a point number or string point expression.
<i>DestPoint</i>	The target destination of the motion using a point number or string point expression.

**See Also**

Accel, AccelR, AccelS, Speed, SpeedR, SpeedS  
 Arc, Arc3, CVMove, Go, Jump, Jump3CP, Move  
 BGo, BMove, TGo, TMove  
 Arch, CP, Sense, Till

**Jump3 Example****VB Example:**

```
' Points specified using point numbers
m_spel.Tool(1)
m_spel.Jump3(1, 2, 3)

' Points specified using SpelPoint
Dim pd As SpelPoint
Dim pa As SpelPoint
Dim pt As SpelPoint
pd = m_spel.GetPoint("P*")
pd.Z = 125.5
pa = m_spel.GetPoint("P2")
pa.Z = 125.5
pt = m_spel.GetPoint("P2")
m_spel.Jump3(pd, pa, pt)

' Points specified using expressions
m_spel.Jump3("P1", "P2", "P3 C0")
m_spel.Jump3("P1", "P2", "P3 C0 Sense Sw(0)=On")
m_spel.Jump3("P0 -TLZ(10), P1 -TLZ(10), P1")

' Using parallel processing
m_spel.Jump3("P1", "P2", "P3 !D50; On 1; D90; Off 1!")

' Points specified using labels
m_spel.Jump3("depart", "approach", "place")
```

**C# Example:**

```
// Points specified using point numbers
m_spel.Tool(1);
m_spel.Jump3(1, 2, 3);

// Points specified using SpelPoint
SpelPoint pd, pa, pt;
pd = m_spel.GetPoint("P1");
pd.Z = 125.5;
pa = m_spel.GetPoint("P2");
pa.Z = 125.5;
pt = m_spel.GetPoint("P2");
m_spel.Jump3(pd, pa, pt);

// Points specified using expressions
m_spel.Jump3("P1", "P2", "P3 C0");
m_spel.Jump3("P1", "P2", "P3 C0 Sense Sw(0)=On");
m_spel.Jump3("P0 -TLZ(10), P1 -TLZ(10), P1");

// Using parallel processing
m_spel.Jump3("P1", "P2", "P3 !D50; On 1; D90; Off 1!");

// Points specified using labels
m_spel.Jump3("depart", "approach", "place");
```

## Jump3CP Method, Spel Class

**Description**

Motion with 3D gate using a combination of three CP motions.

**Syntax**

Sub **Jump3CP** (*DepartPoint* As Integer, *ApproPoint* As Integer, *DestPoint* As Integer)

Sub **Jump3CP** (*DepartPoint* As SpelPoint, *ApproPoint* As SpelPoint, *DestPoint* As SpelPoint)

Sub **Jump3CP** (*DepartPoint* As String, *ApproPoint* As String, *DestPoint* As String)

**Parameters**

*DepartPoint*            The departure point above the current position using a point number or string point expression.

*ApproPoint*            The approach point above the destination position using a point number or string point expression.

*DestPoint*             The target destination of the motion using a point number or string point expression.

**See Also**

AccelR, AccelS, SpeedR, SpeedS  
Arc, Arc3, CVMove, Go, Jump, Jump3, Move  
BGo, BMove, TGo, TMove  
Arch, CP, Sense, Till

**Jump3CP Example****VB Example:**

```
' Points specified using point numbers
m_spel.Tool(1)
m_spel.Jump3CP(1, 2, 3)

' Points specified using SpelPoint
Dim pd As SpelPoint
Dim pa As SpelPoint
Dim pt As SpelPoint
pd = m_spel.GetPoint("P*")
pd.Z = 125.5
pa = m_spel.GetPoint("P2")
pa.Z = 125.5
pt = m_spel.GetPoint("P2")
m_spel.Jump3CP(pd, pa, pt)

' Points specified using expressions
m_spel.Jump3CP("P1", "P2", "P3 C0")
m_spel.Jump3CP("P1", "P2", "P3 C0 Sense Sw(0)=On")
m_spel.Jump3CP("P0 -TLZ(10), P1 -TLZ(10), P1")

' Using parallel processing
m_spel.Jump3CP("P1", "P2", "P3 !D50; On 1; D90; Off 1!")

' Points specified using labels
m_spel.Jump3CP("depart", "approch", "place")
```

**C# Example:**

```
// Points specified using point numbers
m_spel.Tool(1);
m_spel.Jump3CP(1, 2, 3);

// Points specified using SpelPoint
SpelPoint pd, pa, pt;
pd = m_spel.GetPoint("P0");
pd.Z = 125.5;
pa = m_spel.GetPoint("P2");
pa.Z = 125.5;
pt = m_spel.GetPoint("P2");
m_spel.Jump3CP(pd, pa, pt);

// Points specified using expressions
m_spel.Jump3CP("P1", "P2", "P3 C0");
m_spel.Jump3CP("P1", "P2", "P3 C0 Sense Sw(0)=On");
m_spel.Jump3CP("P0 -TLZ(10), P1 -TLZ(10), P1");

// Using parallel processing
m_spel.Jump3CP("P1", "P2", "P3 !D50; On 1; D90; Off 1!");

// Points specified using labels
m_spel.Jump3CP("depart", "approch", "place");
```

**LimitTorque Method, Spel Class****Description**

Sets the upper limit torque in high power mode for the current robot.

**Syntax**

Sub **LimitTorque** (*AllJointsMax* As Integer)

Sub **LimitTorque** (*J1Max* As Integer, *J2Max* As Integer, *J3Max* As Integer, *J4Max* As Integer, *J5Max* As Integer, *J6Max* As Integer)

**Parameters**

*AllJointsMax* Integer expression for the desired upper limit of torque for all joints in high power mode.

*J1Max – J6Max* Integer expression for the desired upper limit of torque for each joint in high power mode.

**Return Value**

Integer value between 1 and 9 which represents the limit torque setting for the specified joint.

**See Also**

GetRealTorque, GetRobotPos

**LimitTorque Example****VB Example:**

```
Dim j1LimitTorque As Integer  
j1LimitTorque = m_spel.LimitTorque(1)
```

**C# Example:**

```
int j1LimitTorque1  
j1LimitTorque = m_spel.LimitTorque(1);
```



## LimZ Method, Spel Class

**Description**

Sets the default value of the Z axis height for JUMP commands.

**Syntax**

Sub **LimZ** (*ZLimit* As Single)

**Parameters**

*ZLimit*            A coordinate value within the movable range of the Z axis.

**See Also**

Jump

**LimZ Example****VB Example:**

```
saveLimZ = m_spel.GetLimZ()  
m_spel.LimZ(-22)
```

**C# Example:**

```
saveLimZ = m_spel.GetLimZ();  
m_spel.LimZ(-22);
```

LoadPoints Method, Spel Class

**Description**

Loads a SPEL<sup>+</sup> point file into the Controller's point memory for the current robot.

**Syntax**

Sub **LoadPoints** (*FileName* As String [, *Merge* As Boolean])

**Parameters**

*FileName*        A valid point file in the current project.

*Merge*            Optional. Sets to integrate the current point into the specified point file.

**See Also**

ImportPoints, SavePoints

**LoadPoints Example**

**VB Example:**

```
With m_spel
    .LoadPoints ("part1.pts")
End With
```

**C# Example:**

```
m_spel.LoadPoints ("part1.pts");
```

## Local Method, Spel Class

**Description**

Defines local coordinate systems.

**Syntax**

Sub **Local** (*LocalNumber* As Integer, *OriginPoint* As SpelPoint, [*XAxisPoint* As SpelPoint], [*YAxisPoint* As SpelPoint])

Sub **Local** (*LocalNumber* As Integer, *LocalPoint1* As Integer, *BasePoint1* As Integer, *LocalPoint2* As Integer, *BasePoint2* As Integer)

Sub **Local** (*LocalNumber* As Integer, *LocalPoint1* As String, *BasePoint1* As String, *LocalPoint2* As String, *BasePoint2* As String)

**Parameters**

<i>LocalNumber</i>	The local coordinate system number. A total of 15 local coordinate systems (of the integer value from 1 to 15) may be defined.
<i>OriginPoint</i>	SpelPoint variable for the origin of the local coordinate system.
<i>XAxisPoint</i>	Optional. SpelPoint variable for a point along the X axis of the local coordinate system.
<i>YAxisPoint</i>	Optional. SpelPoint variable for a point along the Y axis of the local coordinate system.
<i>LocalPoint1, LocalPoint2</i>	Specifies by interger expression or string expression representing the point data of local coordinate system.
<i>BasePoint1, BasePoint2</i>	Specifies by interger expression or string expression representing the point data of base coordinate system.

**See Also**

Base

**Local Example****VB Example:**

```
Dim originPoint As New SpelPoint
originPoint.X = 100
originPoint.Y = 50
m_spel.Local(1, originPoint)
```

**C# Example:**

```
SpelPoint originPoint = new SpelPoint();
originPoint.X = 100;
originPoint.Y = 50;
m_spel.Local(1, originPoint);
```

LocalClr Method, Spel Class

**Description**

Clears a Local defined for the current robot.

**Syntax**

Sub **LocalClr** (*LocalNumber* As Integer)

**Parameters**

*LocalNumber* Integer expression representing which of 15 locals (integer from 1 to 15) to clear (undefine).

**See Also**

Local, LocalDef

**LocalClr Example**

**VB Example:**

```
m_spel.LocalClr(1)
```

**C# Example:**

```
m_spel.LocalClr(1);
```

**LocalDef Method, Spel Class****Description**

Returns local definition status.

**Syntax**

Function **LocalDef** (*LocalNumber* As Integer) As Boolean

**Parameters**

*LocalNumber* Integer expression (1~15) representing which local coordinate to return status for.

**Return Value**

True if the specified local is defined, False if not.

**See Also**

Local, LocalClr

**LocalDef Example****VB Example:**

```
Dim localExists As Boolean  
localExists = m_spel.LocalDef(1)
```

**C# Example:**

```
bool localExists;  
localExists = m_spel.LocalDef(1);
```

## Login Method, Spel Class

**Description**

Log into EPSON RC+ 7.0 as another user.

**Syntax**

Sub **Login** (*LoginID* As String, *Password* As String)

**Parameters**

*LoginID* String expression containing user login ID.

*Password* String expression containing user password.

**Remarks**

You can utilize EPSON RC+ 7.0 security in your application. For example, you can display a menu that allows different users to log into the system. Each type of user can have its own security rights. For more details on security, see *EPSON RC+ 7.0 User's Guide*.

If security is enabled and you do not execute `LogIn`, then your Visual Basic application will be logged in as the guest user. Or if Auto `LogIn` is enabled in EPSON RC+ 7.0, your application will automatically be logged in as the current Windows user if such a user has been configured in EPSON RC+ 7.0.

**See Also**

`GetCurrentUser`

**Login Example****VB Example:**

```
With m_spel
    .Project = "c:\EpsonRC70\projects\myproject\myproject.sprj"
    .LogIn("operator", "oprpass")
End With
```

**C# Example:**

```
m_spel.Project =
@"c:\EpsonRC70\projects\myproject\myproject.sprj";
m_spel.LogIn("operator", "oprpass");
```

---

**MCal Method, Spel Class****Description**

Executes machine calibration for robots with incremental encoders.

**Syntax**

Sub **MCal** ()

**See Also**

MCalComplete, MotorsOn

**MCal Example****VB Example:**

```
If Not m_spel.MCalComplete() Then  
    m_spel.MCal ()  
End If
```

**C# Example:**

```
if (!m_spel.MCalComplete())  
    m_spel.MCal ();
```

**MCalComplete Method, Spel Class**

**Description**

Returns True if MCal has been completed successfully.

**Syntax**

Function **MCalComplete** () As Boolean

**Return Value**

True if the MCal has completed, False if not.

**See Also**

MCal

**MCalComplete Example**

**VB Example:**

```
If m_spel.MCalComplete() Then
    lblStatus.Text = "MCal Complete"
Else
    lblStatus.Text = "MCal Not Complete"
End If
```

**C# Example:**

```
if (m_spel.MCalComplete())
    lblStatus.Text = "MCal Complete";
else
    lblStatus.Text = "MCal Not Complete";
```



## Mcordr Method, Spel Class

**Description**

Specifies the moving axis order for machine calibration MCal.

**Syntax**

Sub **MCordr** ( *Step1* As Integer, *Step2* As Integer, *Step3* As Integer,  
*Step4* As Integer, *Step5* As Integer, *Step6*As Integer,  
 [*Step7* As Integer], [*Step8* As Integer], [*Step9* As Integer] )

**Parameters**

*Step 1 - 9* Bit pattern that tells which axes should home during each step of the MCal process.  
 Axes between 0 to all axes or any number of axes may home during the 1st step.  
*Step 7 – 9* are optional and are used with robots that have more than 7 axes.

**See Also**

Home, HomeSet, Hordr, MCal

**Mcordr Example****VB Example:**

```
m_spel.Mcordr(2, 13, 0, 0, 0, 0)
```

**C# Example:**

```
m_spel.Mcordr(2, 13, 0, 0, 0, 0);
```

MemIn Method, Spel Class

**Description**

Returns the status of the specified memory I/O byte port. Each port contains 8 memory I/O bits.

**Syntax**

Function **MemIn** (*PortNumber* As Integer) As Integer

Function **MemIn** (*Label* As String) As Integer

**Parameters**

*PortNumber* Integer expression representing one of the memory I/O ports.

*Label* String expression containing a memory I/O byte label.

**Return Value**

Integer containing the port value.

**See Also**

In, InBCD, MemOut, MemSw, Sw, Off, On, Oport

**MemIn Example**

**VB Example:**

```
data = m_spel.MemIn(1)
```

**C# Example:**

```
data = m_spel.MemIn(1);
```

**MemInW Method, Spel Class****Description**

Returns the status of the specified memory I/O word port. Each word port contains 16 memory I/O bits.

**Syntax**

Function **MemInW** (*PortNumber* As Integer) As Integer

Function **MemInW** (*Label* As String) As Integer

**Parameters**

*PortNumber* Integer expression representing the memory I/O word.

*Label* String expression containing a memory I/O word label.

**Return Value**

Integer expression from 0 to 65535 representing the input port status.

**See Also**

In, InBCD, MemIn, MemSw, Sw, Off, On, Oport

**MemInW Example****VB Example:**

```
data = m_spel.MemInW(1)
```

**C# Example:**

```
data = m_spel.MemInW(1);
```

MemOff Method, Spel Class

**Description**

Turns off the specified bit of the S/W memory I/O.

**Syntax**

Sub **MemOff** (*BitNumber* As Integer)

Sub **MemOff** (*Label* As String)

**Parameters**

*BitNumber* Integer expression representing one of the memory I/O bits.

*Label* String expression containing a memory I/O bit label.

**See Also**

In, InBCD, MemOut, MemSw, Sw, Off, On, Oport

**MemOff Example**

**VB Example:**

```
m_spel.MemOff(500)
```

**C# Example:**

```
m_spel.MemOff(500);
```

---

**MemOn Method, Spel Class****Description**

Turns on the specified bit of memory I/O.

**Syntax**

Sub **MemOn** (*BitNumber* As Integer)

Sub **MemOn** (*Label* As String)

**Parameters**

*BitNumber* Integer expression representing one of the memory I/O bits.

*Label* String expression containing a memory I/O bit label.

**See Also**

In, InBCD, MemOut, MemSw, Sw, Off, On, Oport

**MemOn Example****VB Example:**

```
m_spel.MemOn(500)
```

**C# Example:**

```
m_spel.MemOn(500);
```

MemOut Method, Spel Class

**Description**

Simultaneously sets 8 memory I/O bits based on the 8 bit value specified by the user.

**Syntax**

Sub **MemOut** (*PortNumber* As Integer, *Value* As Integer)

Sub **MemOut** (*Label* As String, *Value* As Integer)

**Parameters**

*PortNumber* Integer expression representing one of the memory I/O bytes.

*Label* String expression containing a memory I/O byte label.

*Value* Integer expression containing the output pattern for the specified byte.  
Valid values are from 0 - 255.

**See Also**

In, InBCD, MemIn, MemSw, Sw, Off, On, Oport

**MemOut Example**

**VB Example:**

```
m_spel.MemOut(2, 25)
```

**C# Example:**

```
m_spel.MemOut(2, 25);
```

## MemOutW Method, Spel Class

**Description**

Simultaneously sets 16 memory I/O bits based on the 16 bit value specified by the user.

**Syntax**

Sub **MemOutW** (*PortNumber* As Integer, *Value* As Integer)

Sub **MemOutW** (*Label* As String, *Value* As Integer)

**Parameters**

*PortNumber* Integer expression representing one of the memory I/O words.

*Label* String expression containing a memory I/O word label.

*Value* Specifies output data (integer from 0 to 65535) using an expression or numeric value.

**See Also**

In, InBCD, MemIn, MemSw, Sw, Off, On, Oport

**MemOutW Example****VB Example:**

```
m_spel.MemOutW(2, 25)
```

**C# Example:**

```
m_spel.MemOutW(2, 25);
```

**MemSw Method, Spel Class****Description**

Returns the specified memory I/O bit status.

**Syntax**

Function **MemSw** (*BitNumber* As Integer) As Boolean

Function **MemSw** (*Label* As String) As Boolean

**Parameters**

*BitNumber* Integer expression representing one of the memory I/O bits.

*Label* String expression containing a memory I/O bit label.

**Return Value**

True if the specified memory I/O bit is on, False if not.

**See Also**

In, InBCD, MemIn, Sw, Off, On, Oport

**MemSw Example****VB Example:**

```
If m_spel.MemSw(10) Then  
    m_spel.On(2)  
End If
```

**C# Example:**

```
if (m_spel.MemSw(10))  
    m_spel.On(2);
```



## Move Method, Spel Class

**Description**

Moves the arm from the current position to the specified point using linear interpolation (i.e. moving in a straight line).

**Syntax**

Sub **Move** (*PointNumber* As Integer)

Sub **Move** (*Point* As SpelPoint)

Sub **Move** (*Point* As SpelPoint, *AttribExpr* As String)

Sub **Move** (*PointExpr* As String)

**Parameters**

Each syntax has one parameter that specifies the end point which the arm travels to during the Move motion. This is the final position at the end of the linear interpolated motion.

*PointNumber* Specifies the end point by using the point number for a previously taught point in the Controller's point memory for the current robot.

*Point* Specifies the end point by using a SpelPoint data type.

*AttribExpr* Specifies the end point attributes by using a string expression.

*PointExpr* Specifies the end point by using a string expression.

**See Also**

AccelR, AccelS, SpeedR, SpeedS

Arc, Arc3, CVMove, Go, Jump, Jump3, Jump3CP

BGo, BMove, TGo, TMove

Arch, CP, Till

**Move Example****VB Example:**

```
' Point specified using point number
m_spel.Tool(1)
m_spel.Move(100)

' Point specified using SpelPoint
Dim pt As SpelPoint
pt = m_spel.GetPoint("P*")
pt.X = 125.5
m_spel.Move(pt)

' Point specified using expression
m_spel.Move("P0 /L /2 ROT")
m_spel.Move("P1 :Z(-20)")

' Using parallel processing
m_spel.Move("P1 !D50; On 1; D90; Off 1!")

' Point specified using label
m_spel.Move("pick")
```

### C# Example:

```
// Point specified using point number
m_spel.Tool(1);
m_spel.Move(100);

// Point specified using SpelPoint
SpelPoint pt;
pt = m_spel.GetPoint("P0");
pt.X = 125.5;
m_spel.Move(pt);

// Point specified using expression
m_spel.Move("P0 /L /2 ROT");
m_spel.Move("P1 :Z(-20)");

// Using parallel processing
m_spel.Move("P1 !D50; On 1; D90; Off 1!");

// Point specified using label
m_spel.Move("pick");
```

## Off Method, Spel Class

**Description**

Turns off the specified output.

**Syntax**

Sub **Off** (*BitNumber* As Integer)

Sub **Off** (*Label* As String)

**Parameters**

*BitNumber* Integer expression representing one of the standard or expansion outputs. This tells the **Off** instruction which output to turn off.

*Label* String expression containing an output bit label.

**See Also**

On, Oport, Out, OutW

**Off Example****VB Example:**

```
m_spel.Off(1)
```

**C# Example:**

```
m_spel.Off(1);
```

**OLRate Method, Spel Class****Description**

Returns overload rating for one joint for the current robot.

**Syntax**

Function **OLRate** (*JointNumber* As Integer) As Single

**Parameters**

*JointNumber*                    Integer expression from 1-9 representing the joint number.

**Return Value**

Returns the OLRate for the specified joint. Values are between 0.0 and 2.0.

**Remarks**

OLRate can be used to check whether a cycle is causing stress on the servo system. Factors such as temperature and current can cause servo errors during applications with high duty cycles. OLRate can help to check if the robot system is close to having a servo error.

During a cycle, run another task to command OLRate. If OLRate exceeds 1.0 for any joint, then a servo error will occur.

Servo errors are more likely to occur with heavy payloads. By using OLRate during a test cycle, you can help insure that the speed and acceleration settings will not cause a servo error during production cycling.

To get valid readings, you must execute this method while the robot is moving.

This method will not be used in proper payloads state.

**OLRate Example****VB Example:**

```
Dim data As Single  
data = m_spel.OLRate(1)
```

**C# Example:**

```
float data;  
data = m_spel.OLRate(1);
```

---

**On Method, Spel Class****Description**

Turns on the specified output.

**Syntax**

Sub **On** (*BitNumber* As Integer)

Sub **On** (*Label* As String)

**Parameters**

*BitNumber* Integer expression representing one of the standard or expansion outputs.

*Label* String expression containing an output bit label.

**See Also**

Off, Oport, Out, OutW

**On Example****VB Example:**

```
m_spel.On(1)
```

**C# Example:**

```
m_spel.On(1);
```

OpBCD Method, Spel Class

**Description**

Simultaneously sets 8 output bits using BCD (Binary Coded Decimal) format.

**Syntax**

**OpBCD** (*PortNumber* As Integer, *Value* As Integer)

**OpBCD** (*Label* As String, *Value* As Integer)

**Parameters**

*PortNumber* Integer number representing one of the ports.  
Each port contains 8 output bits (one byte).

*Value* Integer number between 0-99 representing the output pattern for the specified port.  
The 2nd digit (called the 1's digit) represents the lower 4 outputs in the port and the 1st digit (called the 10's digit) represents the upper 4 outputs in the port.

**See Also**

Off, Out, Sw

**OpBCD Example**

**VB Example:**

```
m_spel.OpBCD(1, 25)
```

**C# Example:**

```
m_spel.OpBCD(1, 25);
```

---

**Oport Method, Spel Class****Description**

Returns the state of the specified output bit.

**Syntax**

Function **Oport** (*BitNumber* As Integer) As Boolean

Function **Oport** (*Label* As String) As Boolean

**Parameters**

*BitNumber* Integer expression representing one of the standard or expansion outputs.

*Label* String expression containing an output byte label.

**Return Value**

True if the specified output bit is on, False if not.

**See Also**

Off, On, OpBCD, Out, Sw

**Oport Example****VB Example:**

```
If m_spel.Oport(1) Then
    m_spel.On(2)
End If
```

**C# Example:**

```
if(m_spel.Oport(1))
    m_spel.On(2);
```

**Out Method, Spel Class****Description**

Simultaneously reads or sets 8 output bits (one byte).

**Syntax**

Sub **Out** (*PortNumber* As Integer, *Value* As Integer)

Sub **Out** (*Label* As String, *Value* As Integer)

Function **Out** (*PortNumber* As Integer) As Integer

Function **Out** (*Label* As String) As Integer

**Parameters**

*PortNumber* Integer number representing one of the output ports.

*Label* String expression containing an output byte label.

*Value* Integer number between 0-255 representing the output pattern for the output port. If represented in hexadecimal form the range is from &H0 to &HFF.

**Return Value**

Integer number between 0-255 containing the port value.

**See Also**

InBCD, OpBCD, Oport, OutW, Sw

**Out Example****VB Example:**

```
m_spel.Out(1, 240)
```

**C# Example:**

```
m_spel.Out(1, 240);
```



## OutReal Method, Spel Class

**Description**

Gets or sets the output port status as the 32 bits floating-point data (IEEE754 compliant).

**Syntax**

Function OutReal (*WordPortNumber* As Integer) As Single

Sub OutReal (*WordPortNumber* As Integer, *Value* As Single)

**Parameters**

*WordPortNumber* Integer representing output port

*Value* Real vale representing output data

**Return Value**

Returns the specified output port status in 32 bits floating-point data (IEEE754 compliant).

**See Also**

In, InBCD, InReaql, InW, Out, OutW

**OutReal Example****VB Example:**

```
Dim val As Single  
val = m_spel.OutReal(32)
```

**C# Example:**

```
float val;  
val = m_spel.OutReal(32);
```

**OutW Method, Spel Class****Description**

Simultaneously reads or sets 16 output bits (one word).

**Syntax**

Sub **OutW** (*PortNumber* As Integer, *Value* As Integer)

Sub **OutW** (*Label* As String, *Value* As Integer)

Function **OutW** (*PortNumber* As Integer) As Integer

Function **OutW** (*Label* As String) As Integer

**Parameters**

*PortNumber* Integer number representing one of the output ports.

*Label* String expression containing an output word label.

*Value* Integer number between 0-65535 representing the output pattern for the output port. If represented in hexadecimal form the range is from &H0 to &HFFFF.

**Return Value**

Integer number between 0-65535 containing the port value.

**See Also**

InBCD, OpBCD, Oport, Out, Sw

**OutW Example****VB Example:**

```
m_spel.OutW(1, 240)
```

**C# Example:**

```
m_spel.OutW(1, 240);
```

## PAgl Method, Spel Class

**Description**

Returns the joint angle for the selected rotational axis, or position for the selected linear axis, of the specified point.

**Syntax**

Function **PAgl** (*PointNumber* As Integer, *JointNumber* As Integer) As Single

Function **PAgl** (*Point* As SpelPoint, *JointNumber* As Integer) As Single

Function **PAgl** (*Label* As String, *JointNumber* As Integer) As Single

**Parameters**

*PointNumber* Integer expression representing the point number of a point in the current robot's point memory.

*Point* A previously initialized SpelPoint.

*Label* A string expression containing a point label of a point in the current robot's point memory.

*JointNumber* Integer expression representing the desired joint number.  
The value can be from 1 to 9.

**Return Value**

Single containing the angle for the specified joint in degrees or millimeters.

**See Also**

Agl, Pls, CX – CT

**PAgl Example****VB Example:**

```
Dim t1Angle As Single
t1Angle = m_spel.PAgl(1, 1)
```

**C# Example:**

```
float t1Angle;
t1Angle = m_spel.PAgl(1, 1);
```

## Pallet Method, Spel Class

**Description**

Defines pallets.

**Syntax**

Sub **Pallet** ( *PalletNumber* As Integer, *Point1* As String, *Point2* As String, *Point3* As String  
[, *Point4* As String] , *rows* As Integer, *columns* As Integer )

**Parameters**

<i>PalletNumber</i>	Pallet number represented by an integer number from 0 to 15.
<i>Point1</i>	Point variable which defines first pallet position.
<i>Point2</i>	Point variable which defines second pallet position.
<i>Point3</i>	Point variable which defines third pallet position.
<i>Point4</i>	Optional. Point variable which defines fourth pallet position.
<i>Rows</i>	Numbers of points on lateral side of the pallet. Each number is an integer from 1 to 32767.
<i>Columns</i>	Numbers of points on longitudinal side of the pallet. Each number is an integer from 1 to 32767.

**See Also**

Jump, Go, SetPoint

**Pallet Example****VB Example:**

```
m_spel.Pallet(1, 1, 2, 3, 4, 3, 4)
```

**C# Example:**

```
m_spel.Pallet(1, 1, 2, 3, 4, 3, 4);
```

## Pass Method, Spel Class

**Description**

Specifies the PTP motion to pass a neighborhood of a specified point without stopping motion.

**Syntax**

Sub **Pass**(PointNumber As Integer)

Sub **Pass**(PassExpr As String )

**Parameters**

<i>PointNumber</i>	Specifies a point using the taught point from point memory of the current robot saved in the Controller.
<i>PassExpr</i>	Specifies a point using string expression. Point specification [, {On   Off   MemOn   MemOff} bit number [,point specification ... ]] [LJM [Orientation flag]]
Point specification	Specifies a point number, P(expression), or point label. If the point data is complete and listed in ascending or descending order, two point numbers can be combined using a colon and specified like P(1:5).
Bit number	Specifies I/O output bit or memory I/O bit to turn on/off using an integer or output label.
LJM	Optional. Converts the departure coordinates, approach coordinates, and target coordinates using LJM function.
Orientation flag	Optional. Specifies an orientation flag parameter for the LJM function.

**See Also**

Accel, Go, Jump, Speed

**Pass Example****VB Example:**

```
m_spel.Jump(1)
m_spel.Pass(2) ' Move the Arm #2 closer to P2 and execute the following
               command before reaching P2

m_spel.On(2)
m_spel.Pass(3)
m_spel.Pass(4)
m_spel.Off(0)
m_spel.Pass(5)
```

**C# Example:**

```
m_spel.Jump(1);
m_spel.Pass(2); //Move the Arm #2 closer to P2 and execute the following
               command before reaching P2

m_spel.On(2);
m_spel.Pass(3);
m_spel.Pass(4);
m_spel.Off(0);
m_spel.Pass(5);
```

**Pause Method, Spel Class****Description**

Causes all normal SPEL<sup>+</sup> tasks in the Controller to pause. If the robot is moving, it will immediately decelerate to a stop.

**Syntax**

```
Sub Pause ()
```

**See Also**

Continue, EventReceived, Stop

**Pause Example****VB Example:**

```
Sub btnPause_Click() _  
    ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) _  
    Handles btnPause.Click  
  
    m_spel.Pause ()  
    btnPause.Enabled = False  
    btnContinue.Enabled = True  
End Sub
```

**C# Example:**

```
void btnPause_Click(object sender, EventArgs e)  
{  
    m_spel.Pause ();  
    btnPause.Enabled = false;  
    btnContinue.Enabled = true;  
}
```

## PDef Method, Spel Class

**Description**

Returns the definition status of a specified point.

**Syntax**

Function **PDef** (*PointNumber* As Integer) As Boolean

**Parameters**

*PointNumber* Integer expression for the point number of a point in the current robot's point memory.

**Return Value**

True if the specified point is defined, False if not.

**See Also**

PDel

**PDef Example****VB Example:**

```
Dim p1Defined As Boolean  
p1Defined = m_spel.PDef (1)
```

**C# Example:**

```
bool p1Defined;  
p1Defined = m_spel.PDef (1);
```

**PDel Method, Spel Class**

**Description**

Deletes specified position data.

**Syntax**

Sub **PDel** (*FirstPointNumber* As Integer [, *LastPointNumber* As Integer])

**Parameters**

*FirstPointNumber* Integer expression that specifies the first point in the range to delete.

*LastPointNumber* Optional. Integer expression the specifies the last point in range to delete. If omitted, only the point specified in *FirstPointNumber* is deleted.

**See Also**

PDef, LoadPoints, Clear, SavePoints

**PDel Example**

**VB Example:**

```
m_spel.PDel(1, 10)  
m_spel.SavePoints("modell.pts")
```

**C# Example:**

```
m_spel.PDel(1, 10);  
m_spel.SavePoints("modell.pts");
```



## PeakSpeed Method, Spel Class

**Description**

Returns the peak speed values for the specified joint.

**Syntax**

Function **PeakSpeed** (*JointNumber* As Integer) As Single

**Parameters**

*JointNumber* Integer expression from 1-9 representing the joint number.

**Return Value**

Real value from -1 to 1.

**Remarks**

This method returns the value of the maximum absolute speed for the joint with a sign. The peak speed is a real number from -1 to 1 with 1 being the maximum speed.

Execute PeakSpeedClear method first, and then execute this method to display the peak speed value for the joint.

When using the virtual controller or conducting dry-run, the average of the absolute speed values is calculated from the commanded speed instead of the actual speed.

This method does not support the PG additional axes.

**See Also**

AvgSpeed, AvgSpeedClear, PeakSpeedClear

**PeakSpeed Example****VB Example:**

```
Dim val As Single  
val = m_spel.PeakSpeed(1)
```

**C# Example:**

```
float val;  
val = m_spel.PeakSpeed(1);
```

PeakSpeedClear Method, Spel Class

**Description**

Clears and initializes the peak speed for one or more joints.

**Syntax**

Sub **PeakSpeedClear** ()

**Remarks**

This method clears the peak speed values for the specified joints.

You must execute this method before executing PeakSpeed method.

This method does not support the PG additional axes.

**See Also**

AvgSpeed, AvgSpeedClear, PeakSpeed

**PeakSpeedClear Example**

**VB Example:**

```
m_spel.PeakSpeedClear ()
```

**C# Example:**

```
m_spel.PeakSpeedClear ();
```

**PF\_Abort Method, Spel Class****Description**

Forces the Part Feeding process to stop for the specified part.

**Syntax**

Sub **PF\_Abort** (*PartID* As Integer)

**Parameters**

*PartID* Specifies the part ID (1 to 16).

**Remarks**

Immediately aborts the Part Feeding process for the specified part.

Unlike PF\_Stop method, this aborts the callback function in progress.

Nothing will occur when using this function when the Part Feeding process has not been started.

**PF\_Abort Example****VB Example:**

```
m_spel.PF_Abort(1)
```

**C# Example:**

```
m_spel.PF_Abort(1);
```

**PF\_Backlight Method, Spel Class**

**Description**

Turns the built-in backlight on or off.

**Syntax**

Sub **PF\_Backlight** (*FeederNumber* As Integer, *State* As Boolean)

**Parameters**

*FeederNumber*            Integer number representing the feeder number  
*State*                    Sets On (True) / Off (False)

**Remarks**

When the system is controlling vision, the backlight is automatically turned on and off as needed. Use this method to control the built-in backlight on status when the system is not controlling vision.

**PF\_Backlight Example**

**VB Example:**

```
m_spel.PF_Backlight(1, True)
```

**C# Example:**

```
m_spel.PF_Backlight(1, true);
```

**PF\_BacklightBrightness Method, Spel Class****Description**

Sets the brightness for the built-in backlight.

**Syntax**

Sub **PF\_BacklightBrightness** (*FeederNumber* As Integer, *Brightness* As Integer)

**Parameters**

*FeederNumber* Integer number representing the feeder number  
*SBrightness* The percentage of brightness from 0 to 100%.

**Remarks**

Normally, the built-in backlight brightness to be used for a part is set in the Part Feeding Configuration dialog. If changes to brightness are required at runtime, you can use this method to change it.

**PF\_BacklightBrightness Example****VB Example:**

```
m_spel.PF_BacklightBrightness(1, 80)
```

**C# Example:**

```
m_spel.PF_BacklightBrightness(1, 80);
```

PF\_Name Method, Spel Class

**Description**

Gets a part name from a part ID.

**Syntax**

Function **PF\_Name** (*PartID* As Integer) As String

**Parameters**

*PartID* Integer number representing the part ID (1 to 16).

**Return Value**

Returns the name of the specified part ID as a character string.

**Remarks**

If the specified part ID is disabled, “ ” (empty character) is returned.

**PF\_Name Example**

**VB Example:**

```
Dim part1Name As String  
Part1Name = m_spel.PF_Name(1)
```

**C# Example:**

```
string part1Name;  
part1Name = m_spel.PF_Name(1);
```

## PF\_Number Method, Spel Class

**Description**

Gets a part ID from a part name.

**Syntax**

Function **PF\_Number** (*PartName* As String) As Integer

**Parameters**

*PartName*                      Character string representing the part name

**Return Value**

Returns the part ID (integer number from 1 to 16) for the specified part name.

**Remarks**

Returns -1 if the corresponding part name does not exist.

If multiple parts with the same name exist, the part with the smallest ID will be retrieved.

**PF\_Number Example****VB Example:**

```
Dim part1ID As Integer
Part1ID = m_spel.PF_Number("Part1")
```

**C# Example:**

```
int part1ID;
part1ID = m_spel.PF_Number("Part1");
```

## PF\_Start Method, Spel Class

**Description**

Starts the Part Feeding process for a specified part.

**Syntax**

Sub PF\_Start (PartID1 As Integer, [PartID2 As Integer], [PartID3 As Integer], [PartID4 As Integer])

**Parameters**

<i>PartID1</i>	Integer number representing the master part ID (1 to 16).
<i>PartID2</i>	Optional. Integer number representing the slave part ID (1 to 16).
<i>PartID3</i>	Optional. Integer number representing the slave part ID (1 to 16).
<i>PartID4</i>	Optional. Integer number representing the slave part ID (1 to 16).

**Remarks**

Perform the following before starting this method.

- Select the robot in use
- Turn the motors on
- Run PF\_InitLog when outputting a log

Running this method generates a new task and returns control to the caller.

When this occurs, the Status callback function will be run under the following conditions. The Part Feeding process will not start.

Condition	Status callback function Status parameter value
The part ID is invalid	PF STATUS BAD ID
Part parameter settings are invalid (Enabled check box not selected, etc.)	PF_STATUS_BAD_PARAMETER
Feeder calibration not complete	PF STATUS CAL NOT COMPLETE
An error occurred	PF STATUS ERROR

This method cannot be run multiple times at the same time. Once initiated, the processing already executed will continue. An error will not occur.

PF\_Start should be run from a normal task. An error will occur when attempting to run PF\_Task from a background task.

**Caution**

If a part ID that does not exist is specified in this method, a 7600 error will occur.

**PF\_Start Example****VB Example:**

```
m_spel.PF_Start(1)
```

**C# Example:**

```
m_spel.PF_Start(1);
```



## PF\_Stop Method, Spel Class

**Description**

Issues a Part Feeding process end request.

This will wait for running callback functions to finish.

Once complete, the PF\_CycleStop callback function will run and the process will stop.

**Syntax**

Sub **PF\_Stop** (*PartID* As Integer)

**Parameters**

*PartID* Integer number representing the part ID (1 to 16).

**Remarks**

Stops the Part Feeding process for the specified part.

Unlike the PF\_Abort method, PF\_Stop will wait for running callback functions to finish.

Once callback functions are complete, the PF\_CycleStop callback function will run.

Nothing will occur when using this function when the Part Feeding process has not been started.

**PF\_Stop Example****VB Example:**

```
m_spel.PF_Stop(1)
```

**C# Example:**

```
m_spel.PF_Stop(1);
```

**PLabel Method, Spel Class**

**Description**

Gets or sets the point label associated with a point number.

**Syntax**

Function **PLabel** (*PointNumber* As Integer) As String

Sub **PLabel** (*PointNumber* As Integer, *PointName* As String)

**Parameters**

*PointNumber* Integer number representing the point number.

*PointName* String expression representing the label to use for the specified point.

**Return Value**

Returns the corresponding label to the specified point number.

**See Also**

PDef

**PLabel Example**

**VB Example:**

```
Dim pt1Label As String  
Pt1Label = m_spel.PLabel(1)
```

**C# Example:**

```
string pt1Label;  
pt1Label = m_spel.PLabel(1);
```

## Plane Method, Spel Class

**Description**

Defines a Plane.

**Syntax**

Sub **Plane** (*PlaneNumber* As Integer, *Point* As SpelPoint)

Sub **Plane** (*PlaneNumber* As Integer, *X* As Single, *Y* As Single, *Z* As Single, *U* As Single, *V* As Single, *W* As Single)

**Parameters**

<i>PlaneNumber</i>	Integer number from 1-15 representing which of the 15 Planes to define.
<i>Point</i>	Point data representing the coordinate data of the approach check plane.
<i>X</i>	The X coordinate of the point representing the coordinate data of the approach check plane.
<i>Y</i>	The Y coordinate of the point representing the coordinate data of the approach check plane.
<i>Z</i>	The Z coordinate of the point representing the coordinate data of the approach check plane.
<i>U</i>	The U coordinate of the point representing the coordinate data of the approach check plane.
<i>V</i>	The V coordinate of the point representing the coordinate data of the approach check plane.
<i>W</i>	The W coordinate of the point representing the coordinate data of the approach check plane.

**See Also**

PlaneClr, PlaneDef

**Plane Example****VB Example:**

```
m_spel.Plane(1, -5, 5, -10, 10, -20, 20)
```

**C# Example:**

```
m_spel.Plane(1, -5, 5, -10, 10, -20, 20);
```

PlaneClr Method, Spel Class

**Description**

Clears (undefines) a Plane.

**Syntax**

Sub **PlaneClr** (*PlaneNumber* As Integer)

**Parameters**

*PlaneNumber* Integer number from 1-15 representing which of the 15 Planes to clear.

**See Also**

Plane, PlaneDef

**PlaneClr Example**

**VB Example:**

```
m_spel.PlaneClr(1)
```

**C# Example:**

```
m_spel.PlaneClr(1);
```

---

**PlaneDef Method, Spel Class****Description**

Returns whether a plane is defined.

**Syntax**

Function **PlaneDef** (*PlaneNumber* As Integer) As Boolean

**Parameters**

*PlaneNumber* Integer expression representing the plane number from 1 to 15.

**Return Value**

True if the specified plane is defined, False if not.

**See Also**

Plane, PlaneClr

**PlaneDef Example****VB Example:**

```
x = m_spel.PlaneDef(1)
```

**C# Example:**

```
x = m_spel.PlaneDef(1);
```

Pls Method, Spel Class

**Description**

Returns the current encoder pulse count for each axis at the current position.

**Syntax**

Function **Pls** (*JointNumber* As Integer) As Integer

**Parameters**

*JointNumber*    The specific axis for which to get the current encoder pulse count.  
(1 to 9)

**Return Value**

Integer containing the current pulse count for the specified joint.

**See Also**

Agl, Pulse

**Pls Example**

**VB Example:**

```
j1Pulses = m_spel.Pls(1)
```

**C# Example:**

```
j1Pulses = m_spel.Pls(1);
```

**PTCLR Method, Spel Class****Description**

Clears and initializes the peak torque for one or more joints.

**Syntax**

Sub **PTCLR** ()

**Remarks**

This method clears the peak torque values for the specified joints.

You must execute this method before executing PTRQ method.

**See Also**

ATCLR, ATRQ, PTRQ

**PTCLR Example****VB Example:**

```
m_spel . PTCLR ()
```

**C# Example:**

```
m_spel . PTCLR () ;
```

## PTPBoost Method, Spel Class

**Description**

Sets the boost parameters for short distance PTP (point to point) motion.

**Syntax**

Sub **PTPBoost** (*BoostValue* As Integer [, *DepartBoost* As Integer] [, *ApproBoost* As Integer])

**Parameters**

*BoostValue* Integer expression from 0 - 100.

*DepartBoost* Optional. Jump depart boost value. Integer expression from 0 - 100.

*ApproBoost* Optional. Jump approach boost value. Integer expression from 0 - 100.

**See Also**

PTPBoostOK

**PTPBoost Example****VB Example:**

```
m_spel.PTPBoost(50)  
m_spel.PTPBoost(50, 30, 30)
```

**C# Example:**

```
m_spel.PTPBoost(50);  
m_spel.PTPBoost(50, 30, 30);
```



## PTPBoostOK Method, Spel Class

**Description**

Returns whether or not the PTP (Point to Point) motion from a current position to a target position is a small travel distance.

**Syntax**

Function **PTPBoostOK** (*PointNumber* As Integer) As Boolean

Function **PTPBoostOK** (*Point* As SpelPoint) As Boolean

Function **PTPBoostOK** (*PointExpr* As String) As Boolean

**Parameters**

Each syntax has one parameter that specifies the target point to check.

*PointNumber* Specifies the target point by using the point number for a previously taught point in the Controller's point memory for the current robot.

*Point* Specifies the target point by using a SpelPoint data type.

*PointExpr* Specifies the target point by using a string expression.

**Return Value**

True if PTPBoost will be used, False if not.

**See Also**

PTPBoost

**PTPBoostOK Example****VB Example:**

```
If m_spel.PTPBoostOK(1) Then
    m_spel.Go(1)
End If
```

**C# Example:**

```
if (m_spel.PTPBoostOK(1))
    m_spel.Go(1);
```

**PTran Method, Spel Class**

**Description**

Executes a relative joint move in pulses.

**Syntax**

Sub **PTran** (*JointNumber* As Integer, *Pulses* As Integer)

**Parameters**

*JointNumber*     The specific joint to move.

*Pulses*             The number of pulses to move.

**See Also**

JTran, Pulse

**PTran Example**

**VB Example:**

```
' Move joint 1 5000 pulses in the plus direction.  
m_spel.PTran(1, 5000)
```

**C# Example:**

```
//Move joint 1 5000 pulses in the plus direction.  
m_spel.PTran(1, 5000);
```

## PTRQ Method, Spel Class

**Description**

Returns the peak torque for the specified joint.

**Syntax**

Function PTRQ (*JointNumber* As Integer) As Single

**Parameters**

*JointNumber* Integer expression from 1-9 representing the joint number.

**Return Value**

Real value from 0 to 1.

**See Also**

ATCLR, ATRQ, PTCLR

**PTRQ Example****VB Example:**

```
Dim peakTorque As Single  
peakTorque = m_spel.PTRQ(1)
```

**C# Example:**

```
float peakTorque;  
peakTorque = m_spel.PTRQ(1);
```

## Pulse Method, Spel Class

**Description**

Moves the robot arm by Point to Point control to the point specified by the pulse values for all robot joints.

**Syntax**

Sub **Pulse** ( *J1Pulses* As Integer, *J2Pulses* As Integer, *J3Pulses* As Integer,  
*J4Pulses* As Integer [, *J5Pulses* As Integer ] [, *J6Pulses* As Integer]  
[, *J7Pulses* As Integer] [, *J8Pulses* As Integer] [, *J9Pulses* As Integer] )

**Parameters**

*J1Pulses* – *J9Pulses* Integer expression containing the pulse value for joints 1 – 9.  
Joints 5 – 9 are optional.

---

Note: The pulse values must be within the range specified each joint.

---

**See Also**

Go, Move, Jump

**Pulse Example****VB Example:**

```
m_spel.Pulse(5000, 1000, 0, 0)
```

**C# Example:**

```
m_spel.Pulse(5000, 1000, 0, 0);
```

---

**Quit Method, Spel Class****Description**

Terminates execution of the specified task.

**Syntax**

Sub **Quit** (*TaskNumber* As Integer)

Sub **Quit** (*TaskName* As String)

**Parameters**

*TaskNumber*     The task number of the task to be interrupted.  
The range of the task number is 1 to 32.

*TaskName*     A string expression containing the name of the task.

**See Also**

Halt, Resume, Xqt

**Quit Example****VB Example:**

```
m_spel.Quit(3)
```

**C# Example:**

```
m_spel.Quit(3);
```

RadToDeg Method, Spel Class

**Description**

Converts Radians into Degrees.

**Syntax**

Function **RadToDeg** (*Radians* As Double) As Double

**Parameters**

*Radians* Double expression containing the radians to convert into degrees.

**Return Value**

Double containing the converted value in degrees.

**See Also**

DegToRad

**RadToDeg Example**

**VB Example:**

```
Dim deg As Double
```

```
deg = m_spel.RadToDeg(1)
```

**C# Example:**

```
double deg;
```

```
deg = m_spel.RadToDeg(1);
```

**RebootController Method, Spel Class****Description**

Reboot the currently connected controller.

**Syntax**

Sub **RebootController** (*ShowStatusDialog* As Boolean)

**Parameters**

*ShowStatusDialog* Boolean expression that specifies whether to show a status dialog or not.  
True = show a dialog False = do not show a dialog

**Remarks**

Use ShowStatusDialog to show a dialog with a progress bar. You can abort the operation from the status dialog or by executing the Abort method.

**See Also**

Abort

**RebootController Example****VB Example:**

```
m_spel.RebootController(True)
```

**C# Example:**

```
m_spel.RebootController(true);
```

RebuildProject Method, Spel Class

**Description**

Completely rebuilds the current project specified in the Project property.

**Syntax**

Sub **RebuildProject** ()

**See Also**

BuildProject, EnableEvent, EventReceived, Project, ProjectBuildComplete

**RebuildProject Example**

**VB Example:**

```
With m_spel
    .Project = "c:\EpsonRC70\projects\myproject\myproject.sprj"
    .RebuildProject()
End With
```

**C# Example:**

```
m_spel.Project =
@"c:\EpsonRC70\projects\myproject\myproject.sprj";
m_spel.RebuildProject();
```



---

**Recover Method, Spel Class****Description**

Recover moves the robot back to the position it was in when the safeguard was open.

**Syntax**

Function **Recover ()** As Boolean

**Remarks**

The Recover method can be used after the safeguard is closed to turn on the robot motors and slowly move the robot back to the position it was in when the safeguard was open. After Recover has completed successfully, you can execute the Cont method to continue the cycle.

If Recover was completed successfully, it will return True. Recover will return False if a pause, abort, or safeguard open occurred during recover motion.

**Return Value**

True if the recover motion was completed, False if not.

**See Also**

Continue, Pause

## Recover Example

### VB Examples:

This example executes a recover, then continue

```
Sub btnCont_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnCont.Click
    Dim sts As Boolean
    Dim answer As Integer

    sts = m_spel.Recover()
    If sts = False Then
        Exit Sub
    End If
    answer = MsgBox("Ready to continue?", vbYesNo)
    If answer = vbYes Then
        m_spel.Continue()
    EndIf
End sub
```

This example shows how a button can be used to execute recover as long as the button is down. If the button is released during recover motion, a pause is issued and recover is aborted. If the button is held down until recover has completed, then a message is displayed.

```
Sub btnRecover_MouseDown( _
    ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.MouseEventArgs) _
    Handles btnRecover.MouseDown
    Dim sts As Boolean

    sts = m_spel.Recover()
    If sts = True Then
        MsgBox("Recover complete")
    EndIf
End Sub

Sub btnRecover_MouseUp( _
    ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.MouseEventArgs) _
    Handles btnRecover.MouseUp

    m_spel.Pause()
End Sub
```

**C# Examples:**

This example executes a recover, then continue

```
void btnCont_Click(object sender, EventArgs e)
{
    bool sts;
    DialogResult answer;

    sts = m_spel.Recover();
    if (sts == true){
        answer = MessageBox.Show("Continue?", "",
            MessageBoxButtons.YesNo);
        If (answer == DialogResult.Yes)
            m_spel.Continue();
    }
}
```

This example shows how a button can be used to execute recover as long as the button is down. If the button is released during recover motion, a pause is issued and recover is aborted. If the button is held down until recover has completed, then a message is displayed.

```
void btnCont_Click(object sender, EventArgs e)
{
    bool sts;

    sts = m_spel.Recover();
    if (sts == true)
        MessageBox.Show("Recover complete");
}

void btnRecover_MouseUp(object sender, EventArgs e)
{
    m_spel.Pause();
}
```

Reset Method, Spel Class

**Description**

Resets the Controller to the initialized state.

**Syntax**

Sub **Reset** ()

**See Also**

ResetAbort

**Reset Example**

**VB Example:**

```
m_spel.Reset ()
```

**C# Example:**

```
m_spel.Reset ();
```

---

**ResetAbort Method, Spel Class**

---

**Description**

Resets the abort flag that is set with the Stop method.

**Syntax**

```
Sub ResetAbort ()
```

**Remarks**

When the Stop method is executed and no other Spel method is in cycle, then the next Spel method will generate a user abort error. This is done so that no matter when the Stop is issued, the routine that is executing Spel methods will receive the error. Use **ResetAbort** to clear this condition.

---

Note: The ResetAbortEnabled property must be set to True for the ResetAbort feature to work.

---

**See Also**

Abort, Reset, ResetAbortEnabled

**ResetAbort Example****VB Example:**

```
Sub btnMcal_Click() Handles btnMcal.Click
    m_spel.ResetAbort()
    m_spel.MCal()
End Sub
```

**C# Example:**

```
void btnMCal_Click(object sender, EventArgs e)
{
    m_spel.ResetAbort();
    m_spel.MCal();
}
```

Resume Method, Spel Class

**Description**

Resumes a task which was suspended by the Halt method.

**Syntax**

Sub **Resume** (*TaskNumber* As Integer)

Sub **Resume** (*TaskName* As String)

**Parameters**

*TaskNumber*     The task number of the task that was interrupted. The range of the task number is 1 to 32.

*TaskName*       A string expression containing the name of the task.

**See Also**

Quit, Xqt

**Resume Example**

**VB Example:**

```
m_spel.Resume(2)
```

**C# Example:**

```
m_spel.Resume(2);
```

## RunDialog Method, Spel Class

**Description**

Runs an EPSON RC+ 7.0 dialog.

**Syntax**

Sub **RunDialog** (*DialogID* As SpelDialogs [, *Parent* As Form])

**Parameters**

*DialogID*        The ID of the EPSON RC+ 7.0 dialog to run.

*Parent*         Optional. A .NET form that will be the parent of the window.

**See Also**

ShowWindow

**RunDialog Example****VB Example:**

```
Sub btnRobotManager_Click( _  
    ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) _  
    Handles btnRobotManager.Click  
  
    m_spel.RunDialog(SpelDialogs.RobotManager)  
End Sub
```

**C# Example:**

```
void btnRobotManager_Click(object sender, EventArgs e)  
{  
    m_spel.RunDialog(SpelDialogs.RobotManager);  
}
```

SavePoints Method, Spel Class

**Description**

Save points for the current robot into a file.

**Syntax**

Sub **SavePoints** (*FileName* As String)

**Parameters**

*FileName*        The file name to save the points in the current project.

**See Also**

LoadPoints

**SavePoints Example**

**VB Example:**

```
With m_spel  
    .SavePoints ("part1.pts")  
End With
```

**C# Example:**

```
m_spel.SavePoints ("part1.pts");
```



## Sense Method, Spel Class

**Description**

Specifies input condition that, if satisfied, completes the Jump in progress by stopping the robot above the target position.

**Syntax**

Sub **Sense** (*Condition* As String) As Boolean

**Parameters**

*Condition* Specifies the I/O condition.  
For details, see *Sense Statement* in *SPEL+ Language Reference manual*.

**See Also**

Jump, JS

**Sense Example****VB Example:**

```
With m_spel
    .Sense("Sw(1) = On")
    .Jump("P1 SENSE")
    stoppedOnSense = .JS()
End With
```

**C# Example:**

```
m_spel.Sense("Sw(1) = On");
m_spel.Jump("P1 SENSE");
stoppedOnSense = m_spel.JS();
```

## SetIODef Method, Spel Class

**Description**

Sets the I/O label and description for an input, output, or memory I/O bit, byte, or word.

**Syntax**

Sub **SetIODef** (*Type* As SpellLabelTypes, *Index* As Integer, *Label* As String, *Description* As String)

**Parameters**

*Type* Specifies the I/O type as shown below:  
 InputBit = 1    InputByte = 2    InputWord = 3  
 OutputBit = 4    OutputByte = 5    OutputWord = 6  
 MemoryBit = 7    MemoryByte = 8    MemoryWord = 9  
 InputReal = 10    OutputReal = 11

*Index* Specifies the bit or port number.

*Label* Specifies the new label.

*Description* Specifies the new description.

**Remarks**

Use SetIODef to define the label and description for any I/O point.

**See Also**

GetIODef

**SetIODef Example****VB Example:**

```
Dim label, desc As String
label = "StartCycle"
desc = "Starts the robot cycle"
m_spel.SetIODef(SpellLabelTypes.InputBit, 0, label, desc)
```

**C# Example:**

```
string label, desc;
label = "StartCycle";
desc = "Starts the robot cycle";
m_spel.SetIODef(SpellLabelTypes.InputBit, 0, label, desc);
```

## SetPoint Method, Spel Class

**Description**

Sets the coordinate data for a point for the current robot.

**Syntax**

Sub **SetPoint**(*PointNumber* As Integer, *Point* As SpelPoint)

Sub **SetPoint**(*PointLabel* As String, *Point* As SpelPoint)

Sub **SetPoint**(*PointNumber* As Integer, *X* As Single, *Y* As Single, *Z* As Single, *U* As Single)

Sub **SetPoint**(*PointLabel* As String, *X* As Single, *Y* As Single, *Z* As Single, *U* As Single)

Sub **SetPoint**(*PointNumber* As Integer, *X* As Single, *Y* As Single, *Z* As Single, *U* As Single,

*Local* As Integer, *Hand* As SpelHand)

Sub **SetPoint**(*PointLabel* As String, *X* As Single, *Y* As Single, *Z* As Single, *U* As Single, *Local* As Integer, *Hand* As SpelHand)

Sub **SetPoint**(*PointNumber* As Integer, *X* As Single, *Y* As Single, *Z* As Single, *U* As Single,

*V* As Single, *W* As Single)

Sub **SetPoint**(*PointLabel* As String, *X* As Single, *Y* As Single, *Z* As Single, *U* As Single, *V* As Single, *W* As Single)

Sub **SetPoint**(*PointNumber* As Integer, *X* As Single, *Y* As Single, *Z* As Single, *U* As Single,

*V* As Single, *W* As Single, *Local* As Integer, *Hand* As SpelHand, *Elbow* As SpelElbow, *Wrist* As SpelWrist, *J4Flag* As Integer, *J6Flag* As Integer)

Sub **SetPoint**(*PointLabel* As String, *X* As Single, *Y* As Single, *Z* As Single, *U* As Single, *V* As Single, *W* As Single, *Local* As Integer, *Hand* As SpelHand, *Elbow* As SpelElbow, *Wrist* As SpelWrist, *J4Flag* As Integer, *J6Flag* As Integer)

Sub **SetPoint**(*PointNumber* As Integer, *X* As Single, *Y* As Single, *Z* As Single, *U* As Single,

*V* As Single, *W* As Single, *S* As Single, *T* As Single)

Sub **SetPoint**(*PointLabel* As String, *X* As Single, *Y* As Single, *Z* As Single, *U* As Single, *V* As Single, *W* As Single, *S* As Single, *T* As Single)

Sub **SetPoint**(*PointNumber* As Integer, *PointExpr* As String)

Sub **SetPoint**(*PointLabel* As String, *PointExpr* As String)

**Parameters**

<i>PointNumber</i>	Integer expression that specifies the point number for a point in the current robot's point memory.
<i>X</i>	The X coordinate for the specified point.
<i>Y</i>	The Y coordinate for the specified point.
<i>Z</i>	The Z coordinate for the specified point.
<i>U</i>	The U coordinate for the specified point.
<i>V</i>	The V coordinate for the specified point.
<i>W</i>	The W coordinate for the specified point.
<i>S</i>	The S coordinate for the specified point.
<i>T</i>	The T coordinate for the specified point.
<i>Local</i>	The Local Number for the specified point. Use 0 when there is no local.
<i>Hand</i>	The hand orientation of the specified point.
<i>Elbow</i>	The elbow orientation of the specified point.
<i>Wrist</i>	The wrist orientation of the specified point.
<i>PointExpr</i>	Specifies the point by using a string expression.

**Note**

Do not enter integer values to X, Y, Z, U, V, W, S, and T parameters. Use Single variables or directly enter Single type values.

**See Also**

GetPoint, LoadPoints, SavePoints

**SetPoint Example****VB Example:**

```
Dim pt As SpelPoint
' Get coordinates of P1
pt = m_spel.GetPoint(1)
' Set it with changes
pt.U = pt.U - 10.5
m_spel.SetPoint(1, pt)
```

**C# Example:**

```
SpelPoint pt;
// Get coordinates of P1
pt = m_spel.GetPoint(1);
// Set it with changes
pt.U = pt.U - 10.5;
m_spel.SetPoint(1, pt);
```

## SetVar Method, Spel Class

**Description**

Sets the value of a SPEL+ global preserve variable in the Controller.

**Syntax**

Sub **SetVar** (*VarName* As String, *Value* As Object)

**Parameters**

*VarName*            The name of the SPEL+ global preserve variable.

*Value*              The new value.

**Remarks**

You can use SetVar to set the values for single variables and array variables. See the examples below.

**See Also**

GetVar

**SetVar Example****VB Example:**

```
m_spel.SetVar("g_myIntVar", 123)

Dim i, myArray(10) As Integer
For i = 1 To 10
    myArray(i) = i
Next i
m_spel.SetVar("g_myIntArray", myArray)

m_spel.SetVar("g_myIntArray(1)", myArray(1))
```

**C# Example:**

```
m_spel.SetVar("g_myIntVar", 123);

int[] myArray = new int[10];
for(int i = 1; i < 10; i++)
    myArray[i] = i;

m_spel.SetVar("g_myIntArray", myArray);

m_spel.SetVar("g_myIntArray[1]", myArray[1]);
```

SFree Method, Spel Class

**Description**

Free joint the specified robot axes.

**Syntax**

Sub **SFree** ()

Sub **SFree** (ParamArray *Axes*() As Integer)

**Parameters**

*Axes* An integer parameter array containing one element for each robot axis to free.  
You can specify axis numbers from 1 – 9.

**See Also**

SLock

**SFree Example**

**VB Example:**

```
' State J1 and J2 to free joint  
m_spel.SFree (1, 2)
```

**C# Example:**

```
// State J1 and J2 to free joint  
m_spel.SFree (1, 2);
```

## ShowWindow Method, Spel Class

**Description**

Shows an EPSON RC+ 7.0 window.

**Syntax**

Sub **ShowWindow** (*WindowID* As SpelWindows [, *Parent* As Form])

**Parameters**

*WindowID*      The ID of the EPSON RC+ 7.0 window to show.

*Parent*          Optional. A .NET form that will be the parent of the window.

**Remarks**

You can use the Parent parameter to specify the .NET parent form for the window. If you cannot use a .NET parent form, you must omit the Parent parameter and use the ParentWindowHandle property to set the handle of the parent.

**See Also**

HideWindow, ParentWindowHandle, RunDialog, ServerOutOfProcess

**ShowWindow Example****VB Example:**

```
Sub btnShowIOMonitor_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnShowIOMonitor.Click

    m_spel.ShowWindow(RCAPINet.SpelWindows.IOMonitor, Me)
End Sub
```

**C# Example:**

```
void btnShowIOMonitor_Click(object sender, EventArgs e)
{
    m_spel.ShowWindow(RCAPINet.SpelWindows.IOMonitor, this);
}
```

## SimGet Method, Spel Class

**Description**

Acquire the setting values of each object properties of simulator.

**Syntax**

Sub **SimGet** (*Object* As String, *Property* As SpelSimProps, ByRef *Value* As Boolean)

Sub **SimGet** (*RobotName* As String, *HandName* As String, *Property* As SpelSimProps, ByRef *Value* As Boolean)

Sub **SimGet** (*Object* As String, *Property* As SpelSimProps, ByRef *Value* As Double)

Sub **SimGet** (*RobotName* As String, *HandName* As String, *Property* As SpelSimProps, ByRef *Value* As Double)

Sub **SimGet** (*Object* As String, *Property* As SpelSimProps, ByRef *Value* As Integer)

Sub **SimGet** (*RobotName* As String, *HandName* As String, *Property* As SpelSimProps, ByRef *Value* As Integer)

Sub **SimGet** (*Object* As String, *Property* As SpelSimProps, ByRef *Value* As Boolean)

Sub **SimGet** (*RobotName* As String, *HandName* As String, *Property* As SpelSimProps, ByRef *Value* As String)

**Parameters**

<i>Object</i>	String variable that indicates object names acquiring the property values.
<i>RobotName</i>	String variable that indicates the robot name which the hand specified by “Hand” is installed
<i>HandName</i>	String variable that indicates the hand name which acquires the property values.
<i>Property</i>	Property name that acquires values. Descriptions of properties are described later.
<i>Value</i>	Variable that shows a returned value

**Remarks**

Use this method to acquire the property setting value of each object of the simulator.

For each property details, refer to the following manual.

*EPSON RC+ Language Reference SimGet*

**See Also**

SimSet

**SimGet Example****VB Example:**

```
Dim posX As Double
m_spel.SimGet("SBox_1", SpelSimProps.PositionX, posX)
```

**C# Example:**

```
double posX;
SimGet("SBox_1", SpelSimProps.PositionX, out posX);
```



**SimResetCollision Method, Spel Class****Description**

Resets collision detection.

**Syntax**

Sub **SimResetCollision** ()

**Remarks**

If there is no collision between the robot and the object after executing this method, the collision state is reset and the 3D display of simulator is updated.

If there is collision between the robot and the object, the collision state will not be reset and the 3D display of simulator will not be updated.

For more details, refer to the following manual.

*EPSON RC+ 7.0 SPEL+ Language Reference SimSet*

**See Also**

SimSet

**SimResetCollision Example****VB Example:**

```
m_spel.SimResetCollision ()
```

**C# Example:**

```
m_spel.SimResetCollision ();
```

## SimSet Method, Spel Class

**Description**

Set properties of each object of the simulator. Operate the robot motion, objects, and simulator settings.

**Syntax**

Sub **SimSet** (*Object* As String, *Property* As SpelSimProps, *Value* As Boolean)

Sub **SimSet** (*RobotName* As String, *HandName* As String, *Property* As SpelSimProps, *Value* As Boolean)

Sub **SimSet** (*Object* As String, *Property* As SpelSimProps, *Value* As Integer)

Sub **SimSet** (*RobotName* As String, *HandName* As String, *Property* As SpelSimProps, *Value* As Integer)

Sub **SimSet** (*Object* As String, *Property* As SpelSimProps, *Value* As Double)

Sub **SimSet** (*RobotName* As String, *HandName* As String, *Property* As SpelSimProps, *Value* As Double)

Sub **SimSet** (*Object* As String, *Property* As SpelSimProps, *Value* As String)

Sub **SimSet** (*RobotName* As String, *HandName* As String, *Property* As SpelSimProps, *Value* As String)

**Parameters**

*Object* String variable that indicates object names setting the property values.

*RobotName* String variable that indicates robot name that the hand specified by “Hand” is mounted.

*HandName* String variable that indicates hand name that sets a property value.

*Property* Property name that sets values.

*Value* Formula with new values

**Remarks**

Use this method to set properties of each object of the simulator. Also, use the command to change the robot motion, objects, and simulator settings.



You cannot specify SpelSimProps.Type for the property.

For details of each property, refer to the following manual.

*EPSON RC+ Language Reference SimSet*

**See Also**

SimGet

**SimSet Example****VB Example:**

```
m_spel.SimSet("SBox_1", SpelSimProps.PositionX, 100.0)
```

**C# Example:**

```
m_spel.SimSet("SBox_1", SpelSimProps.PositionX, 100.0);
```

**SimSetParent Method, Spel Class****Description**

Sets an object operation.

**Syntax**

Sub **SimSetParent** (*Object* As String)

Sub **SimSetParent** (*Object* As String, *ParentObject* As String)

**Parameters**

*Object*               String variable representing object name that sets a parent object

*ParentObject*       String variable representing a parent object.

**Remarks**

Sets the object specified by "ParentObject" as parent object for the object specified by "Object". "ParentObject" can be omitted. In this case, the object specified by "Object" will be parent object. For example, if the object specified by "Object" is a child object of another object, the setting as a child object will be canceled.

If the object specified by "Object" is a part or an object set as an arm installation tool, you cannot specify parent object.

For the objects that can be specified as SetParent, refer to the following manual.

*EPSON RC+ Language Reference SimSet*

For camera objects, SetParent can be used only for objects that are set as fixed cameras.

**See Also**

SimSet

**SimSetParent Example****VB Example:**

```
m_spel.SimSetParent ("SBox_1")
```

**C# Example:**

```
m_spel.SimSetParent ("SBox_1");
```

## SimSetPick Method, Spel Class

**Description**

Picks an object using the specified robot.

**Syntax**

Sub **SimSetPick** (RobotName As String, Object As String)

Sub **SimSetPick** (RobotName As String, Object As String, *ToolNumber* As Integer)

**Parameters**

<i>RobotName</i>	String variable that specifies the robot used for the pick operation.
<i>Object</i>	String variable that specifies the object to be picked.
<i>ToolNumber</i>	Integer variable that specifies the tool number to use for the pick operation.

**Remarks**

The robot specified by *RobotName* grasps the object specified by *Object*.

Grasped object is registered as the part of the robot. Also, if any tool number is specified by *ToolNumber*, you can operate grasped motion by using the specified tool number. If the *ToolNumber* is omitted, use Tool 0 to operate grasped motion.

You cannot grasp the object that is already registered as the part or set as an arm installation tool. Also, you cannot grasp the camera..

For more details, refer to the following manual and section:

*EPSON RC+ Language Reference SimSet*

**See Also**

SimGet, SimSet, SimSetPlace

**SimSetPick Example****VB Example:**

```
m_spel.SimSetPick ("Robot1", "SBox_1")
```

**C# Example:**

```
m_spel.SimSetPick ("Robot1", "SBox_1");
```

**SimSetPlace Method, Spel Class****Description**

Places an object using the specified robot.

**Syntax**

Sub **SimSetPlace** (*RobotName* As String, *Object* As String)

**Parameters**

*RobotName*      String variable that specifies the robot used for the place operation.

*Object*            String variable that specifies the object to be placed.

**Remarks**

The robot specified by *RobotName* places the object specified by *Object*. The placed object is deregistered as part of the robot.

For more details, refer to the following manual and section:

*EPSON RC+ Language Reference SimSet*

**See Also**

SimGet, SimSet, SimSetPick

**SimSetPlace Example****VB Example:**

```
m_spel.SimSetPlace("Robot1", "SBox_1")
```

**C# Example:**

```
m_spel.SimSetPlace("Robot1", "SBox_1");
```

Shutdown Method, Spel Class

**Description**

Shutdown or restart Windows.

**Syntax**

Sub **Shutdown** (*Mode* As SpelShutdownMode)

**Parameters**

*Mode*            0 = Shutdown Windows.  
                  1 = Restart Windows.

**See Also**

Reset

**Shutdown Example**

**VB Example:**

```
' Restart Windows  
m_spel.Shutdown(1)
```

**C# Example:**

```
// Restart Windows  
m_spel.Shutdown(1);
```

**SLock Method, Spel Class****Description**

Release the free joint state for the specified robot axes.

**Syntax**

Sub **SLock** ()

Sub **SLock** (ParamArray *Axes*() As Integer)

**Parameters**

*Axes* An integer parameter array containing one element for each robot axis to lock. You can specify axis numbers from 1 – 9.

**See Also**

SFree

**SLock Example****VB Example:**

' Release the free joint state of J1 and J2.

```
m_spel.SLock (1, 2)
```

**C# Example:**

// Release the free joint state of J1 and J2.

```
m_spel.SLock (1, 2);
```

Speed Method, Spel Class

**Description**

Specifies the arm speed for use with the point to point instructions Go, Jump and Pulse.

**Syntax**

Sub **Speed** ( *PointToPointSpeed* As Integer [, *JumpDepartSpeed* As Integer ]  
[, *JumpApproSpeed* As Integer] )

**Parameters**

*PointToPointSpeed* Specifies the arm speed for use with the point to point instructions Go, Jump and Pulse.

*JumpDepartSpeed* Integer number between 1-100 representing the Z axis upward motion speed for the Jump instruction.

*JumpApproSpeed* Integer number between 1-100 representing the Z axis downward motion speed for the Jump instruction.

**See Also**

Accel, Jump, Go

**Speed Example**

**VB Example:**

```
m_spel.Speed(50)
```

**C# Example:**

```
m_spel.Speed(50);
```



---

**SpeedR Method, Spel Class****Description**

Specifies the tool rotation speed when ROT is used.

**Syntax**

Sub **SpeedR** (*RotationSpeed* As Single)

**Parameters**

*RotationSpeed* Specifies the tool rotation speed in degrees / second.

**See Also**

Arc, Arc3, BMove, Jump3CP, Power, TMove

**SpeedR Example****VB Example:**

```
m_spel.SpeedR(100)
```

**C# Example:**

```
m_spel.SpeedR(100);
```

**SpeedS Method, Spel Class****Description**

Specifies the arm speed for use with the Continuous Path instructions Jump3CP, Move, Arc, and CVMove.

**Syntax**

Sub **SpeedS** ( *LinearSpeed* As Single [, *JumpDepartSpeed* As Single] [, *JumpApproSpeed* As Single] )

**Parameters**

*LinearSpeed* Specifies the arm speed for use with the Continuous Path instructions Jump3CP, Move, Arc, and CVMove.

*JumpDepartSpeed* A real number or formula defining depart speed for Jump, Jump3CP. (unit: mm/sec)

*JumpApproSpeed* A real number or formula defining approach speed for Jump, Jump3CP. (unit: mm/sec)

**See Also**

AccelS, Jump3CP, Move, TMove

**SpeedS Example****VB Example:**

```
m_spel.SpeedS(500)
```

**C# Example:**

```
m_spel.SpeedS(500);
```

## Start Method, Spel Class

**Description**

Start one SPEL<sup>+</sup> program.

**Syntax**

Sub **Start** (*ProgramNumber* As Integer)

**Parameters**

*ProgramNumber* The program number to start, corresponding to the 64 main functions in SPEL<sup>+</sup> as shown in the table below. The range is 0 to 63.

Program Number	SPEL+ Function Name
0	main
1	main1
2	main2
3	main3
4	main4
5	main5
...	...
63	main63

**Remarks**

When **Start** is executed, control will return immediately to the calling program. You cannot start a program that is already running. Note that Start causes global variables in the controller to be cleared and default robot points to be loaded.

**See Also**

Continue, Pause, Stop, Xqt

**Start Example****VB Example:**

```
Sub btnStart_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnStart.Click

    m_spel.Start(0)
End Sub
```

**C# Example:**

```
void btnStart_Click(object sender, EventArgs e)
{
    m_spel.Start(0);
}
```

## StartBGTask Method, Spel Class

**Description**

Start one SPEL<sup>+</sup> task as a background task.

**Syntax**

Sub **StartBGTask** (*FuncName* As String)

**Parameters**

*FuncName*      The name of the function to be executed.

**Remarks**

Use StartBGTask to start a Spel+ background task in the Controller. Background tasks must be enabled in the Controller.

Note that BGMain automatically starts when the Controller switches to auto mode, so normally StartBGTask is not required. StartBGTask is provided in case you need to stop all tasks, then start background tasks again.

**See Also**

Call, Start, Stop, Xqt

**StartBGTask Example****VB Example:**

```
' Stop all tasks, including background tasks
m_spel.Stop(SpelStopType.StopAllTasks)
...
m_spel.RebuildProject()

' Start the main background task
m_spel.StartBGTask("BGMain")
```

**C# Example:**

```
// Stop all tasks, including background tasks
m_spel.Stop(SpelStopType.StopAllTasks);
...
m_spel.RebuildProject();

// Start the main background task
m_spel.StartBGTask("BGMain");
```

## Stat Method, Spel Class

**Description**

Returns the Controller status.

**Syntax**

Function **Stat** (*Address* As Integer) As Integer

**Parameters**

*Address* Specifies the address representing the status of the Controller.  
(integer from 0 to 2)

**Return Value**

Returns 4 byte value representing the status of the Controller. (See the table below.)

Address	Bit		The status of the Controller while the bit is on	
0	0-15	&H1-&H8000	Task 1 to 16 are being executed (Xqt) or Halt state	
	16	&H10000	The task is being executed	
	17	&H20000	Pause state	
	18	&H40000	Error state	
	19	&H80000	TEACH mode	
	20	&H100000	Emergency stop state	
	21	&H200000	Power Low mode	
	22	&H400000	The safeguard is open	
	23	&H800000	Enable switch is open	
	24	&H1000000	Undefined	
	25	&H2000000	Undefined	
	26	&H4000000	Test mode	
	27	&H8000000	T2 mode state	
	28-31		Undefined	
	1	0	&H1	Log of Stop above target position upon satisfaction of condition in Jump...Sense statement. (This log is erased when another Jump statement is executed).
		1	&H2	Log of stop at intermediate travel position upon satisfaction of condition in Go/Jump/Move...Till statement. (This log is erased when another Go/Jump/Move...Till statement is executed)
		2	&H4	Undefined
		3	&H8	The log of motion stop in progress if Trap statement is detected.
		4	&H10	Motor On state
		5	&H20	Home position
		6	&H40	Power Low mode
		7	&H80	Undefined
		8	&H100	Joint #4 is engaged.
		9	&H200	Joint #3 is engaged.
		10	&H400	Joint #2 is engaged.
		11	&H800	Joint #1 is engaged.
		12	&H1000	Joint #6 is engaged.
		13	&H2000	Joint #5 is engaged.
14		&H4000	T axis is engaged.	
15		&H8000	S axis is engaged.	
16	&H10000	Joint #7 is engaged.		

## 14. RCAPINet Reference

---

Address	Bit		The status of the Controller while the bit is on
	17-31		Undefined
2	0-15	&H1-&H8000	Task 17 to 32 are being executed (Xqt) or in Halt state

### See Also

EStopOn, PauseOn, SafetyOn

### Stat Example

#### VB Example:

```
Dim ctr_stat As Integer  
ctr_stat = m_spel.Stat(0)
```

#### C# Example:

```
int ctr_stat;  
ctr_stat = m_spel.Stat(0);
```

## Stop Method, Spel Class

### Description

Stops all normal SPEL+ tasks running in the Controller and optionally stop all background tasks.

### Syntax

```
Sub Stop ()
Sub Stop (SpelStopType StopType)
```

### Parameters

*StopType* Optional. Specifies whether to stop only normal tasks (StopNormalTasks) or all tasks (StopAllTasks).  
If omitted, StopNormalTasks is specified.

---

Note: If the Stop method is executed when ResetAbortEnabled is True, the error 10101 occurs when executing Start or Reset methods.  
To release the error, execute ResetAbort method after executing Stop method.

---

### See Also

Continue, Pause, ResetAbort, ResetAbortEnabled, Start, SpelStopType

### Stop Example

#### VB Example:

```
Sub btnStop_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnStop.Click

    m_spel.Stop()
End Sub
```

#### C# Example:

```
void btnStop_Click(object sender, EventArgs e)
{
    m_spel.Stop();
}
```

Sw Method, Spel Class

**Description**

Returns the selected input bit status.

**Syntax**

Function **Sw** (*BitNumber* As Integer) As Boolean

Function **Sw** (*Label* As String) As Boolean

**Parameters**

*BitNumber* Integer expression representing one of the standard or expansion inputs.

*Label* String expression containing an input bit label.

**Return Value**

True if the specified input bit is on, False if not.

**See Also**

In, InBCD, MemSw, Off, On, Oport

**Sw Example**

**VB Example:**

```
If m_spel.Sw(1) Then  
    m_spel.On(2)  
End If
```

**C# Example:**

```
if (m_spel.Sw(1))  
    m_spel.On(2);
```



## TargetOK Method, Spel Class

**Description**

Returns a status indicating whether or not the PTP (Point to Point) motion from the current position to a target position is possible.

**Syntax**

Function **TargetOK** (*PointNumber* As Integer) As Boolean

Function **TargetOK** (*Point* As SpelPoint) As Boolean

Function **TargetOK** (*PointExpr* As String) As Boolean

**Parameters**

Each syntax has one parameter that specifies the target point to check.

*PointNumber* Specifies the target point by using the point number for a previously taught point in the Controller's point memory for the current robot.

*Point* Specifies the target point by using a SpelPoint data type.

*PointExpr* Specifies the target point by using a string expression.

**Return Value**

True if the target can be moved to from the current position, False if not.

**See Also**

Go, Jump, Move, TGo, TMove

**TargetOK Example****VB Example:**

```
If m_spel.TargetOK("P1 /F") Then
    m_spel.Go("P1 /F")
End If
```

**C# Example:**

```
if (m_spel.TargetOK("P1 /F"))
    m_spel.Go("P1 /F");
```

TasksExecuting Method, Spel Class

**Description**

Returns True if any SPEL+ tasks are executing.

**Syntax**

Function **TasksExecuting** () As Boolean

**Return Value**

True if any SPEL+ tasks are executing, False if not.

**See Also**

TaskState, Xqt

**TasksExecuting Example**

**VB Example:**

```
tasksRunning = m_spel.TasksExecuting()
```

**C# Example:**

```
tasksRunning = m_spel.TasksExecuting();
```

**TaskState Method, Spel Class****Description**

Returns the status of a task.

**Syntax**

Function **TaskState** (*TaskNumber* As Integer) As SpelTaskState

Function **TaskState** (*TaskName* As String) As SpelTaskState

**Parameters**

*TaskNumber* Task Number to return the execution status of.

*TaskName* String expression containing the name of the task.

**Return Value**

A SpelTaskState value.

**See Also**

TasksExecuting, Xqt

**TaskState Example****VB Example:**

```
Dim taskState As SpelTaskState  
taskState = m_spel.TaskState(2)
```

**C# Example:**

```
SpelTaskState taskState;  
taskState = m_spel.TaskState(2);
```

## TeachPoint Method, Spel Class

**Description**

Runs a dialog that allows an operator to jog and teach one point.

**Syntax**

Function **TeachPoint** ( *PointFile* As String, *PointNumber* As Integer, *Prompt* As String )  
As Boolean

Function **TeachPoint** ( *PointFile* As String, *PointName* As String, *Prompt* As String ) As  
Boolean

Function **TeachPoint** ( *PointFile* As String, *PointNumber* As Integer, *Prompt* As String,  
*Parent* As Form ) As Boolean

Function **TeachPoint** ( *PointFile* As String, *PointName* As String, *Prompt* As String,  
*Parent* As Form ) As Boolean

**Parameters**

*PointFile*        A string containing the name of the point file.

*PointNumber*    The point number to teach.

*PointName*      A string expression for an existing point label.

*Prompt*          A string containing the instructional text that is displayed on the bottom  
of the teach dialog.

*Parent*          Optional. A .NET form that will be the parent of the window.

**Return Value**

Returns True if the operator clicked the Teach button, False if the operator clicked Cancel.

**Remarks**

Use TeachPoints to allow an operator to teach one robot point in the Controller. When TeachPoints is executed, the point file is loaded in the Controller. When the Teach button is clicked, the point is taught in the Controller and the point file is saved on the Controller.

## TeachPoint Example

### VB Example:

```
Sub btnTeachPick_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnTeachPick.Click

    Dim sts As Boolean
    Dim prompt As String

    prompt = "Jog to Pick position and click Teach"
    sts = m_spel.TeachPoint("points.pts", 1, prompt)
End Sub
```

### C# Example:

```
void btnTeachPick_Click(object sender, EventArgs e)
{
    bool sts;
    string prompt;

    prompt = "Jog to Pick position and click Teach";
    sts = m_spel.TeachPoint("points.pts", 1, prompt);
}
```

Till Method, Spel Class

**Description**

Specifies event condition that, if satisfied, completes the motion command (Jump, Go, Move, etc.) in progress by decelerating and stopping the robot at an intermediate position.

**Syntax**

Sub **Till** (*Condition* As String) As Boolean

**Parameters**

*Condition* Specifies the I/O condition. For details see *Till Statement* in *SPEL+ Language Reference manual*.

**See Also**

Go, Jump, JS, Sense, TillOn

**Till Example**

**VB Example:**

```
With m_spel
    .Till("Sw(1) = On")
    .Go("P1 TILL")
End With
```

**C# Example:**

```
m_spel.Till("Sw(1) = On");
m_spel.Go("P1 TILL");
```

**TillOn Method, Spel Class****Description**

Returns True if a stop has occurred from a till condition during the last Go/Jump/Move statement.

**Syntax**

Function **TillOn** () As Boolean

**Return Value**

True if the robot stopped due to a Till condition, False if not.

**Remarks**

Use TillOn to check if the Till condition turned on during the last motion command using Till.

TillOn is equivalent to ((Stat(1) And 2) <> 0)

**See Also**

Jump, Till

**TillOn Example****VB Example:**

```
If m_spel.TillOn() Then  
    m_spel.Jump(2)  
End If
```

**C# Example:**

```
if (m_spel.TillOn())  
    m_spel.Jump(2);
```

## TGo Method, Spel Class

**Description**

Executes Point to Point relative motion, in the current tool coordinate system.

**Syntax**

Sub **TGo** (*PointNumber* As Integer)

Sub **TGo** (*Point* As SpelPoint)

Sub **TGo** (*Point* As SpelPoint, *AttribExpr* As String)

Sub **TGo** (*PointExpr* As String)

**Parameters**

Each syntax has one parameter that specifies the end point which the arm travels to during the TGo motion. This is the final position at the end of the point to point motion.

*PointNumber* Specifies the end point by using the point number for a previously taught point in the Controller's point memory for the current robot.

*Point* Specifies the end point by using a SpelPoint data type.

*AttribExpr* Specifies the end point attributes by using a string expression.

*PointExpr* Specifies the end point by using a string expression.

**See Also**

Accel, Speed

Arc, Arc3, CVMove, Go, Jump, Jump3, Jump3CP, Move

BGo, BMove, TMove

CP, Till

**TGo Example****VB Example:**

```
' Point specified using point number
m_spel.Tool(1)
m_spel.TGo(100)

' Point specified using SpelPoint
Dim pt As SpelPoint
pt = m_spel.GetPoint("P*")
pt.X = 125.5
m_spel.TGo(pt)

' Point specified using expression
m_spel.TGo("P0 /L /2")
m_spel.TGo("P1 :Z(-20)")

' Using parallel processing
m_spel.TGo("P1 !D50; On 1; D90; Off 1!")

' Point specified using label
m_spel.TGo("pick")
```



**C# Example:**

```
// Point specified using point number
m_spel.Tool(1);
m_spel.TGo(100);

// Point specified using SpelPoint
SpelPoint pt;
pt = m_spel.GetPoint("P0");
pt.X = 125.5;
m_spel.TGo(pt);

// Point specified using expression
m_spel.TGo("P0 /L /2");
m_spel.TGo("P1 :Z(-20)");

// Using parallel processing
m_spel.TGo("P1 !D50; On 1; D90; Off 1!");

// Point specified using label
m_spel.TGo("pick");
```

TLClr Method, Spel Class

**Description**

Clears (undefines) a tool coordinate system.

**Syntax**

Sub **TLClr** (*ToolNumber* As Integer)

**Parameters**

*ToolNumber* Integer expression representing which of the tools to clear (undefine).  
(Tool 0 is the default tool and cannot be cleared.)

**See Also**

Tool, TLDef

**TLClr Example**

**VB Example:**

```
m_spel.TLClr(1)
```

**C# Example:**

```
m_spel.TLClr(1);
```

## TLDef Method, Spel Class

**Description**

Returns tool definition status.

**Syntax**

Function **TLDef** (*ToolNumber* As Integer) As Boolean

**Parameters**

*ToolNumber* Integer expression representing which tool to return status for.

**Return Value**

True if the specified tool is defined, False if not.

**See Also**

Tool, TLClr

**TLDef Example****VB Example:**

```
m_spel.TLDef(1)
```

**C# Example:**

```
m_spel.TLDef(1);
```

## TLSet Method, Spel Class

**Description**

Defines a tool coordinate system.

**Syntax**

Sub **TLset** (*ToolNumber* As Integer , *Point* As SpelPoint)

Sub **TLset** ( *ToolNumber* As Integer, *XCoord* As Single, *YCoord* As Single, *ZCoord* As Single,  
*UCoord* As Single, *VCoord* As Single, *WCoord* As Single )

**Parameters**

<i>ToolNumber</i>	Integer expression from 1-15 representing which of 15 tools to define. (Tool 0 is the default tool and cannot be modified.)
<i>Point</i>	A SpelPoint containing the point data.
<i>XCoord</i>	The tool coordinate system origin X coordinate.
<i>YCoord</i>	The tool coordinate system origin Y coordinate.
<i>ZCoord</i>	The tool coordinate system origin Z coordinate.
<i>UCoord</i>	The tool coordinate system rotation about the Z axis.
<i>VCoord</i>	The tool coordinate system rotation about the Y axis.
<i>WCoord</i>	The tool coordinate system rotation about the X axis.

**See Also**

Arm, Armset, GetTool, Tool

**TLSet Example****VB Example:**

```
m_spel.TLSet(1, .5, 4.3, 0, 0, 0, 0)
```

**C# Example:**

```
m_spel.TLSet(1, .5, 4.3, 0, 0, 0, 0);
```

## TMove Method, Spel Class

**Description**

Executes linear interpolation relative motion, in the current tool coordinate system

**Syntax**

Sub **TMove** (*PointNumber* As Integer)

Sub **TMove** (*Point* As SpelPoint)

Sub **TMove** (*Point* As SpelPoint, *AttribExpr* As String)

Sub **TMove** (*PointExpr* As String)

**Parameters**

Each syntax has one parameter that specifies the end point which the arm travels to during the TMove motion. This is the final position at the end of the linear interpolated motion.

*PointNumber* Specifies the end point by using the point number for a previously taught point in the Controller's point memory for the current robot.

*Point* Specifies the end point by using a SpelPoint data type.

*AttribExpr* Specifies the end point attributes by using a string expression.

*PointExpr* Specifies the end point by using a string expression.

**See Also**

AccelR, AccelS, SpeedR, SpeedS

Arc, Arc3, CVMove, Go, Jump, Jump3, Jump3CP, Move

BGo, BMove, TGo

CP, Till

**TMove Example****VB Example:**

```
' Point specified using point number
m_spel.Tool(1)
m_spel.TMove(100)

' Point specified using SpelPoint
Dim pt As SpelPoint
pt = m_spel.GetPoint("P*")
pt.X = 125.5
m_spel.TMove(pt)

' Point specified using expression
m_spel.TMove("P0")
m_spel.TMove("XY(0, 0, -20, 0)")

' Using parallel processing
m_spel.TMove("P1 !D50; On 1; D90; Off 1!")

' Point specified using label
m_spel.TMove("pick")
```

### C# Example:

```
// Point specified using point number
m_spel.Tool(1);
m_spel.TMove(100);

// Point specified using SpelPoint
SpelPoint pt;
pt = m_spel.GetPoint("P0");
pt.X = 125.5;
m_spel.TMove(pt);

// Point specified using expression
m_spel.TMove("P0");
m_spel.TMove("XY(0, 0, -20, 0)");

// Using parallel processing
m_spel.TMove("P1 !D50; On 1; D90; Off 1!");

// Point specified using label
m_spel.TMove("pick");
```

## Tool Method, Spel Class

**Description**

Selects the current robot tool.

**Syntax**

Sub **Tool** (*ToolNumber* As Integer)

**Parameters**

*ToolNumber* Integer number from 0-15 representing which of 16 tool definitions to use with the subsequent motion instructions.

**See Also**

TLSet, Arm, TGo, TMove

**Tool Example****VB Example:**

```
m_spel.Tool(1)
m_spel.TGo(100)
```

**C# Example:**

```
m_spel.Tool(1);
m_spel.TGo(100);
```

TrapStop Method, Spel Class

**Description**

Returns True if the current robot was stopped by a trap during the previous motion command.

**Syntax**

Function **TrapStop** () As Boolean

**Return Value**

True if the robot was stopped by a trap, False if not.

**See Also**

EStopOn, ErrorOn

**TrapStop Example**

**VB Example:**

```
If m_spel.TrapStop() Then  
    MsgBox "Robot stopped by Trap"  
End If
```

**C# Example:**

```
if (m_spel.TrapStop())  
    MessageBox.Show("Robot stopped by trap");
```



## TW Method, Spel Class

**Description**

Returns the status of the WAIT condition and WAIT timer interval.

**Syntax**

Function **TW** () As Boolean

**Return Value**

True if a timeout occurred, False if not.

**See Also**

WaitMem, WaitSw

**TW Example****VB Example:**

```
Const PartPresent = 1
m_spel.WaitSw(PartPresent, True, 5)
If m_spel.TW() Then
    MsgBox "Part present time out occurred"
End If
```

**C# Example:**

```
const int PartPresent = 1;
m_spel.WaitSw(PartPresent, True, 5);
if (m_spel.TW())
    MessageBox.Show("Part present time out occurred");
```

UserHasRight Method, Spel Class

**Description**

Returns whether the currently logged in user has the specified right.

**Syntax**

Function **UserHasRight** (SpelUserRights *Right*) As Boolean

**Parameters**

*Right*      The right you want to check for the current logged in user.

**Return Value**

True if the user has the specified right, False if not.

**See Also**

Login, GetCurrentUser

**UserHasRight Example**

**VB Example:**

```
Dim hasRight As Boolean  
hasRight = m_spel.UserHasRight(SpelUserRights.EditPoints)
```

**C# Example:**

```
bool hasRight;  
hasRight = m_spel.UserHasRight(SpelUserRights.EditPoints);
```

## VCal Method, Spel Class

**Description**

This command allows you to execute a vision calibration cycle.

**Syntax**

Sub **VCal** (*CalibName* As String)

Sub **VCal** (*CalibName* As String, *ByRef Status* As Integer)

Sub **VCal** (*CalibName* As String, *Parent* As Form)

Sub **VCal** (*CalibName* As String, *Parent* As Form, *ByRef Status* As Integer)

**Parameters**

*CalibName*            A string expression that evaluates to the name of a calibration scheme in the current project.

*Status*                Optional. An integer variable that receives the status of the calibration.  
0: Unsuccessful, 1: Successful

*Parent*                Optional. .NET parent form.

**Remarks**

When you execute the **VCal** method, the robot will move. You should verify that the operator is ready before executing **VCal**.

**VCal** only executes the calibration cycle. It does not allow you to teach points. Use **VCalPoints** to teach points. Also, you must first set up a calibration in EPSON RC+ 7.0. See your Vision Guide manuals for details.

Use the *Status* parameter to check if the calibration was successful. If the calibration property *ShowConfirmation* is *True*, the confirmation dialog box is displayed. When the operator clicks the <OK> button, *Status* returns 1: Successful.

**See Also**

**VCalPoints**

**VCal Example****VB Example:**

```
Dim status As Integer
m_spel.VCal ("CAMCAL1", status)
```

**C# Example:**

```
int status;
m_spel.VCal ("CAMCAL1", status);
```

## VCalPoints Method, Spel Class

**Description**

This command enables you to teach vision calibration points.

**Syntax**

Sub **VCalPoints** (*CalibName* As String)

Sub **VCalPoints** (*CalibName* As String, *ByRef Status* As Integer)

Sub **VCalPoints** (*CalibName* As String, *Parent* As Form)

Sub **VCalPoints** (*CalibName* As String, *ByRef Status* As Integer, *Parent* As Form)

**Parameters**

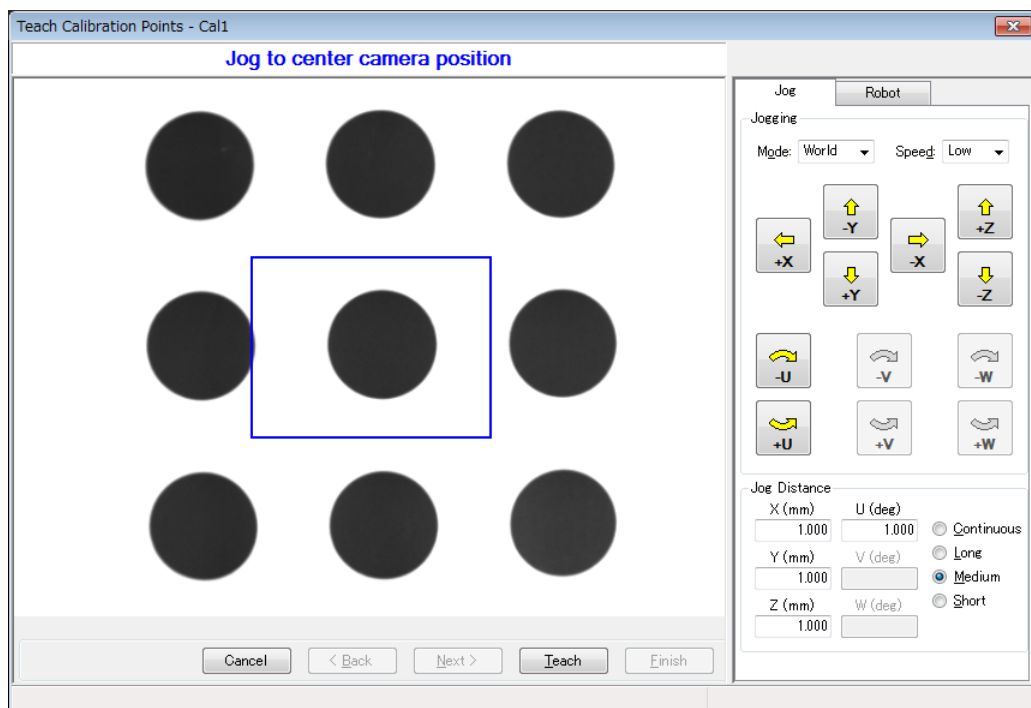
*CalibName* A string expression that evaluates to the name of a calibration scheme in the current project.

*Status* Optional. An integer variable that receives the status of the point teaching.  
0: Unsuccessful, 1: Successful

*Parent* Optional. .NET parent form.

**Remarks**

When you execute the **VCalPoints** command, the Teach Calibration Points dialog is opened. When the <Finish> button is clicked, the calibration data is automatically saved. You must have already created the calibration scheme from EPSON RC+ 7.0.



Use the *Status* parameter to check if the point teaching was successful. When all points are taught and the <Finish> button is clicked, *Status* returns 1: Successful.

**See Also**

VCal

**VCalPoints Example****VB Example:**

```
Dim status As Integer  
m_spel.VCalPoints("CAMCAL1", status)
```

**C# Example:**

```
int status;  
m_spel.VCalPoints("CAMCAL1", out status);
```

VClS Method, Spel Class

**Description**

Clears vision graphics.

**Syntax**

Sub **VClS** ()

**Remarks**

Use the VClS method to clear the vision screen.

**See Also**

VRun

**VClS Example**

**VB Example:**

```
m_spel.VClS ()
```

**C# Example:**

```
m_spel.VClS ();
```

## VCreateCalibration Method, Spel Class

**Description**

Creates a new vision calibration in the current project.

**Syntax**

Sub **VCreateCalibration** (*CameraNumber* As Integer, *CalibName* As String)

Sub **VCreateCalibration** (*CameraNumber* As Integer, *CalibName* As String,  
*CopyCalibName* As String)

**Parameters**

*CameraNumber* Integer expression containing the number of the camera to be calibrated.

*CalibName* String expression containing the name of a vision calibration to create.

*CopyCalibName* Optional. String expression containing the name of a vision calibration to copy.

**See Also**

VCreateObject, VCreateSequence, VDeleteCalibration

**VCreateCalibration Example****VB Example:**

```
m_spel.VCreateCalibration(1, "mycal")
```

**C# Example:**

```
m_spel.VCreateCalibration(1, "mycal");
```

## VCreateObject Method, Spel Class

**Description**

Creates a vision object in the current project.

**Syntax**

Sub **VCreateObject** ( *Sequence* As String, *ObjectName* As String, *ObjectType* As SpelVisionObjectTypes )

**Parameters**

<i>Sequence</i>	String expression containing the name of a vision sequence in the current project.
<i>ObjectName</i>	String expression containing the name of an object to create in sequence <i>Sequence</i> .
<i>ObjectType</i>	A SpelVisionObjectTypes that specifies the vision object type. (Constants shown below are also available)

Object Type	Constant	Value
Correlation	Correlation	1
Blob	Blob	2
Edge	Edge	3
Polar	Polar	4
Line	Line	5
Point	Point	6
Frame	Frame	7
ImageOp	ImageOp	8
Ocr	Ocr	9
CodeReader	CodeReader	10
Geometric	Geometric	11
Color Match	ColorMatch	14
Line Finder	LineFinder	15
Arc Finder	ArcFinder	16
Defect Finder	DefectFinder	17
Line Inspector	LineInspector	18
Arc Inspector	ArcInspector	19
Box Finder	BoxFinder	20
Corner Finder	CornerFinder	21
Contour	Contour	22
Text	Text	23
Decision	Decision	26
Coordinates	Coordinates	27

**See Also**

VCreateSequence, VDeleteObject, VDeleteSequence

**VCreateObject Example****VB Example:**

```
m_spel.VCreateObject("myseq", "myblob",
SpelVisionObjectTypes.Blob)
```

**C# Example:**

```
m_spel.VCreateObject("myseq", "myblob",
SpelVisionObjectTypes.Blob);
```



## VCreateSequence Method, Spel Class

**Description**

Creates a new vision sequence in the current project.

**Syntax**

Sub **VCreateSequence** (*CameraNumber* As Integer, *SequenceName* As String)

Sub **VCreateSequence** (*CameraNumber* As Integer, *SequenceName* As String,  
*CopySequenceName* As String)

**Parameters**

*CameraNumber* Integer expression containing the number of the camera to be used.

*SequenceName* String expression containing the name of a vision sequence to create.

*CopySequenceName* Optional. String expression containing the name of a vision sequence to copy.

**See Also**

VCreateObject, VDeleteObject, VDeleteSequence

**VCreateSequence Example****VB Example:**

```
m_spel.VCreateSequence(1, "myseq")
```

**C# Example:**

```
m_spel.VCreateSequence(1, "myseq");
```

## VDefArm Method, Spel Class

**Description**

Calculates an arm set value of a mobile camera using a feature point detectable by the vision system.

Note:

Robot operates automatically based on the detection results of the target. Be careful of interference between the robot and peripherals. Also, use with avoiding singularity nearby posture that each axis extends to prevent an error during the arm set.

**Syntax**

Sub **VDefArm** (*ArmNumber* As Integer, *ArmDefType* As SpelArmDefType, *ArmDefMode* As SpelArmDefMode, *Sequence* As String, *Rotation* As Double, *TargetTolerance* As Double)

Sub **VDefArm** (*ArmNumber* As Integer, *ArmDefType* As SpelArmDefType, *ArmDefMode* As SpelArmDefMode, *Sequence* As String, *Rotation* As Double, *TargetTolerance* As Double, *Parent* As Form)

Sub **VDefArm** (*ArmNumber* As Integer, *ArmDefType* As SpelArmDefType, *ArmDefMode* As SpelArmDefMode, *Sequence* As String, *Rotation* As Double, *TargetTolerance* As Double, *RobotSpeed* As Integer, *RobotAccel* As Integer, *ShowWarning* As SpelVDefShowWarning)

Sub **VDefArm** (*ArmNumber* As Integer, *ArmDefType* As SpelArmDefType, *ArmDefMode* As SpelArmDefMode, *Sequence* As String, *Rotation* As Double, *TargetTolerance* As Double, *RobotSpeed* As Integer, *RobotAccel* As Integer, *ShowWarning* As SpelVDefShowWarning, *Parent* As Form)

**Parameters**

<i>ArmNumber</i>	Integer expression that contains the arm number to perform arm set (1 to 15).
<i>ArmDefType</i>	Integer expression that contains the arm type. J2Camera: Calculates a center of mobile J2 camera image.
<i>ArmDefMode</i>	Integer expression that contains the arm set mode. Rough: A mode to run a rough arm set. Robot will move with setting accuracy of 1 mm as a target. Robot motion will be small. Fine: A mode to run a fine arm set. Robot will move largely with arm orientation change and provide arm set with more high accuracy.
<i>Sequence</i>	String expression containing a vision sequence name of current project.
<i>Rotation</i>	Real expression that contains rotation angle (degrees) for a rough arm set. Value range: 0 to 45
<i>TargetTolerance</i>	Real expression containing a pixel distance to consider that the vision detection result matches the target position. Value range: 0 to 3 pixels
<i>Parent</i>	Optional. Parent .NET form of a window.
<i>RobotSpeed</i>	Optional. Integer variable that will contain the robot speed (%). Value range: 0 to 100 If omitted, set to "5".

<i>RobotAccel</i>	Optional. Integer variable that will contain the robot acceleration (%). Value range: 0 to 99 If omitted, set to "5".
<i>ShowWarning</i>	Optional. Integer variable that determines whether to display warning when <i>ArmSetMode</i> is Fine. Always : Always display DependsOnSpeed : Display when either <i>RobotSpeed</i> or <i>RobotAccel</i> is larger than 5. None : Do not display If omitted, set to "DependsOnSpeed".

**See Also**

VDefGetMotionRange, VDefLocal, VDefSetMotionRange, VDefTool, VGoCenter

**VDefArm Example****VB Example:**

```
m_spel.VDefArm(1, SpelArmDefType.J2Camera,
SpelArmDefMode.Rough, "myseq", 5, 1)
```

**C# Example:**

```
m_spel.VDefArm(1, SpelArmDefType.J2Camera,
SpelArmDefMode.Rough, "myseq", 5, 1);
```

## VDefGetMotionRange Method, Spel Class

**Description**

Acquires values of the motion range limited by VDefTool, VDefArm, VDefLocal, and VGoCenter.

**Syntax**

Sub **VDefGetMotionRange**(ByRef MaxMoveDist As Double, ByRef MaxPoseDiffAngle As Double, ByRef LjmMode As Integer)

**Parameters**

<i>MaxMoveDist</i>	Real variable representing the maximum distance of move. If 0 is specified, the range is not limited. ( 0 to 500. Default: 200) VDefTool, VDefArm, VDefLocal, and VGoCenter are used to limit the range.
<i>MaxPoseDiffAngle</i>	Real variable representing the maximum displacement angle (degrees) of tool orientation (UVW). If 0 is specified, the angle is not limited. It only affects VDefLocal. (0 to 180. Default: 45 degrees)
<i>LjmMode</i>	Integer variable representing the LJM mode.

**See Also**

VDefTool, VDefArm, VDefLocal, VGoCenter, VDefSetMotionRange

**VDefGetMotionRange Example****VB Example:**

```
Dim maxMoveDist As Double
Dim maxPoseDiffAngle As Double
Dim ljmMode As Integer
m_spel.VDefGetMotionRange(maxMoveDist, maxPoseDiffAngle,
ljmMode)
```

**C# Example:**

```
double maxMoveDist, maxPoseDiffAngle;
int ljmMode;
m_spel.VDefGetMotionRange(out maxMoveDist, out
maxPoseDiffAngle, out ljmMode);
```

## VDefLocal Method, Spel Class

**Description**

Detects a calibration plate placed on a work plane by a mobile camera, and defines local coordinates parallel to the work plane.

It also detects user's workpiece at the tool end by a fixed camera and defines a local plane which is parallel to a fixed camera sensor.

Note:

Robot operates automatically based on the detection results of the target. Be careful of interference between the robot and peripherals. Also, use with avoiding singularity nearby posture that each axis extends to prevent an error during the local coordinate setting.

**Syntax**

Sub **VDefLocal**(LocalNumber As Integer, LocalDefType As SpelLocalDefType, CalPlateType As SpelCalPlateType, Sequence As String, TargetTolerance As Double, CameraTool As Integer, RefPoint As SpelPoint)

Sub **VDefLocal**(LocalNumber As Integer, LocalDefType As SpelLocalDefType, CalPlateType As SpelCalPlateType, Sequence As String, TargetTolerance As Double, CameraTool As Integer, RefPoint As SpelPoint, Parent As Form)

Sub **VDefLocal**(*LocalNumber* As Integer, *LocalDefType* As SpelLocalDefType, *CalPlateType* As SpelCalPlateType, *Sequence* As String, *TargetTolerance* As Double, *CameraTool* As Integer, *RefPoint* As SpelPoint, *RobotSpeed* As Integer, *RobotAccel* As Integer)

Sub **VDefLocal**(*LocalNumber* As Integer, *LocalDefType* As SpelLocalDefType, *CalPlateType* As SpelCalPlateType, *Sequence* As String, *TargetTolerance* As Double, *CameraTool* As Integer, *RefPoint* As SpelPoint, *RobotSpeed* As Integer, *RobotAccel* As Integer, *Parent* As Form)

**Parameters**

<i>LocalNumber</i>	Integer representing a tool number to set local coordinates. (1-15)
<i>LocalDefType</i>	Integer representing a local type. J5Camera: Specifies local coordinates parallel to a calibration plate by using the mobile J5 camera. J6Camera: Specifies local coordinates parallel to a calibration plate by using the mobile J6 camera. FixedUpwardCamera: Specifies local coordinates parallel to an image sensor by using the upward fixed camera. FixedDownwardCamera: Specifies local coordinates parallel to an image sensor by using the downward fixed camera.
<i>CalPlateType</i>	Integer representing a type of calibration plate. Large : Large calibration plate Medium : Medium calibration plate Small : Small calibration plate XSmall : Extra small calibration plate
<i>Sequence</i>	String expression representing a vision sequence name of current project. When using the mobile camera, this is a vision sequence to take a picture of the calibration plate. When using the fixed camera, this is a vision sequence to detect a feature point at tool end, such as user's workpiece.

<i>TargetTolerance</i>	Real value representing a threshold value to judge scale coincidence.
<i>CameraTool</i>	Fixed camera: Specifies a tool number that holds a tool offset of the detection target. To perform auto calibration, specify -1.  Mobile J6 camera: If auto calibration has been executed, specify a tool number of mobile camera. To perform auto calibration, specify -1.  Mobile J5 camera: Setting of this option is ignored.
<i>RefPoint</i>	Point number which a local plane parallel to a work plane passes.  This point is used to specify local plane height.
<i>Parent</i>	Optional. Parent .NET form of a window.
<i>RobotSpeed</i>	Optional. Integer variable that will contain the robot speed (%). Value range: 0 to 100 If omitted, set to "5".
<i>RobotAccel</i>	Optional. Integer variable that will contain the robot acceleration (%). Value range: 0 to 99 If omitted, set to "5".

**See Also**

VDefArm, VDefGetMotionRange, VDefSetMotionRange, VDefTool, VGoCenter

**VDefLocal Example**

**VB Example:**

```
Dim p2 = m_spel.GetPoint("P2")
m_spel.VDefLocal(1, SpelLocalDefType.J6Camera,
SpelCalPlateType.Large, "myseq", 1.0, 1, p2)
```

**C# Example:**

```
SpelPoint p2;
p2 = m_spel.GetPoint("P2");
m_spel.VDefLocal(1, SpelLocalDefType.J6Camera,
SpelCalPlateType.Large, "myseq", 1.0, 1, p2);
```

## VDefSetMotionRange Method, Spel Class

**Description**

Limits a motion range by VDefTool, VDefArm, VDefLocal, and VGoCenter.

**Syntax**

Sub **VDefSetMotionRange**(MaxMoveDist As Double, MaxPoseDiffAngle As Double, LjmMode As Integer)

**Parameters**

<i>MaxMoveDist</i>	Real value representing the maximum distance of move. If 0 is specified, the range is not limited. (0 to 500. Default: 200) VDefTool, VDefArm, VDefLocal, and VGoCenter are used to limit the range.
<i>MaxPoseDiffAngle</i>	Real value representing the maximum displacement angle (degrees) of tool orientation (UVW). If 0 is specified, the angle is not limited. It only affects VDefLocal. (0 to 180. Default: 45 degrees)
<i>LjmMode</i>	Integer representing the LJM mode.

**See Also**

VDefTool, VDefArm, VDefLocal, VGoCenter, VDefGetMotionRange

**VDefSetMotionRange Example****VB Example:**

```
m_spel.VDefSetMotionRange(100, 30, 1)
```

**C# Example:**

```
m_spel.VDefSetMotionRange(100, 30, 1);
```

## VDefTool Method, Spel Class

**Description**

Using vision detection, calculates a tool offset value for TPC and mobile camera position.

Note:

When the tool type is other than FixedCameraWithCal, the robot operates automatically based on the detection results of the target. Be careful of interference between the robot and peripherals. Also, use with avoiding singularity nearby posture that each axis extends to prevent an error during the tool set.

**Syntax**

Sub **VDefTool**(ToolNumber As Integer, ToolDefType As SpelToolDefType, Sequence As String, Object As String)

Sub **VDefTool**(ToolNumber As Integer, ToolDefType As SpelToolDefType, Sequence As String, Object As String, Parent As Form)

Sub **VDefTool**(ToolNumber As Integer, ToolDefType As SpelToolDefType, Sequence As String, FinalAngle As Double, InitAngle As Double, TargetTolerance As Double)

Sub **VDefTool**(ToolNumber As Integer, ToolDefType As SpelToolDefType, Sequence As String, FinalAngle As Double, InitAngle As Double, TargetTolerance As Double, Parent As Form)

Sub **VDefTool**(ToolNumber As Integer, ToolDefType As SpelToolDefType, Sequence As String, FinalAngle As Double, InitAngle As Double, TargetTolerance As Double, RobotSpeed As Integer, RobotAccel As Integer)

Sub **VDefTool**(ToolNumber As Integer, ToolDefType As SpelToolDefType, Sequence As String, FinalAngle As Double, InitAngle As Double, TargetTolerance As Double, RobotSpeed As Integer, RobotAccel As Integer, Parent As Form)

**Parameters**

<i>ToolNumber</i>	Integer representing a tool number to perform tool set (1-15)
<i>ToolDefType</i>	Integer representing a tool type. FixedCamera: Tool set by using the fixed camera which is not calibrated. J4Camera: Calculates image center of the mobile J4 camera. J6Camera: Calculates image center of the mobile J6 camera. FixedCameraWithCal: Tool set by using the fixed camera which is calibrated.
<i>Sequence</i>	String expression representing the name of a vision sequence in the current project.
<i>Object</i>	String expression representing a vision object in the specified sequence. This parameter is required when <i>ToolDefType</i> is FixedCameraWithCal. When <i>ToolDefType</i> is not FixedCameraWithCal, Object should be an empty string.
<i>FinalAngle</i>	Real value representing an angle (degrees) to rotate the tool or camera tool. Value range: 0, 5 to 180, -5 to -180 If omitted, set to "90".
<i>InitAngle</i>	Real value representing an angle (degrees) to rotate the tool or camera tool in provisional tool setting. This value must be smaller than <i>FinalAngle</i> . Value range: -10 to 10 If omitted, set to "5".
<i>TargetTolerance</i>	Real value representing a pixel distance to consider that the vision detection result matches the target position. Value range: 0 to 3 pixels If omitted, set to "1".
<i>Parent</i>	Optional. Parent .NET form of a window.



<i>RobotSpeed</i>	Optional. Integer variable that will contain the robot speed (%). Value range: 0 to 100 If omitted, set to "5".
<i>RobotAccel</i>	Optional. Integer variable that will contain the robot acceleration (%). Value range: 0 to 99 If omitted, set to "5".

**See Also**

VDefArm, VDefGetMotionRange, VDefLocal, VDefSetMotionRange, VGoCenter

**VDefTool Example****VB Example:**

```
m_spel.VDefTool(1, SpelToolDefType.J6Camera, "myseq", 45, 5,
3.0)
m_spel.VDefTool(1, SpelToolDefType.FixedCameraWithCal,
"myseq", "myobj")
```

**C# Example:**

```
m_spel.VDefTool(1, SpelToolDefType.J6Camera, "myseq", 45, 5,
3.0);
m_spel.VDefTool(1, SpelToolDefType.FixedCameraWithCal,
"myseq", "myobj");
```

## VDefToolXYZ Method, Spel Class

**Description**

VDefToolXYZ uses vision detection to calculate tool XYZ offset values and define a tool.

Note:

The robot operates automatically based on the detection results of the target. Be careful of interference between the robot and peripherals. Also, be careful to avoid singularities for nearby postures to prevent an error during the tool

**Syntax**

Sub **VDefToolXYZ**(ToolNumber As Integer, LocalNumber As Integer, PointNumber1 As Integer, PointNumber2 As Integer, Sequence1 As String, Sequence2 As String, FinalAngle As Double, InitAngle As Double, TargetTolerance As Double, RobotSpeed As Integer, RobotAccel As Integer)

Sub **VDefToolXYZ**(ToolNumber As Integer, LocalNumber As Integer, PointNumber1 As Integer, PointNumber2 As Integer, Sequence1 As String, Sequence2 As String, FinalAngle As Double, InitAngle As Double, TargetTolerance As Double, RobotSpeed As Integer, RobotAccel As Integer, Parent As Form)

**Parameters**

<i>ToolNumber</i>	Integer representing a tool number to perform tool set (1 to 15)
<i>LocalNumber</i>	Integer variable representing a local coordinate number for moving the robot. Tool is moved in the XY plane of the specified local coordinate.
<i>PointNumber1</i>	Integer variable representing a point number of first pose
<i>PointNumber2</i>	Integer variable representing a point number of second pose
<i>Sequence1</i>	String expression containing the name of a vision sequence for first pose in the current project.
<i>Sequence2</i>	String expression containing the name of a vision sequence for second pose in the current project.
<i>FinalAngle</i>	Real value representing an angle (degrees). Value range: 5 to 180, -5 to -180
<i>InitAngle</i>	Real value representing an angle (degrees) that indicates the rotation angle when setting a temporary tool. This value must be smaller than <i>FinalAngle</i> . Value range: 0.01 to 10, -0.01 to -10
<i>TargetTolerance</i>	Real value representing a pixel distance to consider that the vision detection result matches the target position. Value range: 0.1 to 3.0 pixels
<i>RobotSpeed</i>	Integer variable that will contain the robot speed (%). Value range: 1 to 100
<i>RobotAccel</i>	Integer variable that will contain the robot acceleration (%). Value range: 1 to 99
<i>Parent</i>	Optional. Parent .NET form of a window.

**See Also**

VDefTool, VDefToolXYZUVW

**VDefToolXYZ Example****VB Example:**

```
m_spel.VDefToolXYZ(1, 0, 1, 2, "seq01", "seq02", 5, 30, 1,
5, 5)
m_spel.VDefToolXYZ(2, 0, 3, 4, "seq03", "seq04", 5, 30, 1,
5, 5)
m_spel.VDefToolXYZ(3, 0, 5, 6, "seq05", "seq06", 5, 30, 1,
5, 5)
m_spel.VDefToolXYZUVW(1, 2, 3, SpelToolDefType3D.Bar)
```

**C# Example:**

```
m_spel.VDefToolXYZ(1, 0, 1, 2, "seq01", "seq02", 5, 30, 1,
5, 5);
m_spel.VDefToolXYZ(2, 0, 3, 4, "seq03", "seq04", 5, 30, 1,
5, 5);
m_spel.VDefToolXYZ(3, 0, 5, 6, "seq05", "seq06", 5, 30, 1,
5, 5);
m_spel.VDefToolXYZUVW(1, 2, 3, SpelToolDefType3D.Bar);
```

## VDefToolXYZUVW Method, Spel Class

**Description**

VDefToolXYZUVW calculates tool UVW offset values using three tool definitions.

Note:

The calculated U, V, W tool offset values are set in the specified ToolNumber1. The X, Y, Z offsets for ToolNumber1 are not changed.

**Syntax**

Sub **VDefToolXYZUVW**(ToolNumber1 As Integer, ToolNumber2 As Integer, ToolNumber3 As Integer, ToolDefType3D As SpelToolDefType3D )

**Parameters**

<i>ToolNumber1</i>	Integer representing a tool number for tool definition 1 (1 to 15). When ToolDefType3D is Bar, specify the tool that defines the tip of the bar. When ToolDefType3D is Plane, specify the tool that defines the center of the plane.
<i>ToolNumber2</i>	Integer representing a tool for tool definition 2 (1 to 15). When ToolType is bar, specify the tool that defines the middle of the bar. When ToolType is plane, specify the tool that defines a position on the plane other than the center of the plane and other than the position defined by tool definition 3.
<i>ToolNumber3</i>	Integer representing a tool number for tool definition 3 (1 to 15). When ToolType is bar, specify the tool that defines the root of the bar. When ToolType is plane, specify the tool that defines a position other than the center of the plane and other than the position defined by tool definition 2.
<i>ToolDefType3D</i>	Integer representing a tool type. Bar: Type of bar Plane: Type of plane.

**See Also**

VDefToolXYZ

**VDefToolXYZUVW Example****VB Example:**

```
m_spel.VDefToolXYZ(1, 0, 1, 2, "seq01", "seq02", 5, 30, 1,
5, 5)
m_spel.VDefToolXYZ(2, 0, 3, 4, "seq03", "seq04", 5, 30, 1,
5, 5)
m_spel.VDefToolXYZ(3, 0, 5, 6, "seq05", "seq06", 5, 30, 1,
5, 5)
m_spel.VDefToolXYZUVW(1, 2, 3, SpelToolDefType3D.Bar)
```

**C# Example:**

```
m_spel.VDefToolXYZ(1, 0, 1, 2, "seq01", "seq02", 5, 30, 1,
5, 5);
m_spel.VDefToolXYZ(2, 0, 3, 4, "seq03", "seq04", 5, 30, 1,
5, 5);
m_spel.VDefToolXYZ(3, 0, 5, 6, "seq05", "seq06", 5, 30, 1,
5, 5);
m_spel.VDefToolXYZUVW(1, 2, 3, SpelToolDefType3D.Bar);
```

VDeleteCalibration Method, Spel Class

**Description**

Deletes a vision calibration in the current project.

**Syntax**

Sub **VDeleteCalibration** (*CalibName* As String)

**Parameters**

*CalibName* String expression containing the name of a vision calibration in the current project.

**See Also**

VCreateCalibration, VDeleteObject, VDeleteSequence

**VDeleteCalibration Example**

**VB Example:**

```
m_spel.VDeleteCalibration("mycal")
```

**C# Example:**

```
m_spel.VDeleteCalibration("mycal");
```

## VDeleteObject Method, Spel Class

**Description**

Deletes a vision object in the current project.

**Syntax**

Sub **VDeleteObject** (*Sequence* As String, *ObjectName* As String)

**Parameters**

*Sequence* String expression containing the name of a vision sequence in the current project.

*ObjectName* String expression containing the name of a vision object in the current project.

**See Also**

VCreateObject, VCreateSequence, VDeleteSequence

**VDeleteObject Example****VB Example:**

```
m_spel.VDeleteObject("myseq", "myobj")
```

**C# Example:**

```
m_spel.VDeleteObject("myseq", "myobj");
```

VDeleteSequence Method, Spel Class

**Description**

Deletes a vision sequence in the current project.

**Syntax**

Sub **VDeleteSequence** (*Sequence* As String)

**Parameters**

*Sequence* String expression containing the name of a vision sequence in the current project.

**See Also**

VCreateObject, VCreateSequence, VDeleteObject

**VDeleteSequence Example**

**VB Example:**

```
m_spel.VDeleteSequence("myseq")
```

**C# Example:**

```
m_spel.VDeleteSequence("myseq");
```



## VEditWindow Method, Spel Class

**Description**

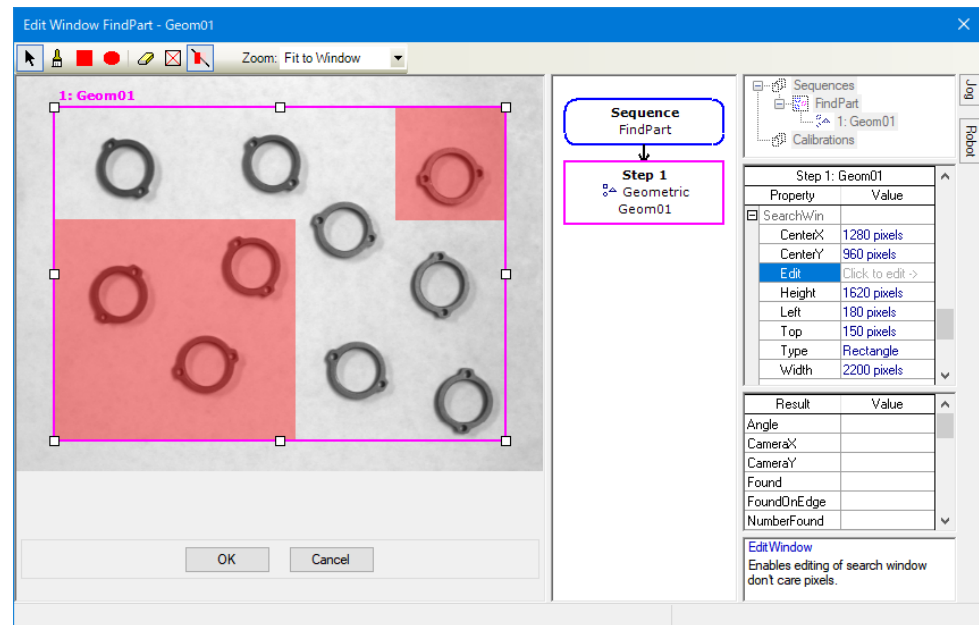
Displays the editor for search window don't care pixels. For more details, see the EditWindow Property in the Vision Guide Properties reference.

**Syntax**

Sub **VEditWindow** (*Sequence As String, Object As String*)

**Parameters**

- Sequence* String expression containing the name of a vision sequence in the current project.
- Object* String expression containing the name of a vision object in the current project.

**See Also**

VSave

**VEditWindow Example****VB Example:**

```
m_spel.VEditWindow("myseq", "myobj")
```

**C# Example:**

```
m_spel.VEditWindow("myseq", "myobj");
```

## VGet Method, Spel Class

**Description**

Gets the value of a vision sequence or object property or result.

**Syntax**

Sub **VGet** (*Sequence* As String, *PropCode* As SpelVisionProps, ByRef *Value* As Integer)  
 Sub **VGet** (*Sequence* As String, *PropCode* As SpelVisionProps, ByRef *Value* As Boolean)  
 Sub **VGet** (*Sequence* As String, *PropCode* As SpelVisionProps, ByRef *Value* As Double)  
 Sub **VGet** (*Sequence* As String, *PropCode* As SpelVisionProps, ByRef *Value* As Single)  
 Sub **VGet** (*Sequence* As String, *PropCode* As SpelVisionProps, ByRef *Value* As String)  
 Sub **VGet** (*Sequence* As String, *Object* As String, *PropCode* As SpelVisionProps,  
     ByRef *Value* As Integer )  
 Sub **VGet** (*Sequence* As String, *Object* As String, *PropCode* As SpelVisionProps,  
     ByRef *Value* As Boolean)  
 Sub **VGet** (*Sequence* As String, *Object* As String, *PropCode* As SpelVisionProps,  
     ByRef *Value* As Color)  
 Sub **VGet** (*Sequence* As String, *Object* As String, *PropCode* As SpelVisionProps,  
     ByRef *Value* As Double)  
 Sub **VGet** (*Sequence* As String, *Object* As String, *PropCode* As SpelVisionProps,  
     ByRef *Value* As Single)  
 Sub **VGet** (*Sequence* As String, *Object* As String, *PropCode* As SpelVisionProps,  
     ByRef *Value* As String)  
 Sub **VGet** (*Sequence* As String, *Object* As String, *PropCode* As SpelVisionProps,  
     *Result* As Integer, ByRef *Value* As Integer)  
 Sub **VGet** (*Sequence* As String, *Object* As String, *PropCode* As SpelVisionProps,  
     *Result* As Integer, ByRef *Value* As Boolean)  
 Sub **VGet** (*Sequence* As String, *Object* As String, *PropCode* As SpelVisionProps,  
     *Result* As Integer, ByRef *Value* As Double)  
 Sub **VGet** (*Sequence* As String, *Object* As String, *PropCode* As SpelVisionProps,  
     *Result* As Integer, ByRef *Value* As Single)  
 Sub **VGet** (*Sequence* As String, *Object* As String, *PropCode* As SpelVisionProps,  
     *Result* As Integer, ByRef *Value* As String)

**Parameters**

<i>Sequence</i>	String expression containing the name of a vision sequence in the current project.
<i>Object</i>	String expression containing the name of an object in sequence <i>Sequence</i> . If the property is for a sequence, then this string must be empty.
<i>PropCode</i>	A SpelVisionProps value that specifies the property code.
<i>Result</i>	The integer expression representing the result number.
<i>Value</i>	Variable containing property or result value. The type of the variable must match the property or result type.

**See Also**

VSet, VRun

## VGet Example

### VB Example:

```
Dim i As Integer
Redim score(10) As Integer

m_spel.VRun("testSeq")
For i = 1 to 10
    m_spel.VGet("testSeq", "corr" & Format$(i, "00"), _
        SpelVisionProps.Score, score(i))
Next i
```

### C# Example:

```
int[] score = new int[11];
for(int i = 1; i <= 10; i++)
{
    m_spel.VGet("testSeq", string.Format("Corr 0{0}, i),
        SpelVisionProps.Score, score(i));
}
```

## VGetCameraXYU Method, Spel Class

**Description**

Retrieves camera X, Y, and U physical coordinates for any object.

**Syntax**

Sub **VGetCameraXYU** (*Sequence* As String, *Object* As String, *Result* As Integer, ByRef *Found* As Boolean, ByRef *X* As Single, ByRef *Y* As Single, ByRef *U* As Single)

**Parameters**

<i>Sequence</i>	String expression containing the name of a vision sequence in the current project.
<i>Object</i>	String expression containing the name of an object in sequence <i>Sequence</i> .
<i>Result</i>	Integer expression representing the result number.
<i>Found</i>	Boolean variable that will contain whether or not the object was found.
<i>X</i>	Real variable that will contain x coordinate in millimeters.
<i>Y</i>	Real variable that will contain y coordinate in millimeters.
<i>U</i>	Real variable that will contain angle in degrees.

**See Also**

VGetPixelXYU, VGetRobotXYU

**VGetCameraXYU Example****VB Example:**

```
Dim found As Boolean
Dim x As Single, y As Single, u As Single
Dim seq As String, blob As String

seq = "testSeq"
blob = "blob01"
m_spel.VRun(seq)
m_spel.VGetCameraXYU(seq, blob, 1, found, x, y, u)
```

**C# Example:**

```
bool found;
float x, y, u;
string seq, blob;

seq = "testSeq";
blob = "blob01";
m_spel.VRun(seq);
m_spel.VGetCameraXYU(seq, blob, 1, out found, out x, out y, out u);
```

## VGetEdgeCameraXYU Method, Spel Class

**Description**

Retrieves camera X, Y, and U physical coordinates for each edge of a Line Finder, Arc Finder search.

**Syntax**

Sub **VGetEdgeCameraXYU** (*Sequence* As String, *Object* As String, *EdgeResultIndex* As Integer, *ByRef Found* As Boolean, *ByRef X* As Single, *ByRef Y* As Single, *ByRef U* As Single)

**Parameters**

<i>Sequence</i>	String expression containing the name of a vision sequence in the current project.
<i>Object</i>	String expression containing the name of an object in sequence <i>Sequence</i> .
<i>EdgeResultIndex</i>	Integer expression representing the edge result index.
<i>Found</i>	Boolean variable that will contain whether or not the object was found.
<i>X</i>	Real variable that will contain x coordinate in millimeters.
<i>Y</i>	Real variable that will contain y coordinate in millimeters.
<i>U</i>	Real variable that will contain angle in degrees.

**See Also**

VGetEdgePixelXYU, VGetEdgeRobotXYU, VGetPixelXYU, VGetRobotXYU

**VGetEdgeCameraXYU Example****VB Example:**

```
Dim found(10) As Boolean
Dim x(10) As Single, y(10) As Single, u(10) As Single
Dim seq As String, lineFinder As String

seq = "testSeq"
lineFinder = "LineFind01"
m_spel.VRun(seq)
' The NumberOfEdges for the LineFinder is 10
For i = 1 To 10
    m_spel.VGetEdgeCameraXYU(seq, lineFinder, i, found(i),
    x(i),
        y(i), u(i))
Next i
```

### C# Example:

```
bool[] found = new bool[11];
float[] x = new float[11];
float[] y = new float[11];
float[] u = new float[11];
string seq, lineFinder;
seq = "testSeq";
lineFinder = "LineFind01";
m_spel.VRun(seq);
// The NumberOfEdges for the LineFinder is 10
for(int i = 1; i <= 10; i++)
    m_spel.VGetEdgeCameraXYU(seq, lineFinder, i, out found[i],
        out x[i], out y[i], out u[i]);
```

## VGetEdgePixelXYU Method, Spel Class

**Description**

Retrieves X, Y, and U pixel coordinates for each edge of a Line Finder, Arc Finder search.

**Syntax**

Sub **VGetEdgePixelXYU** (*Sequence* As String, *Object* As String, *EdgeResultIndex* As Integer, *ByRef Found* As Boolean, *ByRef X* As Single, *ByRef Y* As Single, *ByRef U* As Single)

**Parameters**

<i>Sequence</i>	String expression containing the name of a vision sequence in the current project.
<i>Object</i>	String expression containing the name of an object in sequence <i>Sequence</i> .
<i>EdgeResultIndex</i>	Integer expression representing the edge result index.
<i>Found</i>	Boolean variable that will contain whether or not the object was found.
<i>X</i>	Real variable that will contain x coordinate in millimeters.
<i>Y</i>	Real variable that will contain y coordinate in millimeters.
<i>U</i>	Real variable that will contain angle in degrees.

**See Also**

VGetEdgeCameraXYU, VGetEdgeRobotXYU, VGetPixelXYU, VGetRobotXYU

**VGetEdgePixelXYU Example****VB Example:**

```
Dim found(10) As Boolean
Dim x(10) As Single, y(10) As Single, u(10) As Single
Dim seq As String, lineFinder As String

seq = "testSeq"
lineFinder = "LineFind01"
m_spel.VRun(seq)
' The NumberOfEdges for the LineFinder is 10
For i = 1 To 10
    m_spel.VGetEdgePixelXYU(seq, lineFinder, i, found(i),
    x(i),
        y(i), u(i))
Next i
```

### C# Example:

```
bool[] found = new bool[11];
float[] x = new float[11];
float[] y = new float[11];
float[] u = new float[11];
string seq, lineFinder;
seq = "testSeq";
lineFinder = "LineFind01";
m_spel.VRun(seq);
// The NumberOfEdges for the LineFinder is 10
for(int i = 1; i <= 10; i++)
    m_spel.VGetEdgePixelXYU(seq, lineFinder, i, out found[i],
        out x[i], out y[i], out u[i]);
```



## VGetEdgeRobotXYU Method, Spel Class

**Description**

Retrieves robot X, Y, and U physical coordinates for each edge of a Line Finder, Arc Finder search.

**Syntax**

Sub **VGetEdgeRobotXYU** (*Sequence* As String, *Object* As String, *EdgeResultIndex* As Integer, *ByRef Found* As Boolean, *ByRef X* As Single, *ByRef Y* As Single, *ByRef U* As Single)

**Parameters**

<i>Sequence</i>	String expression containing the name of a vision sequence in the current project.
<i>Object</i>	String expression containing the name of an object in sequence <i>Sequence</i> .
<i>EdgeResultIndex</i>	Integer expression representing the edge result index.
<i>Found</i>	Boolean variable that will contain whether or not the object was found.
<i>X</i>	Real variable that will contain x coordinate in millimeters.
<i>Y</i>	Real variable that will contain y coordinate in millimeters.
<i>U</i>	Real variable that will contain angle in degrees.

**See Also**

VGetEdgeCameraXYU, VGetEdgePixelXYU, VGetPixelXYU, VGetRobotXYU

**VGetEdgeRobotXYU Example****VB Example:**

```
Dim found(10) As Boolean
Dim x(10) As Single, y(10) As Single, u(10) As Single
Dim seq As String, lineFinder As String

seq = "testSeq"
lineFinder = "LineFind01"
m_spel.VRun(seq)
' The NumberOfEdges for the LineFinder is 10
For i = 1 To 10
    m_spel.VGetEdgeRobotXYU(seq, lineFinder, i, found(i),
    x(i),
        y(i), u(i))
Next i
```

### C# Example:

```
bool[] found = new bool[11];
float[] x = new float[11];
float[] y = new float[11];
float[] u = new float[11];
string seq, lineFinder;
seq = "testSeq";
lineFinder = "LineFind01";
m_spel.VRun(seq);
// The NumberOfEdges for the LineFinder is 10
for(int i = 1; i <= 10; i++)
    m_spel.VGetEdgeRobotXYU(seq, lineFinder, i, out found[i],
        out x[i], out y[i], out u[i]);
```

## VGetExtrema Method, Spel Class

**Description**

Retrieves extrema coordinates of a blob object.

**Syntax**

Sub **VGetExtrema** (*Sequence* As String, *Object* As String, *Result* As Integer, ByRef *MinX* As Single, ByRef *MaxX* As Single, ByRef *MinY* As Single, ByRef *MaxY* As Single)

**Parameters**

<i>Sequence</i>	String expression containing the name of a vision sequence in the current project.
<i>Object</i>	String expression containing the name of an object in sequence <i>Sequence</i> .
<i>Result</i>	Integer expression representing the result number.
<i>MinX</i>	Real variable that will contain minimum x coordinate in pixels.
<i>MaxX</i>	Real variable that will contain maximum x coordinate in pixels.
<i>MinY</i>	Real variable that will contain minimum y coordinate in pixels.
<i>MaxY</i>	Real variable that will contain maximum y coordinate in pixels.

**See Also**

VGet

**VGetExtrema Example****VB Example:**

```
Dim xmin As Single, xmax As Single
Dim ymin As Single, ymax As Single
Dim seq As String, blob As String

seq = "testSeq"
blob = "blob01"
m_spel.VRun(seq)
m_spel.VGet(seq, blob, "found", found)
If found <> 0 Then
    m_spel.VGetExtrema(seq, blob, xmin, xmax, ymin, ymax)
End If
```

**C# Example:**

```
float xmin, xmax, ymin, ymax;
bool found;
string seq, blob;

seq = "testSeq";
blob = "blob01";
m_spel.VRun(seq);
m_spel.VGet(seq, blob, "found", found);

if(found == true)
    m_spel.VGetExtrema(seq, blob, out xmin, out xmax, out ymin,
        out ymax);
```

## VGetModelWin Method, Spel Class

**Description**

Retrieves model window coordinates for objects.

**Syntax**

Sub **VGetModelWin** (*Sequence* As String, *Object* As String, ByRef *Left* As Integer, ByRef *Top* As Integer, ByRef *Width* As Integer, ByRef *Height* As Integer)

**Parameters**

<i>Sequence</i>	String expression containing the name of a vision sequence in the current project.
<i>Object</i>	String expression containing the name of an object in sequence <i>Sequence</i> .
<i>Left</i>	Integer variable that will contain left coordinate in pixels.
<i>Top</i>	Integer variable that will contain top coordinate in pixels.
<i>Width</i>	Integer variable that will contain width in pixels.
<i>Height</i>	Integer variable that will contain height in pixels.

**See Also**

VSetModelWin, VGetSearchWin, VSetSearchWin

**VGetModelWin Example****VB Example:**

```
Dim left As Integer, top As Integer
Dim width As Integer, height As Integer
```

```
With m_spel
    .VGetModelWin("testSeq", "corr01", left, top, _
        width, height)
    .VSetModelWin("testSeq", "corr01", left + 20, top, _
        width, height)
    .VTeach("testSeq", "corr01")
End With
```

**C# Example:**

```
int left, top, width, height;

m_spel.VGetModelWin("testSeq", "corr01", out left, out top,
    out width, out height);
m_spel.VSetModelWin("testSeq", "corr01", left + 20, top,
    width, height);
m_spel.VTeach("testSeq", "corr01");
```

## VGetPixelToCamera Method, Spel Class

**Description**

Retrieves the camera coordinates for the specified pixel coordinates.

**Syntax**

Sub **VGetPixelToCamera** (*Calibration* As String, *PixelX* As Single, *PixelY* As Single, *Angle* As Single, *ByRef CameraX* As Single, *ByRef CameraY* As Single, *ByRef CameraU* As Single)

**Parameters**

<i>Calibration</i>	String expression containing the name of a vision calibration in the current project.
<i>PixelX</i>	Real variable that will contain x coordinate in pixels.
<i>PixelY</i>	Real variable that will contain y coordinate in pixels.
<i>Angle</i>	Real variable that will contain the angle in degrees.
<i>CameraX</i>	Real variable that will contain x coordinate in millimeters.
<i>CameraY</i>	Real variable that will contain y coordinate in millimeters.
<i>CameraU</i>	Real variable that will contain angle in degrees.

**See Also**

VGetPixelXYU, VGetCameraXYU, VGetPixelToRobot

**VGetPixelToCamera Example****VB Example:**

```
Dim x As Single, y As Single, u As Single
Dim cal As String, seq As String

cal = "testCal"
seq = "testSeq"
m_spel.VRun(seq)
m_spel.VGetPixelToCamera(cal, 400, 300, 0, x, y, u)
```

**C# Example:**

```
float x, y, u
string cal, seq;

cal = "testCal";
seq = "testSeq";
m_spel.VRun(seq);
m_spel.VGetPixelToCamera(cal, 400, 300, 0, out x, out y, out u);
```

## VGetPixelToRobot Method, Spel Class

**Description**

Retrieves robot world coordinates for the specified pixel coordinates.

**Syntax**

Sub **VGetPixelToRobot** (*Calibration* As String, *PixelX* As Single, *PixelY* As Single, *Angle* As Single, ByRef *RobotX* As Single, ByRef *RobotY* As Single, ByRef *RobotU* As Single)

**Parameters**

<i>Calibration</i>	String expression containing the name of a vision calibration in the current project.
<i>PixelX</i>	Real variable that will contain x coordinate in pixels.
<i>PixelY</i>	Real variable that will contain y coordinate in pixels.
<i>Angle</i>	Real variable that will contain the angle in degrees.
<i>RobotX</i>	Real variable that will contain x coordinate in millimeters.
<i>RobotY</i>	Real variable that will contain y coordinate in millimeters.
<i>RobotU</i>	Real variable that will contain the angle in degrees.

**See Also**

VGetPixelXYU, VGetRobotXYU, VGetPixelToCamera

**VGetPixelToRobot Example****VB Example:**

```
Dim x As Single, y As Single, u As Single
Dim cal As String, seq As String
```

```
cal = "testCal"
seq = "testSeq"
m_spel.VRun(seq)
m_spel.VGetPixelToRobot(cal, 400, 300, 0, x, y, u)
```

**C# Example:**

```
float x, y, u
string cal, seq;
```

```
cal = "testCal";
seq = "testSeq";
m_spel.VRun(seq);
m_spel.VGetPixelToRobot(cal, 400, 300, 0, out x, out y, out u);
```

## VGetPixelXYU Method, Spel Class

**Description**

Retrieves pixel X, Y, and U coordinates for any object.

**Syntax**

Sub **VGetPixelXYU** (*Sequence* As String, *Object* As String, *Result* As Integer, ByRef *Found* As Boolean, ByRef *X* As Single, ByRef *Y* As Single, ByRef *U* As Single)

**Parameters**

<i>Sequence</i>	String expression containing the name of a vision sequence in the current project.
<i>Object</i>	String expression containing the name of an object in sequence <i>Sequence</i> .
<i>Result</i>	Integer expression representing the result number.
<i>Found</i>	Boolean variable that will contain whether or not the object was found.
<i>X</i>	Real variable that will contain x coordinate in pixels.
<i>Y</i>	Real variable that will contain y coordinate in pixels.
<i>U</i>	Real variable that will contain the angle in degrees.

**See Also**

VGetCameraXYU, VGetRobotXYU

**VGetPixelXYU Example****VB Example:**

```
Dim found As Integer
Dim x As Single, y As Single, u As Single
Dim seq As String, blob As String

seq = "testSeq"
blob = "blob01"
m_spel.VRun(seq)
m_spel.VGetPixelXYU(seq, blob, 1, found, x, y, u)
```

**C# Example:**

```
int found;
float x, y, u;
string seq, blob;

seq = "testSeq";
blob = "blob01";
m_spel.VRun(seq);
m_spel.VGetPixelXYU(seq, blob, 1, out found, out x, out y,
out u);
```

## VGetRobotPlacePos Method, Spel Class

**Description**

Retrieves robot place position.

**Syntax**

Sub **VGetRobotPlacePos** (*Sequence* As String, *Object* As String, *Result* As Integer, ByRef *Found* As Boolean, ByRef *PlacePoint* As SpelPoint)

**Parameters**

<i>Sequence</i>	String expression containing the name of a vision sequence in the current project.
<i>Object</i>	String expression containing the name of an object in sequence <i>Sequence</i> .
<i>Result</i>	Integer expression representing the result number.
<i>Found</i>	Integer variable that will contain boolean found status. If found is false, then <i>x</i> , <i>y</i> , and <i>u</i> are undefined.
<i>PlacePoint</i>	SpelPoint variable that will contain the place position

**See Also**

VGetRobotPlaceTargetPos, VSetRobotPlaceTargetPos

**VGetRobotPlacePos Example****VB Example:**

```
Dim found As Integer
Dim x As Single, y As Single, u As Single
Dim seq As String, blob As String
Dim placePoint As SpelPoint

seq = "testSeq"
blob = "blob01"
' Move part above upward camera
m_spel.Jump("camPos")
m_spel.VRun(seq)
m_spel.VGetRobotPlacePos(seq, blob, 1, found, placePoint)
' Using a SCARA, to use +TLZ for approach
m_spel.Jump(placePoint, "+TLZ(10)")
m_spel.Go(placePoint)
```



**C# Example:**

```
bool found;
float x, y, u;
string seq, blob;
SpelPoint placePoint = new SpelPoint();
seq = "testSeq";
blob = "blob01";

// Move part above upward camera
m_spel.Jump("camPos");
m_spel.VRun(seq);
m_spel.VGetRobotPlacePos(seq, blob, 1, out found, out
placePoint);
// Using a SCARA, to use +TLZ for approach
m_spel.Jump(placePoint, "+TLZ(10)");
m_spel.Go(placePoint);
```

## VGetRobotPlaceTargetPos Method, Spel Class

**Description**

Retrieves part place position.

**Syntax**

Sub **VGetRobotPlaceTargetPos** (*Sequence* As String, *Object* As String, ByRef *Point* As SpelPoint)

**Parameters**

*Sequence* String expression containing the name of a vision sequence in the current project.

*Object* String expression containing the name of an object in sequence *Sequence*.

*Point* SpelPoint variable that will contain the place position.

**See Also**

VGetRobotPlacePos, VSetRobotPlaceTargetPos

**VGetRobotPlaceTargetPos Example****VB Example:**

```
Dim seq As String, blob As String
Dim targetPoint As SpelPoint

seq = "testSeq"
blob = "blob01"
m_spel.VGetRobotPlaceTargetPos(seq, blob, targetPoint)

' Adjust the place position
targetPoint.X = targetPoint.X + 10
m_spel.VSetRobotPlaceTargetPos(seq, blob, targetPoint)
```

**C# Example:**

```
string seq, blob;
SpelPoint targetPoint = new SpelPoint();

seq = "testSeq";
blob = "blob01";
m_spel.VGetRobotPlaceTargetPos(seq, blob, out targetPoint);

// Adjust the place position
targetPoint.X = targetPoint.X + 10;
m_spel.VSetRobotPlaceTargetPos(seq, blob, targetPoint);
```

## VGetRobotXYU Method, Spel Class

**Description**

Retrieves robot world X, Y, and U coordinates for any object.

**Syntax**

Sub **VGetRobotXYU** (*Sequence* As String, *Object* As String, *Result* As Integer, ByRef *Found* As Boolean, ByRef *X* As Single, ByRef *Y* As Single, ByRef *U* As Single)

**Parameters**

<i>Sequence</i>	String expression containing the name of a vision sequence in the current project.
<i>Object</i>	String expression containing the name of an object in sequence <i>Sequence</i> .
<i>Result</i>	Integer expression representing the result number.
<i>Found</i>	Integer variable that will contain boolean found status. If found is false, then <i>x</i> , <i>y</i> , and <i>u</i> are undefined.
<i>X</i>	Real variable that will contain x coordinate in millimeters.
<i>Y</i>	Real variable that will contain y coordinate in millimeters.
<i>U</i>	Real variable that will contain the angle in degrees.

**See Also**

VGetCameraXYU, VGetPixelXYU

**VGetRobotXYU Example****VB Example:**

```
Dim found As Integer
Dim x As Single, y As Single, u As Single
Dim seq As String, blob As String

seq = "testSeq"
blob = "blob01"
m_spel.VRun(seq)
m_spel.VGetRobotXYU(seq, blob, 1, found, x, y, u)
```

**C# Example:**

```
bool found;
float x, y, u;
string seq, blob;

seq = "testSeq";
blob = "blob01";
m_spel.VRun(seq);
m_spel.VGetRobotXYU(seq, blob, 1, out found, out x, out y,
out u);
```

## VGetRobotToolXYU Method, Spel Class

**Description**

Retrieves robot world X, Y, and U values for tool definition.

**Syntax**

Sub **VGetRobotToolXYU** (*Sequence* As String, *Object* As String, *Result* As Integer, ByRef *Found* As Boolean, ByRef *X* As Single, ByRef *Y* As Single, ByRef *U* As Single)

**Parameters**

<i>Sequence</i>	String expression containing the name of a vision sequence in the current project.
<i>Object</i>	String expression containing the name of an object in sequence <i>Sequence</i> .
<i>Result</i>	Integer expression representing the result number.
<i>Found</i>	Integer variable that will contain boolean found status. If found is false, then <i>x</i> , <i>y</i> , and <i>u</i> are undefined.
<i>X</i>	Real variable that will contain x coordinate in millimeters.
<i>Y</i>	Real variable that will contain y coordinate in millimeters.
<i>U</i>	Real variable that will contain the angle in degrees.

**Remarks**

Use VGetRobotToolXYU to easily define a tool for a part viewed by an upward camera. This allows you to pick up a part, search for it in the upward camera FOV, define a tool for the part, then place the part.

**See Also**

VGetCameraXYU, VGetPixelXYU, VGetRobotXYU

**VGetRobotToolXYU Example****VB Example:**

```
Dim found As Integer
Dim x As Single, y As Single, u As Single
Dim seq As String, blob As String

seq = "testSeq"
blob = "blob01"
' Move part above upward camera
m_spel.Jump("camPos")
m_spel.VRun(seq)
m_spel.VGetRobotToolXYU(seq, blob, 1, found, x, y, u)
m_spel.TLSet(1, x, y, u)
```

**C# Example:**

```
bool fnd;
float x, y, u;
string seq, blob;

seq = "testSeq";
blob = "blob01";
// Move part above upward camera
m_spel.Jump("camPos");
m_spel.VRun(seq);
m_spel.VGetRobotToolXYU(seq, blob, 1, out fnd, out x, out y,
out u);
m_spel.TLSet(1, x, y, u);
```

## VGetSearchWin Method, Spel Class

**Description**

Retrieves search window coordinates.

**Syntax**

Sub **VGetSearchWin** (*Sequence* As String, *Object* As String, ByRef *Left* As Integer, ByRef *Top* As Integer, ByRef *Width* As Integer, ByRef *Height* As Integer)

**Parameters**

<i>Sequence</i>	String expression containing the name of a vision sequence in the current project.
<i>Object</i>	String expression containing the name of an object in sequence <i>Sequence</i> .
<i>Left</i>	Integer variable that will contain left coordinate in pixels.
<i>Top</i>	Integer variable that will contain top coordinate in pixels.
<i>Width</i>	Integer variable that will contain width in pixels.
<i>Height</i>	Integer variable that will contain height in pixels.

**See Also**

VGetModelWin, VSetModelWin, VSetSearchWin

**VGetSearchWin Example****VB Example:**

```
Dim left As Integer, top As Integer
Dim width As Integer, height As Integer

With m_spel
    .VGetSearchWin("testSeq", "corr01", left, top, _
        width, height)
    .VSetSearchWin("testSeq", "corr01", newLeft, top, _
        width, height)
    .VRun("testSeq")
End With
```

**C# Example:**

```
int left, top, width, height;

m_spel.VGetSearchWin("testSeq", "corr01", out left, out top,
    out width, out height);
m_spel.VSetSearchWin("testSeq", "corr01", newLeft, top,
    width, height);
m_spel.VRun("testSeq");
```

## VGoCenter Method, Spel Class

**Description**

Using a feature point that can be detected by the vision system, moves the robot to a position where the feature point is on the center of the camera image.

**Syntax**

Sub **VGoCenter**(Sequence As String, LocalNumber As Integer, TargetTolerance As Double)

Sub **VGoCenter**(Sequence As String, LocalNumber As Integer, TargetTolerance As Double, Parent As Form)

Sub **VGoCenter**(*Sequence* As String, *LocalNumber* As Integer, *TargetTolerance* As Double, *RobotSpeed* As Integer, *RobotAccel* As Integer)

Sub **VGoCenter**(*Sequence* As String, *LocalNumber* As Integer, *TargetTolerance* As Double, *RobotSpeed* As Integer, *RobotAccel* As Integer, *Parent* As Form)

**Parameters**

<i>Sequence</i>	String expression representing a vision sequence name of current project.
<i>LocalNumber</i>	Integer representing the local coordinate number where the robot is moved. If -1 is specified, the robot moves in the XY plane of the tool rotation
<i>TargetTolerance</i>	Real value representing a pixel distance to consider that the vision detection result matches the target position. Value range: 0 to 3 pixels
<i>Form</i>	Parent .NET form of a window (optional)
<i>RobotSpeed</i>	Optional. Integer variable that will contain the robot speed (%). Value range: 0 to 100 If omitted, set to "5".
<i>RobotAccel</i>	Optional. Integer variable that will contain the robot acceleration (%). Value range: 0 to 99 If omitted, set to "5".

**See Also**

VDefArm, VDefGetMotionRange, VDefLocal, VDefSetMotionRange, VDefTool

**VGoCenter Example****VB Example:**

```
m_spel.VGoCenter("myseq", 1, 1.0)
```

**C# Example:**

```
m_spel.VGoCenter("myseq", 1, 1.0);
```

VLoad Method, Spel Class

**Description**

Loads vision properties from the current project.

**Syntax**

Sub **VLoad** ()

**Remarks**

Use the VLoad method when you want to return the vision property settings, models, and fonts back to their original settings when the program was started.

**See Also**

VSave

**VLoad Example**

**VB Example:**

```
m_spel.VLoad()
```

**C# Example:**

```
m_spel.VLoad();
```



## VLoadModel Method, Spel Class

**Description**

Load a vision model from a disk file.

**Syntax**

Sub **VLoadModel** (*Sequence* As String, *Object* As String, *Path* As String)

**Parameters**

<i>Sequence</i>	String containing the name of a sequence in the current project.
<i>Object</i>	String containing the name of an object. The object must be a Correlation, Geometric, or Polar.
<i>Path</i>	Full path name of the file to load the model from, excluding extension.

**Remarks**

An error will occur if the model data in the file is the wrong type. For example, if you try to load a polar model into a correlation, an error will occur.

If you supply a file extension, it is ignored. There are two files associated with fileName.

For correlation and geometric models, the ModelOrgX and ModelOrgY values are restored along with the model window width and height.

For polar models, the Radius, Thickness, and AngleOffset are restored.

**See Also**

VSaveModel

**VLoadModel Example****VB Example:**

```
m_spel.VLoadModel("seq01", "corr01", "d:\models\part1")
```

**C# Example:**

```
m_spel.VLoadModel("seq01", "corr01", @"d:\models\part1");
```

## VRun Method, Spel Class

**Description**

Run a vision sequence in the current project.

**Syntax**

Sub **VRun** (*Sequence* As String)

**Parameters**

*Sequence* String containing the name of a sequence in the current project.

**Remarks**

VRun works with sequences using any type of camera calibration or no calibration.

To display graphics, you need to use a SPELVideo control and set the SpelVideoControl property of the Spel class instance to the SPELVideo control.

After you execute VRun, use VGet to retrieve results.

**See Also**

VGet, VSet

**VRun Example****VB Example:**

```
Function FindPart(x As Single, y As Single, angle As
Single)
As Boolean
    Dim found As Boolean
    Dim x, y, angle As Single
    With m_spel
        .VRun("seq01")
        .VGet("seq01", "corr01", "found", found)
        If found Then
            .VGet("seq01", "corr01", "cameraX", x)
            .VGet("seq01", "corr01", "cameraY", y)
            .VGet("seq01", "corr01", "angle", angle)
            FindPart = True
        End If
    End With
End Function
```

**C# Example:**

```
bool FindPart(float x, float y, float angle)
{
    bool found;
    m_spel.VRun("seq01");
    m_spel.VGet("seq01", "corr01", "found", found);
    if (found) {
        m_spel.VGet("seq01", "corr01", "cameraX", out x);
        m_spel.VGet("seq01", "corr01", "cameraY", out y);
        m_spel.VGet("seq01", "corr01", "angle", out angle);
    }
    return found;
}
```

**VSave Method, Spel Class****Description**

Saves all vision data in the current project.

**Syntax**

Sub **VSave** ()

**Remarks**

Use **VSave** to make any changes to vision properties permanent.

**See Also**

VSet

**VSave Example****VB Example:**

```
With m_spel
    .VSet("seq01", "blob01", "SearchWinLeft", 100)
    .VSet("seq01", "corr01", "Accept", userAccept)
    .VSave()
End With
```

**C# Example:**

```
m_spel.VSet("seq01", "blob01", "SearchWinLeft", 100);
m_spel.VSet("seq01", "corr01", "Accept", userAccept);
m_spel.VSave();
```

## VSaveImage Method, Spel Class

**Description**

Save a vision video window to a PC disk file.

**Syntax**

Sub **VSaveImage** (*Sequence* As String, *Path* As String)

Sub **VSaveImage** (*Sequence* As String, *Path* As String, *WithGraphics* As Boolean)

**Parameters**

*Sequence* String containing the name of a sequence in the current project.

*Path* Full path name of the file to save the image to, including the extension.

*WithGraphics* Boolean expression that sets whether to save the sequence result graphics in the image file.

**Remarks**

Use VSaveImage to save an image on the Video display to disk. The file extension must be MIM (default format for Vision Guide), BMP, TIF, or JPG.

**See Also**

LoadImage (SPELVideo Control)

**VSaveImage Example****VB Example:**

```
Dim found As Boolean
m_spel.VRun("Seq")
m_spel.VGet("Seq", SpelVisionProps.AllFound, found)
If Not found Then
    m_spel.VSaveImage("Seq", "d:\reject.mim")
End If
```

**C# Example:**

```
bool found;
m_spel.VRun("Seq");
m_spel.VGet("Seq", SpelVisionProps.AllFound, out found);

if (!found)
    m_spel.VSaveImage("Seq", @"d:\reject.mim");
```

## VSaveModel Method, Spel Class

**Description**

Save a vision object search model to a PC disk file.

**Syntax**

Sub **VSaveModel** (*Sequence* As String, *Object* As String, *Path* As String)

**Parameters**

<i>Sequence</i>	String containing the name of a sequence in the current project.
<i>Object</i>	String containing the name of an object. The object must be a Correlation, Geometric, or Polar.
<i>Path</i>	Full path name of the file to save the model to, excluding the extension.

**Remarks**

When **VSaveModel** is executed, EPSON RC+ 7.0 creates two files (*Path* + extensions): *Path.VOB*, *Path.MDL*

For correlation and geometric models, the ModelOrgX and ModelOrgY values are saved along with the model window.

For Polar models, the Radius, Thickness, and AngleOffset are saved.

**See Also**

VLoadModel

**VSaveModel Example****VB Example:**

```
m_spel.VSaveModel("seq01", "corr01", "d:\models\part1")
```

**C# Example:**

```
m_spel.VSaveModel("seq01", "corr01", @"d:\models\part1");
```

## VSet Method, Spel Class

**Description**

Sets the value of a vision sequence or object property.

**Syntax**

```

Sub VSet ( Sequence As String, PropCode As SpelVisionProps, Value As Integer )
Sub VSet ( Sequence As String, PropCode As SpelVisionProps, Value As Boolean )
Sub VSet ( Sequence As String, PropCode As SpelVisionProps, Value As Double )
Sub VSet ( Sequence As String, PropCode As SpelVisionProps, Value As Single )
Sub VSet ( Sequence As String, PropCode As SpelVisionProps, Value As String )
Sub VSet ( Sequence As String, Object As String, PropCode As SpelVisionProps,
  Value As Integer )
Sub VSet ( Sequence As String, Object As String, PropCode As SpelVisionProps,
  Value As Boolean )
Sub VSet ( Sequence As String, Object As String, PropCode As SpelVisionProps, Value
  As Color )
Sub VSet ( Sequence As String, Object As String, PropCode As SpelVisionProps,
  Value As Double )
Sub VSet ( Sequence As String, Object As String, PropCode As SpelVisionProps, Value
  As Single )
Sub VSet ( Sequence As String, Object As String, PropCode As SpelVisionProps,
  Value As String )

```

**Parameters**

<i>Sequence</i>	String expression containing the name of a vision sequence in the current project.
<i>Object</i>	String expression containing the name of an object in sequence <i>Sequence</i> . If the property is for a sequence, then this string must be empty.
<i>PropCode</i>	A SpelVisionProps value that specifies the property code.
<i>Value</i>	Expression containing the new value. The expression type must match the property type.

**See Also**

VGet, VRun

**VSet Example****VB Example:**

```
m_spel.VSet("seq01", "corr01", SpelVisionProps.Accept, 250)
```

**C# Example:**

```
m_spel.VSet("seq01", "corr01", SpelVisionProps.Accept, 250);
```

## VSetModelWin Method, Spel Class

**Description**

Sets model window coordinates.

**Syntax**

Sub **VSetModelWin** ( *Sequence* As String, *Object* As String, *Left* As Integer, *Top* As Integer, *Width* As Integer, *Height* As Integer )

**Parameters**

<i>Sequence</i>	String expression containing the name of a vision sequence in the current project.
<i>Object</i>	String expression containing the name of an object in sequence <i>Sequence</i> .
<i>Left</i>	Integer expression representing left coordinate in pixels.
<i>Top</i>	Integer expression representing top coordinate in pixels.
<i>Width</i>	Integer expression representing width in pixels.
<i>Height</i>	Integer expression representing height in pixels.

**See Also**

VGetMethodWin, VGetSearchWin, VSetSearchWin

**VSetModelWin Example****VB Example:**

```
Dim left As Integer, top As Integer
Dim width As Integer, height As Integer
```

```
With m_spel
    .VGetSearchWin("testSeq", "corr01", left, top, _
        width, height)
    .VSetSearchWin("testSeq", "corr01", left + 50, _
        top - 10, width, height)
    .VRun("testSeq")
End With
```

**C# Example:**

```
int left, top, width, height;

m_spel.VGetSearchWin("testSeq", "corr01", out left, out top,
    out width, out height);
m_spel. .VSetSearchWin("testSeq", "corr01", left + 50,
    top - 10, width, height);
m_spel.VRun("testSeq");
```

## VSetRobotPlaceTargetPos Method, Spel Class

**Description**

Sets part place position.

**Syntax**

Sub **VSetRobotPlaceTargetPos** (*Sequence* As String, *Object* As String, *Point* As SpelPoint)

**Parameters**

*Sequence* String expression containing the name of a vision sequence in the current project.

*Object* String expression containing the name of an object in sequence *Sequence*.

*Point* SpelPoint variable that will contain the place position.

**See Also**

VGetRobotPlacePos, VGetRobotPlaceTargetPos

**VSetRobotPlaceTargetPos Example****VB Example:**

```
Dim seq As String, blob As String
Dim targetPoint As SpelPoint
```

```
seq = "testSeq"
blob = "blob01"
m_spel.VGetRobotPlaceTargetPos(seq, blob, targetPoint)
```

```
' Adjust the place position
```

```
targetPoint.X = targetPoint.X + 10
m_spel.VSetRobotPlaceTargetPos(seq, blob, targetPoint)
```

**C# Example:**

```
string seq, blob;
SpelPoint targetPoint = new SpelPoint();
```

```
seq = "testSeq";
blob = "blob01";
m_spel.VGetRobotPlaceTargetPos(seq, blob, out targetPoint);
```

```
// Adjust the place position
```

```
targetPoint.X = targetPoint.X + 10;
m_spel.VSetRobotPlaceTargetPos(seq, blob, targetPoint);
```



## VSetSearchWin Method, Spel Class

**Description**

Sets search window coordinates.

**Syntax**

Sub **VSetSearchWin** ( *Sequence* As String, *Object* As String, *Left* As Integer, *Top* As Integer, *Width* As Integer, *Height* As Integer )

**Parameters**

<i>Sequence</i>	String expression containing the name of a vision sequence in the current project.
<i>Object</i>	String expression containing the name of an object in sequence <i>Sequence</i> .
<i>Left</i>	Integer expression representing left coordinate in pixels.
<i>Top</i>	Integer expression representing top coordinate in pixels.
<i>Width</i>	Integer expression representing width in pixels.
<i>Height</i>	Integer expression representing height in pixels.

**See Also**

VGetModelWin, VSetModel, VGetSearchWin

**VSetSearchWin Example****VB Example:**

```
Dim left As Integer, top As Integer
Dim width As Integer, height As Integer
```

```
With m_spel
    .VGetSearchWin("testSeq", "corr01", left, top, _
        width, height)
    .VSetSearchWin("testSeq", "corr01", newLeft, top, _
        width, height)
    .VRun("testSeq")
End With
```

**C# Example:**

```
int left, top, width, height;

m_spel.VGetSearchWin("testSeq", "corr01", out left, out top,
    out width, out height);
m_spel. VSetSearchWin("testSeq", "corr01", left + 50,
    top, width, height);
m_spel.VRun("testSeq");
```

## VShowModel Method, Spel Class

**Description**

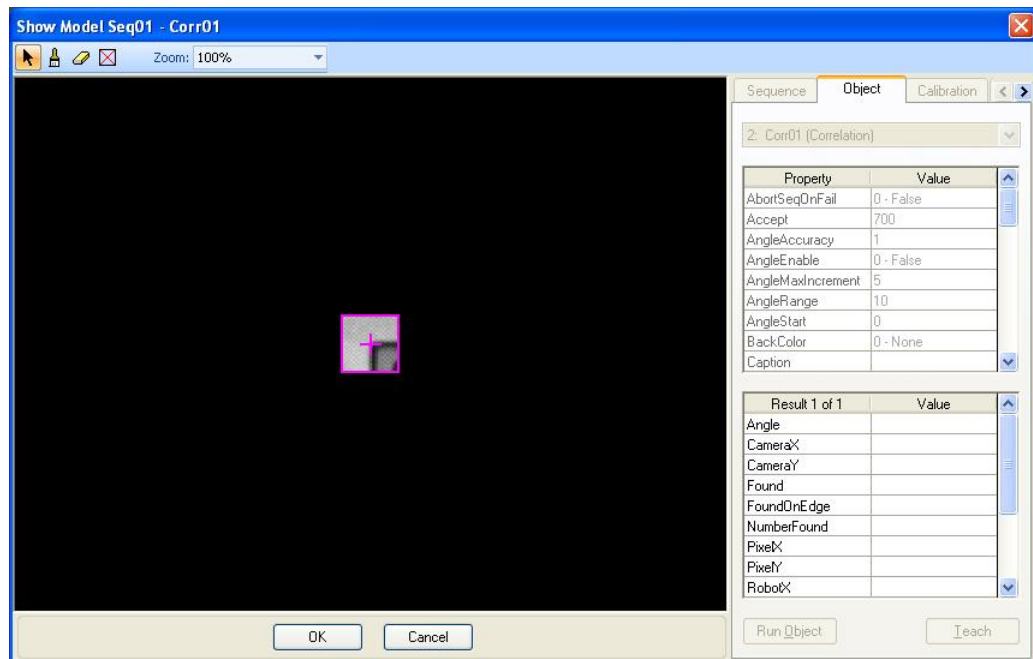
Display the object model. For more details, see the ShowModel Property in the Vision Guide Properties reference.

**Syntax**

Sub **VShowModel** (*Sequence As String, Object As String*)

**Parameters**

- Sequence* String expression containing the name of a vision sequence in the current project.
- Object* String expression containing the name of a vision object in the current project.

**See Also**

VShowSequence, VTrain

**VShowModel Example****VB Example:**

```
m_spel.VShowModel("myseq", "myobj")
```

**C# Example:**

```
m_spel.VShowModel("myseq", "myobj");
```

## VShowSequence Method, Spel Class

**Description**

Displays all objects in a sequence.

**Syntax**

Sub **VShowSequence** (*Sequence* As String)

**Parameters**

*Sequence*                      String expression containing the name of a vision sequence to create.

**Remarks**

Use VShowSequence to display the objects in a sequence without running the sequence. The active object color (magenta) is used for all objects so that they can be seen easily. One use is for when a robot camera is moved over a particular portion of a part being scanned with several sequences. After the robot is positioned, VShowSequence can be called to display the sequence.

**See Also**

VShowModel

**VShowSequence Example****VB Example:**

```
m_spel.VShowSequence("myseq")
```

**C# Example:**

```
m_spel.VShowSequence("myseq");
```

**VStatsReset Method, Spel Class****Description**

Resets vision statistics for a specified sequence in the current project.

**Syntax**

Sub **VStatsReset** (*Sequence* As String)

**Parameters**

*Sequence*     String expression containing the name of a vision sequence in the current project.

**Remarks**

**VStatsReset** resets the statistics for the specified sequence in memory only for the current EPSON RC+ 7.0 session. You should execute **VStatsSave** if you want changes to be permanent. Otherwise, if you restart EPSON RC+ 7.0, the statistics are restored from disk.

**See Also**

**VStatsResetAll**, **VStatsShow**, **VStatsSave**

**VStatsReset Example****VB Example:**

```
Sub btnResetStats_Click()  
    m_spel.VStatsReset("seq01")  
End Sub
```

**C# Example:**

```
void btnResetStats_Click(object sender, EventArgs e)  
{  
    m_spel.VStatsReset("seq01");  
}
```

**VStatsResetAll Method, Spel Class****Description**

Resets vision statistics for all sequences.

**Syntax**

Sub **VStatsResetAll**

**Remarks**

**VStatsResetAll** resets the statistics in memory only for the current EPSON RC+ 7.0 session. You should execute **VStatsSave** if you want changes to be permanent. Otherwise, if you restart EPSON RC+ 7.0, the statistics are restored from disk.

**See Also**

**VStatsReset**, **VStatsShow**, **VStatsSave**

**VStatsResetAll Example****VB Example:**

```
Sub btnResetStats_Click()  
    m_spel.VStatsResetAll()  
End Sub
```

**C# Example:**

```
void btnResetStats_Click(object sender, EventArgs e)  
{  
    m_spel.VStatsResetAll();  
}
```

VStatsSave Method, Spel Class

**Description**

Saves vision statistics for all sequences in the current project.

**Syntax**

Sub VStatsSave ()

**Remarks**

VStatsSave must be executed before EPSON RC+ 7.0 is shut down if you want to preserve changes made to vision statistics.

**See Also**

VStatsReset, VStatsResetAll, VStatsShow

**VStatsSave Example**

**VB Example:**

```
Sub btnResetStats_Click()  
    m_spel.VStatsSave()  
End Sub
```

**C# Example:**

```
void btnResetStats_Click(object sender, EventArgs e)  
{  
    m_spel.VStatsSave();  
}
```

## VStatsShow Method, Spel Class

**Description**

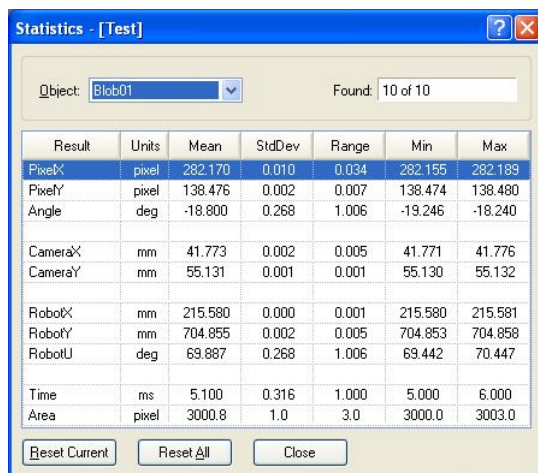
Displays the vision statistics dialog for a specified sequence in the current project.

**Syntax**

Sub **VStatsShow** (*Sequence As String*)

**Parameters**

*Sequence* String expression containing the name of a vision sequence in the current project.



The screenshot shows a window titled "Statistics - [Test]". At the top, there is a dropdown menu for "Object" set to "Blob01" and a text field for "Found:" containing "10 of 10". Below this is a table with the following data:

Result	Units	Mean	StdDev	Range	Min	Max
PixelX	pixel	282.170	0.010	0.034	282.155	282.189
PixelY	pixel	138.476	0.002	0.007	138.474	138.480
Angle	deg	-18.800	0.268	1.006	-19.246	-18.240
CameraX	mm	41.773	0.002	0.005	41.771	41.776
CameraY	mm	55.131	0.001	0.001	55.130	55.132
RoboX	mm	215.580	0.000	0.001	215.580	215.581
RoboY	mm	704.855	0.002	0.005	704.853	704.858
RoboU	deg	69.887	0.268	1.006	69.442	70.447
Time	ms	5.100	0.316	1.000	5.000	6.000
Area	pixel	3000.8	1.0	3.0	3000.0	3003.0

At the bottom of the dialog, there are three buttons: "Reset Current", "Reset All", and "Close".

**See Also**

VStatsReset, VStatsResetAll, VStatsSave

**VStatsShow Example****VB Example:**

```
Sub btnShowStats_Click()
    m_spel.VStatsShow("seq01")
End Sub
```

**C# Example:**

```
void btnShowStats_Click(object sender, EventArgs e)
{
    m_spel.VStatsShow("seq01");
}
```

## VTeach Method, Spel Class

**Description**

Teach a correlation, geometric, or polar model.

**Syntax**

Sub **VTeach** (*Sequence* As String, *Object* As String, *ByRef Status* as Integer)

Sub **VTeach** (*Sequence* As String, *Object* As String, *AddSample* as Boolean, *KeepDontCares* As Boolean, *ByRef Status* as Integer)

**Parameters**

<i>Sequence</i>	The name of a vision sequence in the current project.
<i>Object</i>	The name of an object in <i>Sequence</i> . You can teach Correlation, Geometric, or Polar objects.
<i>AddSample</i>	True if the sample is added, False if a new model is added.
<i>KeepDontCares</i>	True if the old detection mask is used, False if it is disposed.
<i>Status</i>	Return status. 1 if successful, 0 if not.

**Remarks**

Before you call **VTeach**, you must ensure that the model window is in the correct position. For polar objects, the search window and thickness must be set properly. Set the search window location and thickness using VSet.

For correlation and geometric objects, the search window and the model window must be set properly. Set the search and model window locations using VSet for SearchWin and ModelWin. Or you can use the VTrain command so the operator can interactively change the windows.

After teaching the models, you can save them to a PC disk file using the VSaveModel method.

**See Also**

VTrain, VSaveModel

**VTeach Example****VB Example:**

```
Dim status As Integer

' First let the operator change the window position
m_spel.VTrain("seq01", "corr01", status)

' Now teach the model
m_spel.VTeach("seq01", "corr01", status)
```

**C# Example:**

```
int status;

// First let the operator change the window position
m_spel.VTrain("seq01", "corr01", status);

// Now teach the model
m_spel.VTeach("seq01", "corr01", out status);
```



## VTrain Method, Spel Class

**Description**

This command allows you to train objects in an entire sequence or individual objects.

**Syntax**

Function VTrain (Sequence As String [, Object As String] [, Flags as Integer] [, Parent as Form]) As Boolean

**Parameters**

<i>Sequence</i>	The name of a vision sequence in the current project.
<i>Object</i>	The name of an object in <i>Sequence</i> . You can train any type of object. If <i>Object</i> is an empty string, then the entire sequence can be trained.
<i>Flags</i>	Optional. Configures VTrain dialog 1 - Show Teach button 2 - Don't show Model windows.
<i>Parent</i>	Optional. A .NET form that will be the parent of the window.

**Return Values**

If the operator clicks the OK button, VTrain returns True, otherwise it returns False.

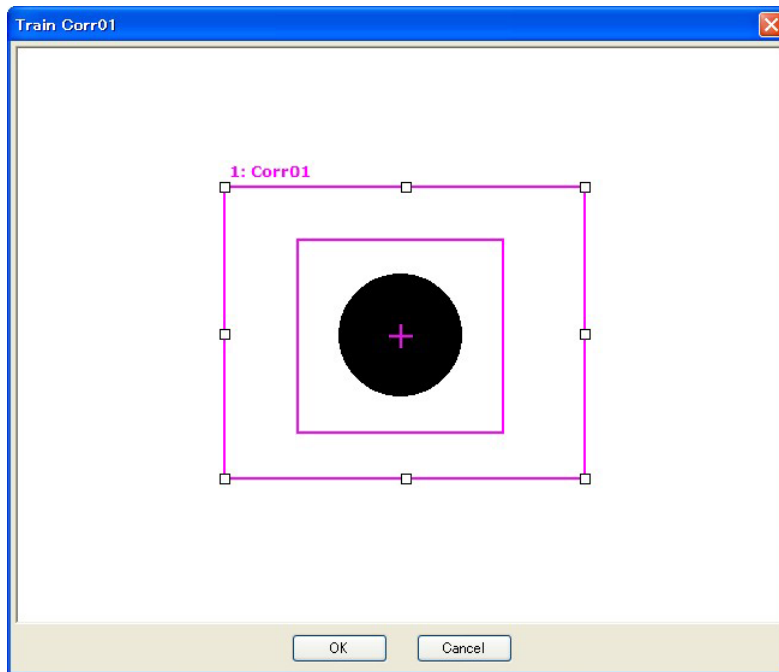
**Remarks**

When **VTrain** is executed, a dialog is opened showing a live video image with the specified object displayed. The operator can size/move the search window, and train the model window (for correlation and geometric objects). When the operator is finished, he can click on OK to save the changes, or Cancel to ignore the changes. If OK is clicked, then the new information is automatically saved in the current project.

If *flags* bit 1 is set, a teach button will be displayed. For Correlation, Geometric, and Polar objects, the model will be taught if the teach button is clicked. You can retrieve the ModelOK property after running VTrain to check if a model was trained. For Blob objects, the button will open the Histogram dialog and the operator can adjust both high and low thresholds and then view the effects of changes.

If *flags* bit 2 is set, model windows will not be displayed. The operator can only change search windows.

For correlation and geometric objects, you can call `VTech` after calling `VTrain` to teach the model if you are not displaying the teach button.



### See Also

`VTech`, `VSaveModel`

### VTrain Example

#### VB Example:

```
Dim status As Integer
Dim trainOK As Boolean
```

```
' First let the operator change the window position
trainOK = m_spel.VTrain("seq01", "corr01")

' Now teach the model
If trainOK Then
    m_spel.VTech("seq01", "corr01", status)
EndIf
```

#### C# Example:

```
int status;
bool trainOK;

// First let the operator change the window position
trainOK = m_spel.VTrain("seq01", "corr01");

// Now teach the model
if (trainOK)
    m_spel.VTech("seq01", "corr01", out status);
```

---

**WaitCommandComplete Method, Spel Class****Description**

This command waits for a command started with AsyncMode = True to complete.

**Syntax**

Sub **WaitCommandComplete** ()

**See Also**

AsyncMode

**WaitCommandComplete Example****VB Example:**

```
With m_spel
    .AsyncMode = True
    .Jump("pick")
    .Delay(500)
    .On(1)
    .WaitCommandComplete()
End With
```

**C# Example:**

```
m_spel.AsyncMode = true;
m_spel.Jump("pick");
m_spel.Delay(500);
m_spel.On(1);
m_spel.WaitCommandComplete();
```

## WaitMem Method, Spel Class

**Description**

Waits for a memory bit status to change.

**Syntax**

Sub **WaitMem** (*BitNumber* As Integer, *Condition* As Boolean, *Timeout* As Single)

Sub **WaitMem** (*Label* As String, *Condition* As Boolean, *Timeout* As Single)

**Parameters**

<i>BitNumber</i>	Integer expression representing the memory bit number.
<i>Label</i>	String representing the memory bit label.
<i>Condition</i>	Boolean expression representing the memory bit status.
<i>Timeout</i>	Single expression representing the maximum time to wait in seconds.

**Remarks**

You should always check if a time out occurred by using the TW method. See the example below.

**See Also**

WaitSw

**WaitMem Example****VB Example:**

```
' Wait for memory bit 1 to be 1 (True)
' Max time is 5 seconds
m_spel.WaitMem(1, True, 5)
' Did WaitMem time out?
If m_spel.TW() Then
    MsgBox "memory bit time out occurred"
End If
```

**C# Example:**

```
// Wait for memory bit 1 to be 1 (True)
// Max time is 5 seconds
m_spel.WaitMem(1, True, 5);
// Did WaitMem time out?
if (m_spel.TW())
    MessageBox.Show("memory bit time out occurred");
```

## WaitSw Method, Spel Class

**Description**

Waits for input bit status to change.

**Syntax**

Sub **WaitSw** (*BitNumber* As Integer, *Condition* As Boolean, *Timeout* As Single)

Sub **WaitSw** (*Label* As String, *Condition* As Boolean, *Timeout* As Single)

**Parameters**

<i>BitNumber</i>	Integer expression representing the input bit number.
<i>Label</i>	String representing the input bit label.
<i>Condition</i>	Boolean expression representing the input bit status.
<i>Timeout</i>	Single expression representing the maximum time to wait in seconds.

**Remarks**

You should always check if a time out occurred by using the TW method. See the example below.

**See Also**

WaitMem

**WaitSw Example****VB Example:**

```
Const PartPresent = 1
m_spel.WaitSw(PartPresent, True, 5)
If m_spel.TW() Then
    MsgBox "Part present time out occurred"
End If
```

**C# Example:**

```
const int PartPresent = 1;
m_spel.WaitSw(PartPresent, True, 5);
if (m_spel.TW())
    MessageBox.Show("Part Present time out occurred");
```

**WaitTaskDone Method, Spel Class****Description**

Waits for a task to finish and returns the status.

**Syntax**

Function **WaitTaskDone** (*TaskNumber* As Integer) As SpelTaskState

Function **WaitTaskDone** (*TaskName* As String) As SpelTaskState

**Parameters**

*TaskNumber*    Task Number to return the execution status of.

*TaskName*     String expression containing the name of the task.

**Return Value**

A SpelTaskState value.

**See Also**

SpelTaskState, TasksExecuting, TaskState, Xqt

**WaitTaskDone Example****VB Example:**

```
Dim taskState As SpelTaskState
m_spel.Xqt 2, "mytask"
'
' Some processing here
'
taskState = m_spel.WaitTaskDone(2)
```

**C# Example:**

```
SpelTaskState taskState;
m_spel.Xqt(2, "mytask");
//
// Some processing here
//
taskState = m_spel.WaitTaskDone(2);
```

## Weight Method, Spel Class

**Description**

Specifies the weight parameters for the current robot.

**Syntax**

Sub **Weight** (*PayloadWeight* As Single, *ArmLength* As Single)

Sub **Weight** (*PayloadWeight* As Single, *Axis* As SpelAxis, [*Axis*])

**Parameters**

<i>PayloadWeight</i>	The weight of the end effector to be carried in Kg units.
<i>ArmLength</i>	The distance from the rotational center of the second arm to the center of the gravity of the end effector in mm units.
<i>Axis</i>	Specifies which additional axis (S or T) is assign the payload weight.

**Note**

Do not enter integer values to PayloadWeight and ArmLength parameters. Use Single variables or directly enter Single type values.

**See Also**

Inertia, JRange, Tool

**Weight Example****VB Example:**

```
m_spel.Weight(2.0F, 2.5F)
```

**C# Example:**

```
m_spel.Weight(2.0F, 2.5F);
```

## Xqt Method, Spel Class

**Description**

Start one SPEL<sup>+</sup> task.

**Syntax**

Sub **Xqt** (*FuncName* As String [, *TaskType* As SpelTaskType])

Sub **Xqt** (*TaskNumber* As Integer, *FuncName* As String [, *TaskType* As SpelTaskType])

**Parameters**

*TaskNumber* The task number for the task to be executed. The range of the task number is 1 to 32.

*FuncName* The name of the function to be executed. You can also optionally supply arguments to the function. Arguments must be in parenthesis, separated by commas. For details, see the SPEL<sup>+</sup> Xqt Statement. Also, see the example.

*TaskType* Optional. Specifies the task type as Normal, NoPause, or NoEmgAbort.

**Remarks**

When **Xqt** is executed, control will return immediately to the calling program. Use the Call method to wait for a task to complete, or you can use EventReceived with the task state event to wait for a task to finish.

**See Also**

Call, EnableEvent, EventReceived

**Xqt Example****VB Example:**

```
m_spel.Xqt(2, "conveyor")
```

' Supply an argument to the RunPart function

```
m_spel.Xqt(3, "RunPart(3)")
```

```
Dim funcToExec As String
```

```
funcToExec = "RunPart(" & partNum & ")"
```

```
m_spel.Xqt(3, funcCall)
```

**C# Example:**

```
m_spel.Xqt(2, "conveyor");
```

// Supply an argument to the RunPart function

```
m_spel.Xqt(3, "RunPart(3)");
```

```
string funcToExec;
```

```
funcToExec = string.Format("RunPart({0})", partNum);
```

```
m_spel.Xqt(3, funcToExec);
```



## XYLim Method, Spel Class

**Description**

Sets the permissible motion range limits for the Manipulator.

**Syntax**

Sub **XYLim** (*XLowerLimit* As Single, *XUpperLimit* As Single, *YLowerLimit* As Single, *YUpperLimit* As Single [, *ZLowerLimit* As Single ] [, *ZUpperLimit* As Single ] )

**Parameters**

<i>XLowerLimit</i>	The minimum X coordinate position to which the Manipulator may travel. (The Manipulator may not move to a position with the X Coordinate less than minX.)
<i>XUpperLimit</i>	The maximum X coordinate position to which the Manipulator may travel. (The Manipulator may not move to a position with the X Coordinate greater than maxX.)
<i>YLowerLimit</i>	The minimum Y coordinate position to which the Manipulator may travel. (The Manipulator may not move to a position with the Y Coordinate less than minY.)
<i>YUpperLimit</i>	The maximum Y coordinate position to which the Manipulator may travel. (The Manipulator may not move to a position with the Y Coordinate greater than maxY.)
<i>ZLowerLimit</i>	Optional. The minimum Z coordinate position to which the Manipulator may travel. (The Manipulator may not move to a position with the Z Coordinate less than minZ.)
<i>ZUpperLimit</i>	Optional. The maximum Z coordinate position to which the Manipulator may travel. (The Manipulator may not move to a position with the Z Coordinate greater than maxZ.)

**Remarks**

XYLim is used to define motion range limits. Many robot systems allow users to define joint limits but the SPEL+ language allows both joint limits and motion range limits to be defined. In effect this allows users to create a work envelope for their application. (Keep in mind that joint range limits are also definable with SPEL.)

The motion range established with XYLim values applies to motion command target positions only, and not to motion paths from starting position to target position. Therefore, the arm may move outside the XYLim range during motion. (i.e. The XYLim range does not affect Pulse.)

To turn off motion range limits, specify 0 for the range limit parameters.

**See Also**

JRange

**XYLim Example****VB Example:**

```
m_spel.XYLim(0, 0, 0, 0)
```

**C# Example:**

```
m_spel.XYLim(0, 0, 0, 0);
```

**XYLimClr Method, Spel Class**

**Description**

Clears (undefines) the XYLim definition.

**Syntax**

Sub **XYLimClr** ()

**See Also**

XYLim, XYLimDef

**XYLimClr Example**

**VB Example:**

```
m_spel.XYLimClr()
```

**C# Example:**

```
m_spel.XYLimClr();
```

## XYLimDef Method, Spel Class

**Description**

Returns whether XYLim has been defined or not.

**Syntax**

Function **XYLimDef** () As Boolean

**Return Value**

True if XYLim is defined, False if not.

**See Also**

XYLim, XYLimClr

**XYLimDef Example****VB Example:**

```
Dim xyLimDefined As Boolean
xyLimDefined = m_spel.XYLimDef()
```

**C# Example:**

```
bool xyLimDefined;
xyLimDefined = m_spel.XYLimDef();
```

## 14.4 Spel Class Events

### EventReceived Event, Spel Class

#### Description

Occurs when EPSON RC+ 7.0 sends a system event or when a program running in SPEL<sup>+</sup> sends an event using a SPELCom\_Event statement.

#### Syntax

**EventReceived** (ByVal *sender* As Object, ByVal *e* As RCAPINet.SpelEventArgs)

#### Parameters

*e.Event*            Number representing a specific user-defined event.

*e.Message*        String containing event message.

#### Remarks

There are several system events that EPSON RC+ 7.0 issues. The following table describes them.

#### System Events

Some events are disabled by default. To use these events you must first enable them using the EnableEvent Method.

Event Number	Event Message	Constant	Description
1	"PAUSE"	SpelEvents.Pause	Occurs when tasks are paused. Enabled by default.
2	"SAFE GUARD OPEN"	SpelEvents.SafeGuardOpen	Occurs when safe guard is open. Enabled by default.
3	"SAFE GUARD CLOSE"	SpelEvents.SafeGuardClose	Occurs when safe guard is closed. Enabled by default.
4	Project build status text	SpelEvents.ProjectBuildStatus	Each build status message is sent during the BuildProject method. CRLFs are added as needed. These messages are the same ones displayed on the Project Build Status window in EPSON RC+ 7.0 GUI. This event must be enabled with the EnableEvent method. Disabled by default.
5	"Error xxx!: mmm in task at line yyy"	SpelEvents.Error	Occurs when a task is aborted due to an unhandled error or a system error is generated. Enabled by default.
6	Text from print statement	SpelEvents.Print	Occurs when a Print statement executes from a SPEL <sup>+</sup> task. Disabled by default.
7	"ESTOP ON"	SpelEvents.EStopOn	Occurs when emergency stop condition changes to ON. Enabled by default.

Event Number	Event Message	Constant	Description
8	"ESTOP OFF"	SpelEvents.EStopOff	Occurs when emergency stop condition changes to OFF. Enabled by default.
9	"CONTINUE"	SpelEvents.Continue	Occurs after a Cont has been executed. Enabled by default.
10	<Robot #>,"MOTOR ON"	SpelEvents.MotorOn	Occurs when motors go ON for the robot indicated. Disabled by default.
11	<Robot #>,"MOTOR OFF"	SpelEvents.MotorOff	Occurs when motors go OFF for the robot indicated. Disabled by default.
12	<Robot #>,"POWER HIGH"	SpelEvents.PowerHigh	Occurs when power goes HIGH for the robot indicated. Disabled by default.
13	<Robot #>,"POWER LOW"	SpelEvents.PowerLow	Occurs when power goes LOW for the robot indicated. Disabled by default.
14	"TEACH MODE"	SpelEvents.TeachMode	Occurs when teach mode is activated. Enabled by default.
15	"AUTO MODE"	SpelEvents.AutoMode	Occurs when auto mode is activated. Enabled by default.
16	"<TaskID>,<Status>,<FuncName>" Status: "RUN", "HALT", "PAUSE", "FINISHED", "ABORTED"	SpelEvents.TaskState	Occurs when task state changes. Disabled by default.
17	"SHUTDOWN"	SpelEvents.Shutdown	Occurs when RC+ is shutting down. Disabled by default.
18	"ALL TASKS STOPPED"	SpelEvents.AllTasksStopped	Occurs when all tasks have been stopped. Disabled by default.
19	"DISCONNECTED"	SpelEvents.Disconnected	Occurs when Controller communication has been disconnected from the PC. When enabled, RC+ does not display a message box indicating disconnection. Disabled by default.
20	"MOTION STARTED"	SpelEvents.MotionStarted	Occurs when the control command started Disabled by default
21	"MOTION COMPLETE"	SpelEvents.MotionComplete	Occurs when the control command started Disabled by default

## User Events

You can send events from your SPEL<sup>+</sup> program to your Visual Basic application using the **SPELCom\_Event** command.

```
Spelcom_Event 3000, cycNum, lotNum, cycTime
```

When this statement executes, the EventReceived routine will be called with the event number and message. See *EPSON RC+ 7.0 Online Help* or *13. SPELCom\_Event* for details on SPELCom\_Event.

## Use Example

### VB Example:

```
Sub m_spel_EventReceived ( _  
    ByVal sender As Object, _  
    ByVal e As RCAPINet.SpelEventArgs) _  
    Handles m_spel.EventReceived  
    Redim tokens(0) As String  
    Select Case e.Event  
        Case 3000  
            tokens = e.Message.Split(New [Char]() {" "c}, _  
  
System.StringSplitOptions.RemoveEmptyEntries)  
            lblCycCount.Text = tokens(0)  
            lblLotNumber.Text = tokens(1)  
            lblCycTime.Text = tokens(2)  
    End Select  
End Sub
```

### C# Example:

```
public void m_spel_EventReceived(object sender,  
    SpelEventArgs e)  
{  
    string[] tokens = new string[3];  
    switch ((int) e.Event){  
        case 3000:  
            tokens = e.Message.Split(' ');  
            lblCycCount.Text = tokens(0);  
            lblLotNumber.Text = tokens(1);  
            lblCycTime.Text = tokens(2);  
            break;  
        default:  
            break;  
    }  
}
```

## Handling Events

When **EventReceived** is called from the Spel class instance, the EPSON RC+ 7.0 process server will wait for the event handling routine to finish. Therefore, you should never try to execute any RC+ API commands or perform long running processing from within the **EventReceived** routine. If you want to execute commands based on an event that occurred, set a flag in **EventReceived** and handle the flag from the main loop of your application, outside of the event handling function.

For example, in your Visual Basic main form Load procedure, you can create an event loop that receives events from SPEL+. In the spel\_EventReceived routine, set global flags to indicate which events were received. Then, you can execute an actual event handling from the event loop created in Load procedure.

### To display event message

Add a TextBox control to a form.

Each time the event is received, you can display the event message in the Text property of the TextBox control.

#### VB Example:

```
Private Sub m_spel_EventReceived(ByVal sender As Object, _
    ByVal e As SpelEventArgs) Handles
    m_spel.EventReceived
    txtEvents.AppendText(e.Event & ": " & e.Message &
vbCrLf)
End Sub
```

#### C# Example:

```
private void m_spel_EventReceived(object sender,
SpelEventArgs e)
{
    txtEvents.AppendText(e.Event + ": " + e.Message);
}
```

### See Also

EnableEvent (Spel Class)

## 14.5 SPELVideo Control

### Description

This control allows you to display video from Vision system. For details on how to use this control, see chapter 11, *Displaying Video*.

### File Name

RCAPINet.dll

## 14.6 SPELVideo Control Properties

This control supports the properties listed below in addition to standard .NET component properties, such as Left, Top, Width, and Height. See the Visual Basic on-line Help for documentation on the standard properties.

- Camera
- GraphicsEnabled
- VideoEnabled

### Camera Property, SPELVideo Control

#### Description

Sets/gets the camera number to display video from. This is useful when you want to display video during jogging operations, live video monitoring, etc. If you are using the control to display graphics for vision sequences, then when the sequence is run, the camera number for the sequence will be used instead of this property value.

#### Syntax

Property **Camera** As Integer

#### Default Value

0 – any camera is displayed

#### Return value

Integer value containing the current camera number

#### See Also

VideoEnabled, GraphicsEnabled

#### Examples

##### VB Example:

```
SpelVideo1.Camera = 1
```

##### C# Example:

```
SpelVideo1.Camera = 1;
```



---

**GraphicsEnabled Property, SPELVideo Control****Description**

Sets / returns whether vision graphics are displayed after a sequence is run. In order to see graphics, you must attach the control to a Spel class instance using the SPELVideo Control property. This property can be set "on the fly" so that graphics can be turned on/off while sequences are being run.

**Syntax**

Property **GraphicsEnabled** As Boolean

**Default Value**

False

**Return value**

True if vision graphics are displayed, False if not.

**See Also**

Camera, VideoEnabled

**Examples****VB Example:**

```
SpelVideo1.GraphicsEnabled = True
```

**C# Example:**

```
SpelVideo1.GraphicsEnabled = true;
```

VideoEnabled Property, SPELVideo Control

**Description**

Determines whether video is displayed.

**Syntax**

Property **VideoEnabled** As Boolean

**Default Value**

False

**Return value**

True if video is displayed, False if not.

**See Also**

Camera, GraphicsEnabled

**Examples**

**VB Example:**

```
SpelVideo1.VideoEnabled = True
```

**C# Example:**

```
SpelVideo1.VideoEnabled = true;
```

## 14.7 SPELVideo Control Methods

### LoadImage Method, SPELVideo Control

**Description**

Loads an image from a file for display.

**Syntax**

Sub **LoadImage** (*Path* As String)

**Parameters**

*Path* Full path name of the file to load the image from, including the extension.

**Remarks**

Use LoadImage to load a previously saved image for display. The file extension must be BMP, TIF, or JPG.

**See Also**

VSaveImage (Spel class)

**LoadImage Example****VB Example:**

```
m_spelVideo.LoadImage("c:\RejectImages\reject001.bmp")
```

**C# Example:**

```
m_spelVideo.LoadImage(@"c:\RejectImages\reject001.bmp");
```

## 14.8 SPELVideo Control Events

All of the events for this control are standard .NET events. See the Visual Basic on-line Help for details.

## 14.9 SpelConnectionInfo Class

Member name	Type	Description
ConnectionIPAddress	String	IP address as configured in EPSON RC+.
ConnectionNumber	Integer	The number of the connection as configured in EPSON RC+.
ConnectionName	String	The name of the connection as configured in EPSON RC+.
ConnectionType	SpelConnectionType	The type of the connection as configured in EPSON RC+.

When the type of the connection is USB or virtual Controller, ConnectionIPAddress is blank.

Here is an example.

### VB Example:

```
Dim connectionInfo() As SpelConnectionInfo
connectionInfo = m_spel.GetConnectionInfo()
```

### C# Example:

```
SpelConnectionInfo[] info = m_spel.GetConnectionInfo();
```

## 14.10 SpelControllerInfo Class

Member name	Type	Description
ProjectName	String	The name of the project in the Controller.
ProjectID	String	The unique project ID of the project in the Controller.
Options	List<SpelOptionInfo>	The list of options in the controller.

Here is an example.

### VB Example:

```
Dim info As RCAPINet.SpelControllerInfo
info = m_spel.GetControllerInfo()
Label1.Text = info.ProjectID + " " + info.ProjectName
```

### C# Example:

```
SpelControllerInfo info;
info = m_spel.GetControllerInfo();
Label1.text = info.ProjectID + " " + info.ProjectName;
```

## 14.11 SpelException Class

The SpelException class is derived from the ApplicationException class. It adds an ErrorNumber property and some constructors.

Here is an example, showing how to retrieve the error number and the error message.

### VB Example:

```
Try
    m_spel.Go(1)
Catch (ex As RCAPINet.SpelException)
    MsgBox(ex.ErrorNumber & ": " & ex.Message)
End Try
```

### C# Example:

```
try{
    m_spel.Go(1);
}
catch(RCAPINet.SpelException ex){
    MessageBox.Show(string.Format("{0}: {1}", ex.ErrorNumber,
ex.Message));
}
```

### SpelException Properties

ErrorNumber As Integer

### SpelException Methods

#### Sub New ()

The default constructor.

#### Sub New (Message As String)

The optional constructor that specifies an error message.

#### Sub New (ErrorNumber As Integer, Message As String)

The optional constructor that specifies the error number and associated message.

#### Sub New (Message As String, Inner As Exception)

The optional constructor that specifies the error message and inner exception.

#### Sub New (ErrorNumber As Integer, Message As String, Inner As Exception)

The optional constructor that specifies the error number, error message, and inner exception.

## 14.12 SpelOptionInfo Class

Member name	Type	Description
Name	String	Robot name
Status	SpelOptionStatus	Robot number

Here is an example.

### VB Example:

```
Dim info As SpelControllerInfo
info = m_spel.GetControllerInfo()
Label1.Text = info.Options(1).Name + " " +
info.Options(1).Status
```

### C# Example:

```
SpelControllerInfo info;
info = m_spel.GetControllerInfo();
Label1.Text = info.Options[1].Name + " " +
info.Options[1].Status;
```

## 14.13 SpelPoint Class

The SpelPoint class can be used in several motion methods and also in the GetPoint and SetPoint methods of Spel class.

Here are some Visual Basic examples:

```
1:
Dim pt As New RCAPINet.SpelPoint(25.5, 100.3, -21, 0)
m_spel.Go(pt)
2:
Dim pt As New RCAPINet.SpelPoint
pt.X = 25.5
pt.Y = 100.3
pt.Z = -21
m_spel.Go(pt)
3:
Dim pt As New RCAPINet.SpelPoint
pt = m_spel.GetPoint("P*")
pt.Y = 222
m_spel.Go(pt)
```

Here are some Visual C# examples:

1:

```
SpelPoint pt = new SpelPoint(25.5, 100.3, -21, 0);  
m_spel.Go(pt);
```

2:

```
SpelPoint pt = new SpelPoint();  
pt.X = 25.5;  
pt.Y = 100.3;  
pt.Z = -21;  
m_spel.Go(pt);
```

3:

```
SpelPoint pt = new SpelPoint();  
pt = m_spel.GetPoint("P0");  
pt.Y = 222;  
m_spel.Go(pt);
```

## 14.13.1 SpelPoint Properties

**VB Example:**

X As Single  
Y As Single  
Z As Single  
U As Single  
V As Single  
W As Single  
R As Single  
S As Single  
T As Single  
Hand As SpelHand  
Elbow As SpelElbow  
Wrist As SpelWrist  
Local As Integer  
J1Flag As Integer  
J2Flag As Integer  
J4Flag As Integer  
J6Flag As Integer  
J1Angle As Single  
J4Angle As Single

**C# Example:**

float x  
float y  
float z  
float u  
float v  
float w  
float r  
float s  
float t  
SpelHand Hand  
SpelElbow Elbow  
SpelWrist Wrist  
int Local  
int J1Flag  
int J2Flag  
int J4Flag  
int J6Flag  
float J1Angle  
float J4Angle



### 14.13.2 SpelPoint Methods

**Sub Clear ()**

Clears all point data.

**Sub New ()**

The default constructor. Creates an empty point (all data is cleared).

**Sub New ( X As Single, Y As Single, Z As Single, U As Single [, V As Single] [, W As Single] )**

The optional constructor for a new point that specifies coordinates.

**Function ToString ([Format As String]) As String**

Override for ToString that allows a Format to be specified. This returns the point as defined in SPEL<sup>+</sup>.

Format can be:

Empty        Returns the entire point with all coordinates and attributes.

"XY"         Returns "XY(...)"

"XYST"      Returns "XY(...) :ST(...)"

## 14.14 SpelRobotInfo Class

Member name	Type	Description
RobotModel	String	Robot model name.
RobotName	String	Robot name.
RobotNumber	Integer	Robot number.
RobotSerial	String	Robot serial number.
RobotType	SpelRobotType	Robot type.

Here is an example.

### VB Example:

```
Dim info As SpelRobotInfo
info = m_spel.GetRobotInfo()
Label1.Text = info.RobotNumber + " " + info.RobotModel
```

### C# Example:

```
SpelRobotInfo info;
info = m_spel.GetRobotInfo();
Label1.Text = info.RobotNumber + " " + info.RobotModel;
```

## 14.15 SpelTaskInfo Class

Member name	Type	Description
CPU	Integer	CPU Load factor for SPEL tasks.
ErrorCode	Integer	Error number.
StartTime	DateTime	Start date and time of the task.
TaskName	String	Task name.
TaskNumber	Integer	Task number.
TaskState	SpelTaskState	Task status.

Here is an example.

### VB Example:

```
Dim info As SpelTaskInfo
info.TaskState = m_spel.TaskState(1)
Label1.Text = info.TaskNumber + " " + info.TaskState
```

### C# Example:

```
SpelTaskInfo info;
info.TaskState = m_spel.TaskState(1);
Label1.Text = info.TaskNumber + " " + info.TaskState;
```

## 14.16 Enumerations

### 14.16.1 SpelArmDefMode Enumeration

Member name	Value	Description
Rough	1	Define the arm using one posture.
Fine	1	Define the arm using two postures.

### 14.16.2 SpelArmDefType Enumeration

Member name	Value	Description
J2Camera	1	Define the arm for a J2 mounted camera.

### 14.16.3 SpelAxis Enumeration

Member name	Value	Description
X	1	X axis.
Y	2	Y axis.
Z	3	Z axis.
U	4	U axis.
V	5	V axis.
W	6	W axis.
R	7	R axis.
S	8	S axis.
T	9	T axis.

### 14.16.4 SpelBaseAlignment Enumeration

Member name	Value	Description
XAxis	0	Align with X axis.
YAxis	1	Align with Y axis.

### 14.16.5 SpelCalPlateType Enumeration

Member name	Value	Description
None	0	No calibration plate.
Large	1	Large calibration plate.
Medium	2	Medium calibration plate.
Small	3	Small calibration plate.
XSmall	4	Extra small calibration plate.

### 14.16.6 SpelConnectionType Enumeration

Member name	Value	Description
USB	1	USB connection.
Ethernet	2	Ethernet connection.
Virtual	3	Connection to virtual Controller.

#### 14.16.7 SpelDialogs Enumeration

Member name	Value	Description
RobotManager	1	ID for Tools   Robot Manager dialog
ControllerTools	2	ID for Tools   Controller dialog
VisionGuide	3	ID for Tools   Vision Guide dialog
ForceGuide	4	ID for Tools   Force Guide dialog
PartFeeding	5	ID for Tools   Part Feeding dialog

#### 14.16.8 SpelElbow Enumeration

Member name	Value	Description
Above	1	Elbow orientation is above.
Below	2	Elbow orientation is below.

#### 14.16.9 SpelEvents Enumeration

Member name	Value	Description
Pause	1	ID for pause event.
SafeguardOpen	2	ID for safeguard open event.
SafeguardClose	3	ID for safeguard close event.
ProjectBuildStatus	4	ID for project build status event.
Error	5	ID for error event.
Print	6	ID for print event.
EstopOn	7	ID for emergency stop on event.
EstopOff	8	ID for emergency stop off event.
Continue	9	ID for continue event.
MotorOn	10	ID for motor on event.
MotorOff	11	ID for motor off event.
PowerHigh	12	ID for power high event.
PowerLow	13	ID for power low event.
TeachMode	14	ID for teach mode event.
AutoMode	15	ID for auto mode event.
TaskState	16	ID for task state event.
Shutdown	17	ID for shutdown event.
AllTasksStopped	18	ID for all tasks stopped event.
Disconnected	19	ID for disconnection event.
MotionStarted	20	ID for control command start event.
MotionComplete	21	ID for control command complete event.

#### 14.16.10 SpelForceAxis Enumeration

Member name	Value	Description
XForce	1	Specifies the X force axis.
YForce	2	Specifies the Y force axis.
ZForce	3	Specifies the Z force axis.
XTorque	4	Specifies the X torque axis.
YTorque	5	Specifies the Y torque axis.
ZTorque	6	Specifies the Z torque axis.

#### 14.16.11 SpelForceCompareType Enumeration

Member name	Value	Description
LessOrEqual	0	Till is triggered when the force is less than or equal to the specified threshold.
GreaterOrEqual	1	Till is triggered when the force is greater than or equal to the specified threshold.

#### 14.16.12 SpelForceProps Enumeration

Member name	Value	Description
EndStatus	28340	End status for a Force Guide sequence or object.
ForceCondOK	28440	Status of force-related end condition of a Force Guide object.
IOCondOK	28590	Status of I/O-related end condition of a Force Guide object.
PosCondOK	28860	Status of achievement of position-related end condition of a Force Guide object.
Time	29070	Execution time for a Force Guide sequence or object.
TimeOut	29080	Reached status to the timeout period of Force Guide object.
LastExecObject	30100	Name of the Force Guide object that was executed at the end.
MeasuredHeight	30500	Measured height of HeightInspect sequence.
FailedStatus	30510	The reason why the force guide sequence failed.
AvgForcesFx	100211	Average value of force in Fx direction during execution of a Force Guide object.
AvgForcesFy	100212	Average value of force in Fy direction during execution of a Force Guide object.
AvgForcesFz	100213	Average value of force in Fz direction during execution of a Force Guide object.
AvgForcesTx	100214	Average value of force in Tx direction during execution of a Force Guide object.
AvgForcesTy	100215	Average value of force in Ty direction during execution of a Force Guide object.
AvgForcesTz	100216	Average value of force in Tz direction during execution of a Force Guide object.

EndForcesFx	102411	Force in Fx direction at end of a Force Guide sequence or Force Guide object.
EndForcesFy	102412	Force in Fy direction at end of a Force Guide sequence or Force Guide object.
EndForcesFz	102413	Force in Fz direction at end of a Force Guide sequence or Force Guide object.
EndForcesTx	102414	Force in Tx direction at end of a Force Guide sequence or Force Guide object.
EndForcesTy	102415	Force in Ty direction at end of a Force Guide sequence or Force Guide object.
EndForcesTz	102416	Force in Tz direction at end of a Force Guide sequence or Force Guide object.
EndPosX	102421	Position at end of a Force Guide object (X coordinate).
EndPosY	102422	Position at end of a Force Guide object (Y coordinate).
EndPosZ	102423	Position at end of a Force Guide object (Z coordinate).
EndPosU	102424	Position at end of a Force Guide object (U coordinate).
EndPosV	102425	Position at end of a Force Guide object (V coordinate).
EndPosW	102426	Position at end of a Force Guide object (W coordinate).
PeakForcesFx	105711	Peak value of force in Fx direction during execution of a Force Guide sequence or object.
PeakForcesFy	105712	Peak value of force in Fy direction during execution of a Force Guide sequence or object.
PeakForcesFz	105713	Peak value of force in Fz direction during execution of a Force Guide sequence or object.
PeakForcesTx	105714	Peak value of force in Tx direction during execution of a Force Guide sequence or object.
PeakForcesTy	105715	Peak value of force in Ty direction during execution of a Force Guide sequence or object.
PeakForcesTz	105716	Peak value of force in Tz direction during execution of a Force Guide sequence or object.
TriggeredForcesFx	109411	Force in Fx direction when force-related end conditions of a Force Guide object are achieved.
TriggeredForcesFy	109412	Force in Fy direction when force-related end conditions of a Force Guide object are achieved.
TriggeredForcesFz	109413	Force in Fz direction when force-related end conditions of a Force Guide object are achieved.
TriggeredForcesTx	109414	Force in Tx direction when force-related end conditions of a Force Guide object are achieved.
TriggeredForcesTy	109415	Force in Ty direction when force-related end conditions of a Force Guide object are achieved.
TriggeredForcesTz	109416	Force in Tz direction when force-related end conditions of a Force Guide object are achieved.
TriggeredPosX	109421	Position for a Force Guide object when force-related end conditions are achieved (X coordinate).
TriggeredPosY	109422	Position for a Force Guide object when force-related end conditions are achieved (Y coordinate).

TriggeredPosZ	109423	Position for a Force Guide object when force-related end conditions are achieved (Z coordinate).
TriggeredPosU	109424	Position for a Force Guide object when force-related end conditions are achieved (U coordinate).
TriggeredPosV	109425	Position for a Force Guide object when force-related end conditions are achieved (V coordinate).
TriggeredPosW	109426	Position for a Force Guide object when force-related end conditions are achieved (Z coordinate).

#### 14.16.13 SpelHand Enumeration

Member name	Value	Description
Righty	1	Hand orientation is righty.
Lefty	2	Hand orientation is lefty.

#### 14.16.14 SpellOLabelTypes Enumeration

Member name	Value	Description
InputBit	1	Specifies input bit.
InputByte	2	Specifies input byte.
InputWord	3	Specifies input word.
OutputBit	4	Specifies output bit.
OutputByte	5	Specifies output byte.
OutputWord	6	Specifies output word.
MemoryBit	7	Specifies memory bit.
MemoryByte	8	Specifies memory byte.
MemoryWord	9	Specifies memory word.
InputReal	10	Specifies real number input.
OutputReal	11	Specifies real number output.

#### 14.16.15 SpelLocalDefType Enumeration

Member name	Value	Description
J5Camera	1	Defines the local for a J5 mounted camera.
J6Camera	2	Defines the local for a J6 mounted camera.
FixedUpwardCamera	3	Defines the local by using a fixed upward camera.
FixedDownwardCamera	4	Defines the local by using a fixed downward camera.

#### 14.16.16 SpelOperationMode Enumeration

Member name	Value	Description
Auto	1	EPSON RC+ 7.0 is in auto mode.
Program	2	EPSON RC+ 7.0 is in program mode.

## 14.16.17 SpelOptions Enumeration

Member name	Value	Description
ECP	1	ECP option
API	2	RC+ API option
PCVision	3	PC Vision option
ConveyorTracking	5	Conveyor tracking option
GUIBuilder	6	GUI Builder option
OCR	7	OCR option
FieldbusMaster	8	Fieldbus master option
LegacyForceSensing	9	Force sensing option
PartFeeding	11	Part Feeding option
ThirdPartyForceSensors	13	Third party force sensor option

## 14.16.18 SpelOptionStatus Enumeration

Member name	Value	Description
Inactive	0	Disable
Active	1	Enable

## 14.16.19 SpelRobotPosType Enumeration

Member name	Value	Description
World	0	Specifies world coordinates.
Joint	1	Specifies joint coordinates.
Pulse	2	Specifies pulses.

## 14.16.20 SpelRobotType Enumeration

Member name	Value	Description
Joint	1	Robot type is joint.
Cartesian	2	Robot type is Cartesian.
Scara	3	Robot type is SCARA.
Cylindrical	4	Robot type is Cylindrical.
SixAxis	5	Robot type is 6-axis.
RS	6	Robot type is SCARA RS series.
N	7	Robot type is N.

## 14.16.21 SpelShutdownMode Enumeration

Member name	Value	Description
ShutdownWindows	0	Windows will be shutdown.
RebootWindows	1	Windows will be rebooted.



## 14.16.22 SpelSimObjectType Enumeration

Member name	Value	Description
Unknown	-1	Object type is not set.
Layout	0	Layout object.
Part	1	Parts object.
MountedDevice	3	Arm mount device.

## 14.16.23 SpelSimProps Enumeration

Member name	Value	Description
PositionX	100	Position of X coordinate system.
PositionY	200	Position of Y coordinate system.
PositionZ	300	Position of Z coordinate system.
RotationX	400	Rotation angle of X axis.
RotationY	500	Rotation angle of Y axis.
RotationZ	600	Rotation angle of Z axis.
CollisionCheck	700	Sets enable/disable of collision detect.
CollisionCheckSelf	800	Sets enable/disable of self-collision detect of the robot.
Visible	900	State of display/non-display.
Type	1000	Types of objects.
HalfSizeX	1500	Length of Box object in X direction.
HalfSizeY	1600	Length of Box object in Y direction.
HalfSizeZ	1700	Length of Box object in Z direction.
HalfSizeHeight	1800	Height of Plane object.
HalfSizeWidth	1900	Width of Plane object.
PlaneType	2000	Type of Plane object.
Radius	2100	Radius of Sphere or Cylinder object.
Height	2200	Height of Cylinder object.
Name	2300	Names of objects.
Color	2400	Display color of objects.

## 14.16.24 SpelStopType Enumeration

Member name	Value	Description
StopNormalTasks	0	Stop only normal tasks (not background tasks).
StopAllTasks	1	Stop all tasks, including background tasks.

## 14.16.25 SpelTaskState Enumeration

Member name	Value	Description
Quit	0	Task is in the quit state.
Run	1	Task is in the run state.
Aborted	2	Task was aborted.
Finished	3	Task was finished.
Breakpoint	4	Task is at a breakpoint.
Halt	5	Task is in the halt state.
Pause	6	Task is in the pause state.
Step	7	Task is being stepped.
Walk	8	Task is being walked.
Error	9	Task is in the error state.
Waiting	10	Task is in the wait state.

## 14.16.26 SpelTaskType Enumeration

Member name	Value	Description
Normal	0	Task is a normal task.
NoPause	1	Task is not affected by pause.
NoEmgAbort	2	Task is not affected by emergency stop.

## 14.16.27 SpelToolDefType Enumeration

Member name	Value	Description
J4Camera	1	Define the tool for a J4 mounted camera.
J6Camera	2	Define the tool for a J6 mounted camera.
FixedCamera	3	Define the tool by using the fixed camera which is not calibrated.
FixedCameraWithCal	4	Define the tool by using the upward camera which is calibrated.

## 14.16.28 SpelToolDefType3D Enumeration

Member name	Value	Description
Bar	1	Define a 3D tool for a bar.
Plane	2	Define a 3D tool for a plane.

#### 14.16.29 SpelUserRights Enumeration

Member name	Value	Description
All	-1	User has all rights.
None	0	User has no rights.
EditSecurity	1	User can configure security.
SysConfig	2	User can change system configuration.
EditPrograms	4	User can edit programs.
EditPoints	8	User can edit points.
EditVision	16	User can change vision properties.
JogAndTeach	32	User can Jog & Teach.
CommandWindow	64	User can use the command window.
EditRobotParameters	128	User can edit robot parameters.
ConfigureOptions	256	User can configure options.
ViewAudit	512	User can view the security audit log.
EditProject	1024	User can edit the project configuration.
DeleteAudit	2048	User can delete security audit log entries.
TeachPoints	4096	User can teach points.
ChangeOutputs	8192	User can change output status.
ChangeMemIO	16384	User can change memory I/O status.
EditGUIBuilder	32768	User can make changes in GUI Builder.
EditForce Control.	65536	User can make changes in Force Guide and Force Control.
EditPartFeeding	131072	User can make changes in Part Feeding.

#### 14.16.30 SpelVDefShowWarning Enumeration

Member name	Value	Description
None	-1	Do not display a warning.
Always	0	Always display a warning.
DependsOnSpeed	1	Display when either RobotSpeed or RobotAccel is larger than 5.

## 14.16.31 SpelVisionImageSize Enumeration

Member name	Value	Description
Size320x240	1	320 x 240 image size.
Size640x480	2	640 x 480 image size.
Size800x600	3	800 x 600 image size.
Size1024x768	4	1024 x 768 image size.
Size1280x1024	5	1280 x 1024 image size.
Size1600x1200	6	1600 x 1200 image size.
Size2048x1536	7	2048 x 1536 image size.
Size2560x1920	8	2560 x 1920 image size.
Size3664x2748	9	3664 x 2748 image size.
Size5472x3648	10	5472 x 3648 image size.
Size4024x3036	11	4024 x 3036 image size.

## 14.16.32 SpelVisionObjectTypes Enumeration

Member name	Value	Description
Correlation	1	Correlation object.
Blob	2	Blob object.
Edge	3	Edge object.
Polar	4	Polar object.
Line	5	Line object.
Point	6	Point object.
Frame	7	Frame object.
ImageOp	8	ImageOp object.
OCR	9	OCR object.
CodeReader	10	CodeReader object.
Geometric	11	Geometric object.
ColorMatch	14	Color Match object.
LineFinder	15	Line Finder object.
ArcFinder	16	Arc Finder object.
DefectFinder	17	Defect Finder object.
LineInspector	18	Line Inspector object.
ArcInspector	19	Arc Inspector object.
BoxFinder	20	Box Finder object
CornerFinder	21	Corner Finder object
Contour	22	Contour object
Text	23	Text object
Decision	26	Decision object
Coordinates	27	Coordinates object

### 14.16.33 SpelVisionProps Enumeration

This enumeration is for all vision properties and results. Refer to the Vision Guide Reference manual for details.

### 14.16.34 SpelWindows Enumeration

Member name	Value	Description
IOMonitor	1	ID for the I/O Monitor window.
TaskManager	2	ID for the Task Manager window.
ForceMonitor	3	ID for the Force Monitor window.
Simulator	4	ID for the Simulator window.

### 14.16.35 SpelWrist Enumeration

Member name	Value	Description
NoFlip	1	Wrist orientation is no flip.
Flip	2	Wrist orientation is flip.

## 14.17 Spel Error Numbers and Messages

For error numbers and error messages, see the Status Code / Error Code List.

## 15. 32 Bit and 64 Bit Applications

The RCAPINet library provided in EPSON RC+ 7.0 version 7.1.0 and greater automatically supports 32 bit and 64 bit applications.

In versions of EPSON RC+ 7.0 prior to 7.1.0, separate libraries were supplied for 32 bit and 64 bit support. These obsolete libraries (SpelNetLib70.dll and SpelNetLib70\_x64.dll) are no longer provided when RC+ 7.5.0 or greater is installed. For details on using the obsolete libraries, see the RC+ API manual for the previous version of EPSON RC+ 7.0 that you were using.

## 16. Using the LabVIEW VI Library

### 16.1 Overview

In versions of EPSON RC+ 7.0 prior to v7.1.0, the API .NET library could be used directly in LabVIEW. In EPSON RC+ 7.0 v7.1.0, a new LabVIEW VI library was introduced. The new library has the following features:

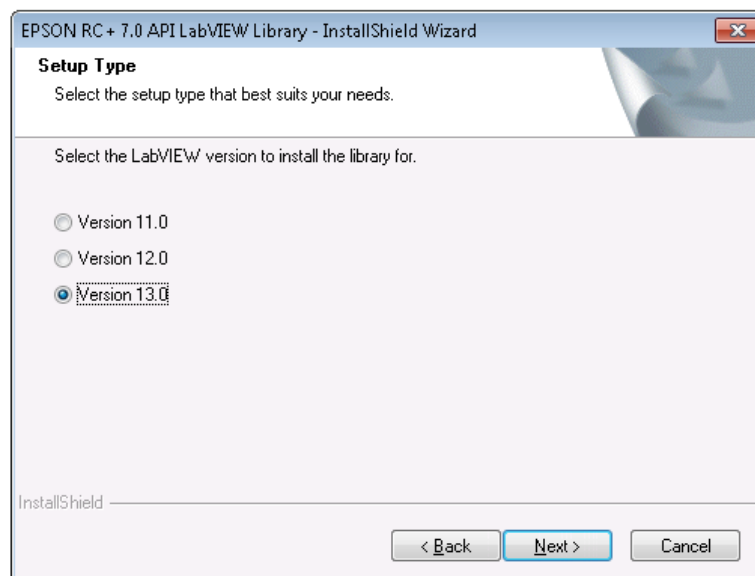
- High level interface to EPSON RC+ 7.0 using VIs (Virtual Instruments).
- The user no longer needs to deal with the .NET interface to EPSON RC+ 7.0 – it is handles automatically.
- Each Spel command is wrapped in an individual VI.
- The VIs are organized in several Tool Palettes.
- Supports both 32 bit and 64 bit LabVIEW applications.
- Supports LabVIEW versions 2009 and greater.

To use the LabVIEW VI library, you must purchase an EPSON RC+ 7.0 API software license for each Controller that you connect with.

### 16.2 Installation

To use the EPSON RC+ 7.0 LabVIEW VI Library, you must install it using the installer provided in the \EpsonRC70\API\LabVIEW folder on your PC.

1. Install LabVIEW version 2009 or greater.
2. Navigate to the \EpsonRC70\API\LabVIEW folder on your PC and run the EpsonRC70\_vxxx\_LabVIEW.exe installer, where xxx is the version number for EPSON RC+ 7.0. For example, EpsonRC70\_v710\_LabVIEW.exe.
3. When the installer starts, it will display the detected versions of LabVIEW installed on your PC. The latest version is selected by default. Select the version that will use the EPSON RC+ 7.0 LabVIEW VI Library.



4. Click Next, then click Install. The VIs, Controls, and palettes will be installed for the selected version of LabVIEW.

### 16.3 Tool and Control Palettes

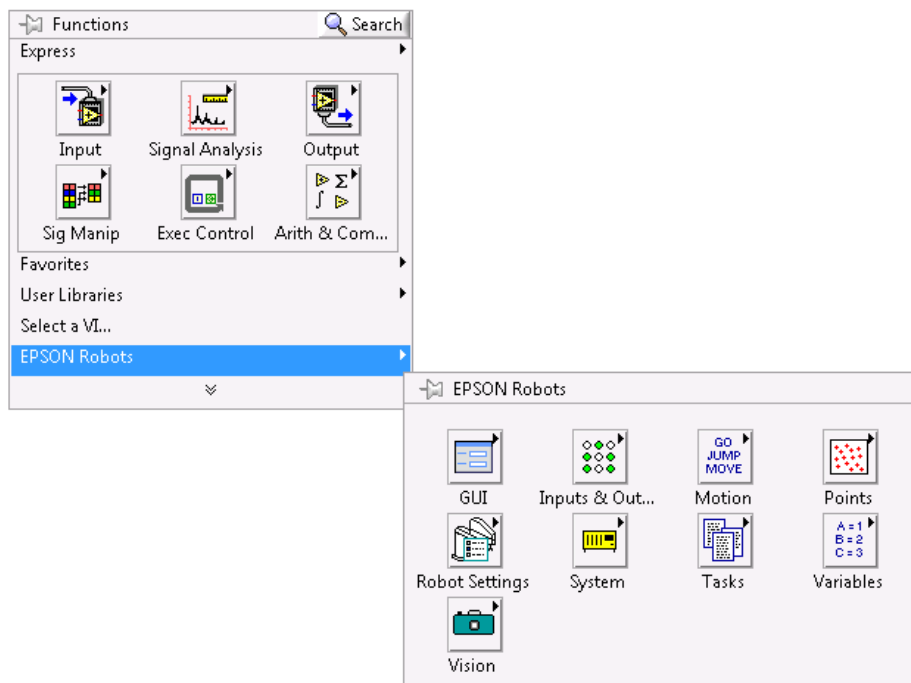
After the EPSON RC+ 7.0 LabVIEW VI Library is installed, you can access the VIs and controls available in the library from the [EPSON Robots] tool palette and [EPSON Robots] control palette.

#### Tool Palette

The tool palette is accessed from the block diagram. Inside the Epson Robots tool palette are several sub-palettes, described in the following table:

Palette	Description
System	Used to initialize and shutdown the API.
Robot Settings	Change robot parameters.
Points	Load, save, change robot points.
Motion	Execute robot motion.
Inputs & Outputs	Control and monitor Controller inputs and outputs.
Tasks	Manage tasks in the Robot Controller.
Variables	Read and write variables in the Controller.
Vision	Execute vision commands.
GUI	Display GUI functions.

To access the [Epson Robots] tool palette, open the block diagram for your VI, then right click on an empty area and select [Epson Robots] to see the sub-palettes described above.

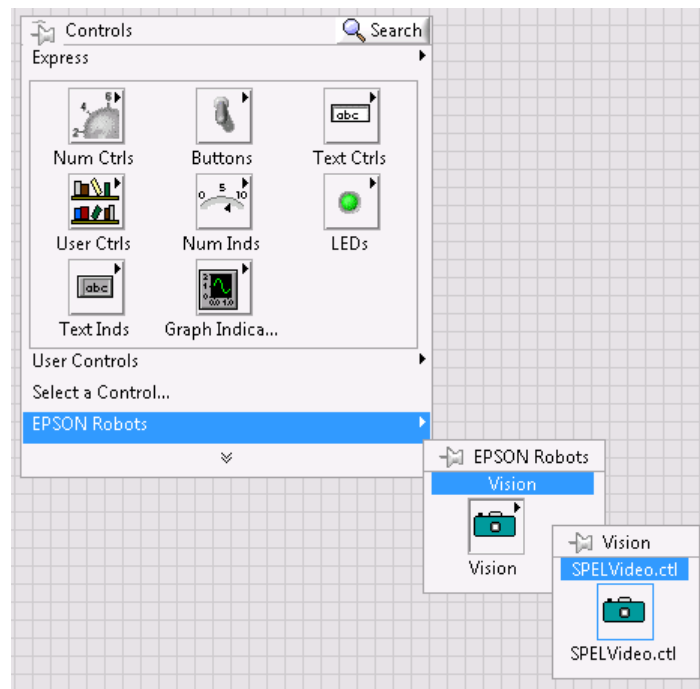




## Control Palette

The control palette is accessed from the front panel.

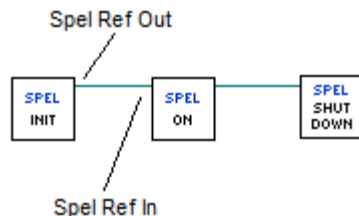
Palette	Description
Vision	Contains the SPELVideo control used to display video.



## 16.4 Getting started

To use the LabVIEW VI library, your application must first call the Spel Initialize VI for each Controller you want to use. The Initialize VI starts an EPSON RC+ 7.0 server process that will connect to the Robot Controller and process the subsequent Spel command VIs. The Initialize VI has a Spel Ref Out output. This must be wired to the next Spel VI Spel Ref In input. Then for each subsequent VI, the Spel Ref Out output from a previous Spel VI must be wired to the Spel Ref In input of the next Spel VI.

For example, the flow diagram below shows the wires for Spel Ref Out and Spel Ref In between each Spel node.



When the application is shutting down, you must call the Spel Shutdown VI. This will disconnect from the Robot Controller and shutdown the associated EPSON RC+ 7.0 server process.

Follow these steps to get started. First, you will create two safe robot points from within the EPSON RC+ 7.0 GUI for the LabVIEW default Spel+ project. Then you will build a small application in LabVIEW to move the robot between the two points.

1. Ensure that the EPSON RC+ 7.0 and the EPSON RC+ 7.0 LabVIEW VI Library are installed on your PC. See section 16.2 for details for installing the LabVIEW VI Library.
2. Start EPSON RC+ 7.0.
3. From the Project menu, select Open, then navigate to the LabVIEW folder and select the LabVIEW\_Default project. Click Open.
4. From the Tools menu, select Robot Manager. Click the Motor On button.
5. Select the Jog & Teach page on the Robot Manager. Jog the robot to some safe position.
6. Click Teach to teach point 0.
7. Jog the robot to another safe position.
8. Select P1 from the Point list, then click Teach to teach point 1.
9. Click the Save button on the main toolbar to save the points.
10. Close EPSONRC+ 7.0.
11. Start LabVIEW and create a new VI.
12. Open the block diagram for the new VI.
13. From the Epson RC+ API | System tool palette, drag the Init VI onto the block diagram. The Initialize VI is required for each Controller that you interface with.
14. From the Epson RC+ API | Robot Settings tool palette, drag the MotorOn VI onto the block diagram. Wire the Spel Ref Out output from the Initialize VI to the Spel Ref In input on the MotorOn VI.
15. From the Epson RC+ API | Motion tool palette, drag the Go VI onto the block diagram. Wire the Spel Ref Out output from the MotorOn VI to the Spel Ref In input on the Go VI. Add a constant for the Point Number input and set the value to 0.

16. From the Epson RC+ API | Motion tool palette, drag another Go VI onto the block diagram. Wire the Spel Ref Out output from the previous Go VI to the Spel Ref In input on the second Go VI. Add a constant for the Point Number input and set the value to 1.
17. From the Epson RC+ API | Robot Settings tool palette, drag the MotorOff VI onto the block diagram. Wire the Spel Ref Out output from the Go VI to the Spel Ref In input on the MotorOff VI.
18. From the Epson RC+ API | System tool palette, drag the Shutdown VI onto the block diagram. The Shutdown VI must be used for each Init VI.  
The block diagram should look similar to this:



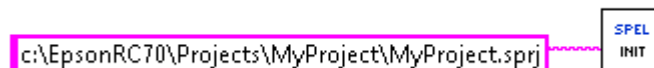
19. Run the application. The robot motors should turn on, then the robot should move to point 0, then move to point 1, and then the robot motors will turn off.

## 16.5 Working with Spel+ projects

Using a Spel+ project with your LabVIEW application is optional. However, if you will be saving point data, or you want to use point labels and / or I/O labels, tasks, or vision sequences, then you will need to use a Spel+ project.

By default, the project is LabVIEW\_Default, located in the \EpsonRC70\Projects\LabVIEW folder.

If desired, you can create your own projects using EPSON RC+ 7.0, and then specify which project you want to use with the Initialize VI *Project* input parameter, as shown below:



To work with EPSON RC+ 7.0 projects, start the EPSON RC+ 7.0 application. Use the Project menu to create, open, and edit projects. For more information, see the EPSON RC+ 7.0 User's Guide.

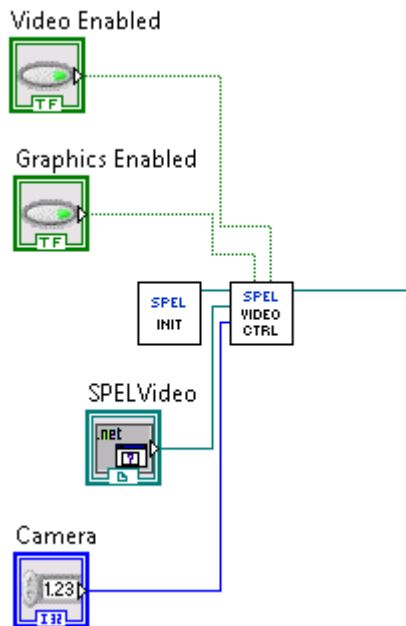
## 16.6 Displaying Video

You can display video for your Vision Guide sequences using the SPELVideo control and the VideoControl VI.

To display video:

1. Add a SPELVideo control to a front panel.
2. Add a VideoControl.vi to the corresponding block diagram.
3. Wire the output from the SPELVideo control to the SPELVideo Ref In input on the VideoControl VI.
4. Wire the Spel Ref In and Spel Ref Out parameters for the VideoControl VI.
5. Add constants or controls for the *Camera*, *Graphics Enabled*, and *Video Enabled* parameters on the VideoControl VI. *Video Enabled* must be set to true in order to display video.

The flow diagram below shows the connections for the SPELVideo control and the SPEL VideoControl VI.



When *Video Enabled* is true, and VRun executes from the VRun VI or in a Controller task, you will see the resulting video, depending on the Camera setting.

By default, the *Camera* input parameter is zero, which allows video from any camera to be displayed. If you set *Camera* to a number other than zero, then video will be displayed for sequences using the specified camera.

When *Graphics Enabled* is true, and VRun executes, then the sequence result graphics are displayed over the video image.

You can only use one SPEL Video control at a time in your application.

## 16.7 VI Reference

This section contains information for all VIs used in the EPSON RC+ 7.0 LabVIEW VI Library.

The following information is provided for each VI:

<b>Tool Palette</b>	The tool palette where the VI is contained.
<b>Description</b>	Brief description of the function.
<b>Inputs</b>	Input parameters
<b>Outputs</b>	Output parameters
<b>Remarks</b>	Additional details.

## Accel VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Sets the point to point acceleration and deceleration for the current robot.

**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Accel</i>	Integer value for point to point acceleration.
<i>Decel</i>	Integer value for point to point acceleration.
<i>Depart Accel</i>	Optional. Integer value for Jump depart acceleration.
<i>Depart Decel</i>	Optional. Integer value for Jump depart deceleration.
<i>Appro Accel</i>	Optional. Integer value for Jump approach acceleration.
<i>Appro Decel</i>	Optional. Integer value for Jump approach deceleration.

**Outputs**

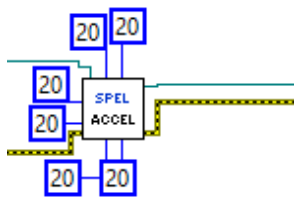
<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**Remarks**

Use Accel to set the point to point acceleration and deceleration values for the current robot. All values can be from 1 to 100%. If Depart Accel is specified, then the remaining inputs must also be specified.

**See Also**

AccelS, Speed, SpeedS

**Accel Example**

## AccelS VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Sets the linear acceleration and deceleration for the current robot.

**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Accel</i>	Double value for linear acceleration.
<i>Decel</i>	Double value for linear acceleration.
<i>Depart Accel</i>	Optional. Double value for Jump depart acceleration.
<i>Depart Decel</i>	Optional. Double value for Jump depart deceleration.
<i>Appro Accel</i>	Optional. Double value for Jump approach acceleration.
<i>Appro Decel</i>	Optional. Double value for Jump approach deceleration.

**Outputs**

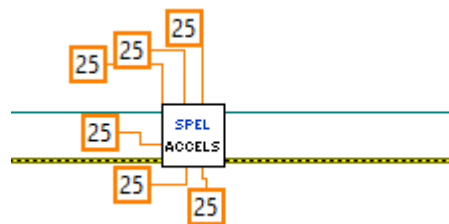
<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**Remarks**

Use AccelS to set the linear acceleration and deceleration values for the current robot. All values are in millimeters / sec<sup>2</sup>. If Depart Accel is specified, then the remaining inputs must also be specified.

**See Also**

Accel, Speed, SpeedS

**AccelS Example**

## Arc VI

**Tool Palette**

Epson Robots | Motion

**Description**

Arc moves the arm to the specified point using circular interpolation in the XY plane.

**Inputs**

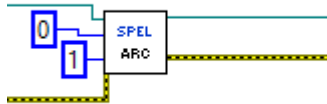
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Mid Point Number</i>	Specifies the mid point by using an integer.
<i>Mid Point Expr</i>	Specifies the mid point by using a string expression. If this input is used, then you must also specify the end point with a string expression.
<i>End Point Number</i>	Specifies the end point by using an integer.
<i>End Point Expr</i>	Specifies the end point by using a string expression. You can include ROT, CP, SYNC, a search expression for Till, and a parallel processing statement.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

Accel, Arc3, BMove, Go, Jump, Jump3, Move, Speed, TGo, TMove

**Arc Example**



## Arc3 VI

**Tool Palette**

Epson Robots | Motion

**Description**

Arc3 moves the arm to the specified point using circular interpolation in 3 dimensions.

**Inputs**

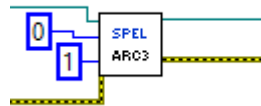
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Mid Point Number</i>	Specifies the mid point by using an integer.
<i>Mid Point Expr</i>	Specifies the mid point by using a string expression. If this input is used, then you must also specify the end point with a string expression.
<i>End Point Number</i>	Specifies the end point by using an integer.
<i>End Point Expr</i>	Specifies the end point by using a string expression. You can include CP, SYNC, a search expression for Till, and a parallel processing statement.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

AccelS, Arc, BMove, Go, Jump, Jump3, Move, SpeedS, TGo, TMove

**Arc3 Example**

## Arch VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Defines ARCH parameters (Z height to move before beginning horizontal motion) for use with the JUMP instructions.

**Inputs**

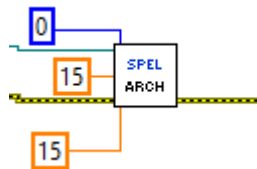
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Arch Number</i>	The depart distance in millimeters moved at the beginning of the Jump instruction before starting horizontal motion.
<i>Depart Dist</i>	The depart distance in millimeters moved at the beginning of the Jump instruction before starting horizontal motion.
<i>Appro Dist</i>	The approach distance in millimeters above the target position of the Jump instruction.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

Jump, Jump3

**Arch Example**

## Arm VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Selects the current robot arm.

**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Arm Number</i>	Integer from 0-15. The user may select up to 16 different arms. Arm 0 is the standard (default) robot arm. Arm(s) 1-15 are auxiliary arms defined by the ArmSet instruction.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

Armset, GetArm, Tool

**Arm Example**

## Armset VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

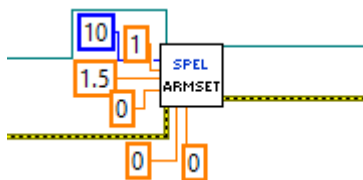
Defines an auxiliary robot arm.

**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>ArmNumber</i>	Integer number: Valid range from 1-15.
<i>Param1</i>	(For SCARA Robots) The horizontal distance from the center line of the elbow joint to the center line of the new orientation axis. (I.E. the position where the new auxiliary arm's orientation axis center line is located.) (For Cartesian Robots) X axis direction position offset from the original X position specified in mm.
<i>Param2</i>	(For SCARA Robots) The offset (in degrees) between the line formed between the normal Elbow center line and the normal orientation Axis center line and the line formed between the new auxiliary arm elbow center line and the new orientation axis center line. (These 2 lines should intersect at the elbow center line and the angle formed is the <i>Param2</i> .) (For Cartesian Robots) Y axis direction position offset from the original Y position specified in mm.
<i>Param3</i>	(For SCARA & Cartesian Robots) The Z height offset difference between the new orientation axis center and the old orientation axis center. (This is a distance.)
<i>Param4</i>	(For SCARA Robots) The distance from the shoulder center line to the elbow center line of the elbow orientation of the new auxiliary axis. (For Cartesian Robots) This is a dummy parameter (Specify 0)
<i>Param5</i>	(For SCARA & Cartesian Robots) The angular offset (in degrees) for the new orientation axis vs. the old orientation axis.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**ArmSet Example**

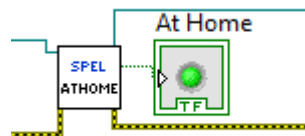
## AtHome VI

**Tool Palette**

Epson Robots | Motion

**Description**

Returns True if the current robot is at the home position.

**Inputs***Spel Ref In* Spel reference from a previous Spel Ref Out.*Error In* Error condition from a previous Spel node.**Outputs***Spel Ref Out* Spel reference output for next VI to use.*Error Out* Error condition output for subsequent Spel nodes.*At Home* Boolean indicating if the current robot is at the home position.**AtHome Example**

AvoidSing VI

**Tool Palette**

Epson Robots | Motion

**Description**

Enables / disables the singularity avoidance feature for Move, Arc, and Arc3 motion methods.

**Inputs**

*Spel Ref In* Spel reference from a previous Spel Ref Out.

*Error In* Error condition from a previous Spel node.

*Enable* True enables singularity avoidance and False disables it.

**Outputs**

*Spel Ref Out* Spel reference output for next VI to use.

*Error Out* Error condition output for subsequent Spel nodes.

**AvoidSing Example**



## BGo VI

**Tool Palette**

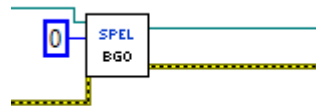
Epson Robots | Motion

**Description**

Executes Point to Point relative motion in the selected local coordinate system.

**Inputs***Spel Ref In* Spel reference from a previous Spel Ref Out.*Error In* Error condition from a previous Spel node.*Point Number* Optional. Specifies the target end point by using the point number for a previously taught point in the Controller's point memory for the current robot. If *Point Expression* is specified, then *Point Number* is ignored.*Point Expression* Optional. Specifies the target end point by using a string expression. If *Point Expression* is not specified, then the *Point Number* input will be used.**Outputs***Spel Ref Out* Spel reference output for next VI to use.*Error Out* Error condition output for subsequent Spel nodes.**See Also**

Accel, Arc, Arc3, BMove, Go, Jump, Jump3, Move, Speed, TGo, TMove

**BGo Example**

## BMove VI

**Tool Palette**

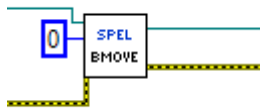
Epson Robots | Motion

**Description**

Executes linear interpolated relative motion in the selected local coordinate system

**Inputs***Spel Ref In* Spel reference from a previous Spel Ref Out.*Error In* Error condition from a previous Spel node.*Point Number* Optional. Specifies the target end point by using the point number for a previously taught point in the Controller's point memory for the current robot. If *Point Expression* is specified, then *Point Number* is ignored.*Point Expression* Optional. Specifies the target end point by using a string expression. If *Point Expression* is not specified, then the *Point Number* input will be used.**Outputs***Spel Ref Out* Spel reference output for next VI to use.*Error Out* Error condition output for subsequent Spel nodes.**See Also**

AccelS, Arc, Arc3, BGo, Go, Jump, Jump3, Move, SpeedS, TGo, TMove

**BMove Example**



## Box VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Specifies an approach check area defined within a box.

**Inputs**

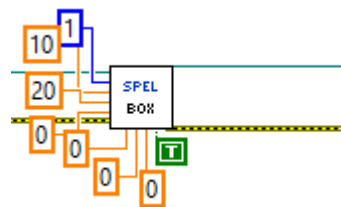
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>AreaNumber</i>	Integer number from 1-15 representing which of the 15 boxes to define.
<i>Min X</i>	The minimum X coordinate position of the approach check area.
<i>Max X</i>	The maximum X coordinate position of the approach check area.
<i>Min Y</i>	The minimum Y coordinate position of the approach check area.
<i>Max Y</i>	The maximum Y coordinate position of the approach check area.
<i>Min Z</i>	The minimum Z coordinate position of the approach check area.
<i>Max Z</i>	The maximum Z coordinate position of the approach check area.
<i>Polarity On</i>	Set the remote output logic when the corresponding remote output is used. To set I/O output to On when the end effector is in the box area, use True. To set I/O output to Off when the end effector is in the box area, use False.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

XYLim

**Box Example**

Continue VI

**Tool Palette**

Epson Robots | Tasks

**Description**

Causes all tasks in the Controller that were paused to resume.

**Inputs**

*Spel Ref In* Spel reference from a previous Spel Ref Out.

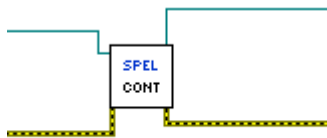
*Error In* Error condition from a previous Spel node.

**Outputs**

*Spel Ref Out* Spel reference output for next VI to use.

*Error Out* Error condition output for subsequent Spel nodes.

**Continue Example**



## Delay VI

**Tool Palette**

Epson Robots | System

**Description**

Delays processing for the specified number of milliseconds.

**Inputs***Spel Ref In* Spel reference from a previous Spel Ref Out.*Error In* Error condition from a previous Spel node.*Milliseconds* The number of milliseconds to delay.**Outputs***Spel Ref Out* Spel reference output for next VI to use.*Error Out* Error condition output for subsequent Spel nodes.**Delay Example**

ECP VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Selects the current ECP definition.

**Inputs**

- Spel Ref In* Spel reference from a previous Spel Ref Out.
- Error In* Error condition from a previous Spel node.
- ECPNumber* Integer number from 0-15 representing which of 16 ECP definitions to use with the next motion instructions.

**Outputs**

- Spel Ref Out* Spel reference output for next VI to use.
- Error Out* Error condition output for subsequent Spel nodes.

**See Also**

ECPSet, GetECP

**ECP Example**



## ECPset VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Defines an ECP (external control point).

**Inputs**

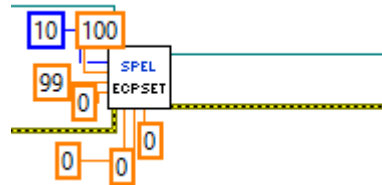
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>ECPNumber</i>	Integer number from 1-15 representing which of 15 external control points to define.
<i>X</i>	The external control point X coordinate.
<i>Y</i>	The external control point Y coordinate.
<i>Z</i>	The external control point Z coordinate.
<i>U</i>	The external control point U coordinate.
<i>V</i>	The external control point V coordinate.
<i>W</i>	The external control point W coordinate.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

ECP, GetECP

**ECPSet Example**

EStopOn VI

**Tool Palette**

Epson Robots | System

**Description**

Returns the Emergency Stop status.

**Inputs**

*Spel Ref In* Spel reference from a previous Spel Ref Out.

*Error In* Error condition from a previous Spel node.

**Outputs**

*Spel Ref Out* Spel reference output for next VI to use.

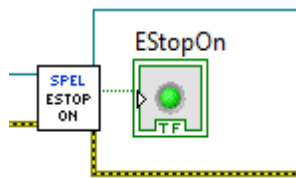
*Error Out* Error condition output for subsequent Spel nodes.

*EStopOn* *True if the status EmergencyStop, otherwise is False*

**See Also**

SafetyOn

**EStopOn Example**



## Find VI

**Tool Palette**

Epson Robots | Motion

**Description**

Specifies a condition for storing coordinates during motion.

**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Condition</i>	String expression that contains functions and operators .

**Outputs**

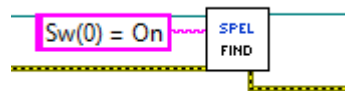
<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**Remarks**

Use Find to specify when a position should be stored during motion. When the condition is satisfied, the current position is stored in FindPos.

**See Also**

FindPos

**Find Example**

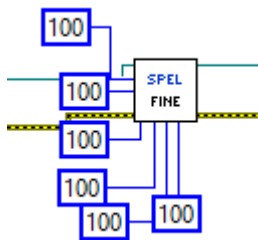
## Fine VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Specifies and displays the positioning accuracy for target points.

**Inputs***Spel Ref In* Spel reference from a previous Spel Ref Out.*Error In* Error condition from a previous Spel node.*J1MaxErr – J9MaxErr* Integer number ranging from (0-32767) which represents the allowable positioning error for the each joint. The values for joints 7, 8, and 9 are optional.**Outputs***Spel Ref Out* Spel reference output for next VI to use.*Error Out* Error condition output for subsequent Spel nodes.**Fine Example**



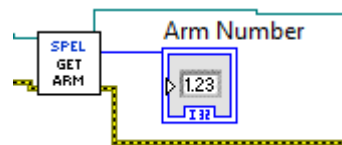
## GetArm VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Returns the current Arm number for the current robot.

**Inputs***Spel Ref In* Spel reference from a previous Spel Ref Out.*Error In* Error condition from a previous Spel node.**Outputs***Spel Ref Out* Spel reference output for next VI to use.*Error Out* Error condition output for subsequent Spel nodes.*Arm Number* The current arm number.**GetArm Example**

## GetAvoidSing VI

### Tool Palette

Epson Robots | Motion

### Description

#### Returns the state of AvoidSingularity.Inputs

*Spel Ref In* Spel reference from a previous Spel Ref Out.

*Error In* Error condition from a previous Spel node.

### Outputs

*Spel Ref Out* Spel reference output for next VI to use.

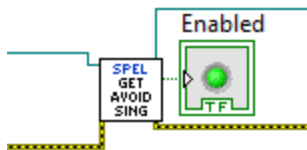
*Error Out* Error condition output for subsequent Spel nodes.

*Enabled* Returns the state of AvoidSingularity.

### See Also

AvoidSing

### GetAvoidSing Example



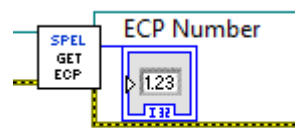
## GetECP VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Returns the current ECP number for the current robot.

**Inputs***Spel Ref In* Spel reference from a previous Spel Ref Out.*Error In* Error condition from a previous Spel node.**Outputs***Spel Ref Out* Spel reference output for next VI to use.*Error Out* Error condition output for subsequent Spel nodes.*ECP Number* The current ECP number.**GetECP Example**

GetLimZ VI

**Tool Palette**

Epson Robots | Motion

**Description**

Returns the current LimZ setting.

**Inputs**

*Spel Ref In* Spel reference from a previous Spel Ref Out.

*Error In* Error condition from a previous Spel node.

**Outputs**

*Spel Ref Out* Spel reference output for next VI to use.

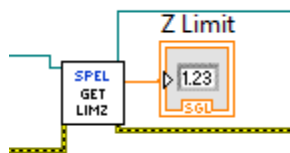
*Error Out* Error condition output for subsequent Spel nodes.

*Z Limit* The current LimZ setting.

**See Also**

LimZ

**GetLimZ Example**



## GetMotor VI

**Tool Palette**

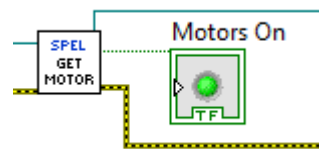
Epson Robots | Robot Settings

**Description**

Returns the motor on status for the current robot.

**Inputs***Spel Ref In* Spel reference from a previous Spel Ref Out.*Error In* Error condition from a previous Spel node.**Outputs***Spel Ref Out* Spel reference output for next VI to use.*Error Out* Error condition output for subsequent Spel nodes.*Motors On* True if motors are on and false if not.**See Also**

GetPower, MotorOn, MotorOff

**GetMotor Example**

GetOprMode VI

**Tool Palette**

Epson Robots | System

**Description**

Reads the EPSON RC+ 7.0 mode of operation.

**Inputs**

*Spel Ref In* Spel reference from a previous Spel Ref Out.

*Error In* Error condition from a previous Spel node.

**Outputs**

*Spel Ref Out* Spel reference output for next VI to use.

*Error Out* Error condition output for subsequent Spel nodes.

*Operation Mode* The mode of operation for the associated EPSON RC+ 7.0 server process.

Mode	ID	Description
Auto	1	EPSON RC+ 7.0 is in auto mode.
Program	2	EPSON RC+ 7.0 is in program mode..

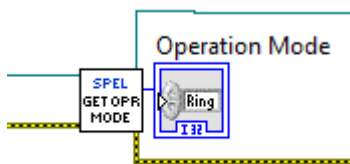
**Remarks**

When *Operation Mode* is set to Program, the EPSON RC+ 7.0 GUI for the associated server process is opened and the Controller operation mode is set to Program. If the user closes the RC+ GUI, *Operation Mode* is set to Auto. If *Operation Mode* is set to Auto, the RC+ GUI also closes.

**See Also**

OprMode

**GetOprMode Example**



## GetPoint VI

**Tool Palette**

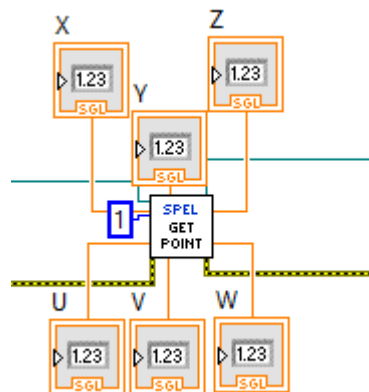
Epson Robots | Points

**Description**

Retrieves coordinate data for a robot point.

**Inputs***Spel Ref In* Spel reference from a previous Spel Ref Out.*Error In* Error condition from a previous Spel node.*Point Number* Optional. Specifies the target end point by using the point number for a previously taught point in the Controller's point memory for the current robot. If *Point Expression* is specified, then *Point Number* is ignored.*Point Expression* Optional. Specifies the target end point by using a string expression. If *Point Expression* is not specified, then the *Point Number* input will be used.**Outputs***Spel Ref Out* Spel reference output for next VI to use.*Error Out* Error condition output for subsequent Spel nodes.*X – W* X, Y, Z, U, V, W coordinates of the specified point.**See Also**

LoadPoints, Robot, SavePoints, SetPoint

**GetPoint Example**

GetPower VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Returns the power high status for the current robot.

**Inputs**

*Spel Ref In*      Spel reference from a previous Spel Ref Out.

*Error In*        Error condition from a previous Spel node.

**Outputs**

*Spel Ref Out*    Spel reference output for next VI to use.

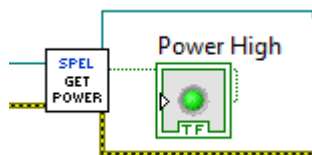
*Error Out*        Error condition output for subsequent Spel nodes.

*Power High*      True if power is high and false if not.

**See Also**

PowerHigh, PowerLow

**GetPower Example**





## GetRobot VI

**Tool Palette**

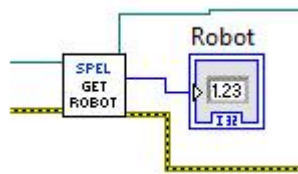
Epson Robots | Robot Settings

**Description**

Returns the current robot number.

**Inputs***Spel Ref In* Spel reference from a previous Spel Ref Out.*Error In* Error condition from a previous Spel node.**Outputs***Spel Ref Out* Spel reference output for next VI to use.*Error Out* Error condition output for subsequent Spel nodes.*Robot Number* The current robot number.**See Also**

Robot

**GetRobot Example**

GetTool VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Returns the current Tool number for the current robot.

**Inputs**

*Spel Ref In*      Spel reference from a previous Spel Ref Out.

*Error In*        Error condition from a previous Spel node.

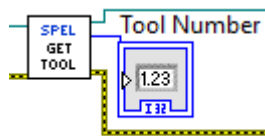
**Outputs**

*Spel Ref Out*    Spel reference output for next VI to use.

*Error Out*        Error condition output for subsequent Spel nodes.

*Tool Number*    The current tool number.

**GetTool Example**



## GetVar VI

**Tool Palette**

Epson Robots | Variables

**Description**Returns the value of a SPEL<sup>+</sup> global preserve variable in the Controller.**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Var Name</i>	The name of the SPEL <sup>+</sup> global preserve variable. For an array, the entire array can be returned or just one element.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.
<i>Value</i>	A variant containing the value.

**See Also**

SetVar

**GetVar Example**

## Go VI

**Tool Palette**

Epson Robots | Motion

**Description**

Moves the arm in a Point to Point fashion from the current position to the specified point or XY position. The **GO** instruction can move any combination of the robot axes at the same time.

**Inputs**

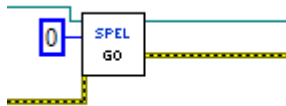
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Point Number</i>	Optional. Specifies the target end point by using the point number for a previously taught point in the Controller's point memory for the current robot. If <i>Point Expression</i> is specified, then <i>Point Number</i> is ignored.
<i>Point Expression</i>	Optional. Specifies the target end point by using a string expression. If <i>Point Expression</i> is not specified, then the <i>Point Number</i> input will be used.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

Accel, Arc, Arc3, BGo, BMove, Jump, Jump3, Move, Speed, TGo, TMove

**Go Example**

## Halt VI

**Tool Palette**

Epson Robots | Tasks

**Description**

Suspends execution of the specified task.

**Inputs**

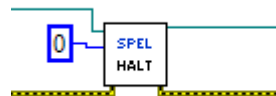
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Task Number</i>	Optional. The task number of the task to be suspended. The range of the task number is 1 to 32. If <i>Task Name</i> is specified, then <i>Task Number</i> is ignored.
<i>Task Name</i>	Optional. Specifies the name of the task to be suspended. If <i>Task Name</i> is not specified, then the <i>Task Number</i> input will be used.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

Quit, Resume

**Halt Example**

Here VI

**Tool Palette**

Epson Robots | Points

**Description**

Teaches a point at the current position.

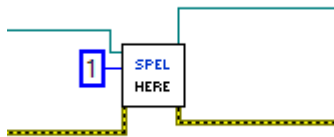
**Inputs**

- Spel Ref In*      Spel reference from a previous Spel Ref Out.
- Error In*        Error condition from a previous Spel node.
- Point Number*    Optional. Specifies the target end point by using the point number for a previously taught point in the Controller's point memory for the current robot. If *Point Expression* is specified, then *Point Number* is ignored.
- Point Name*      Optional. Specifies the name of the point. If *Point Name* is not specified, then the *Point Number* input will be used.

**Outputs**

- Spel Ref Out*     Spel reference output for next VI to use.
- Error Out*        Error condition output for subsequent Spel nodes.

**Here Example**



## HideWindow VI

**Tool Palette**

Epson Robots | GUI

**Description**

Hides an EPSON RC+ 7.0 window that was previously displayed with ShowWindow.

**Inputs***Spel Ref In* Spel reference from a previous Spel Ref Out.*Error In* Error condition from a previous Spel node.*Window ID* The ID of the EPSON RC+ 7.0 window to show.**Window name ID Description**

IOMonitor	1	ID for the I/O Monitor window.
TaskManager	2	ID for the Task Manager window.
ForceMonitor	3	ID for the Force Monitor window.
Simulator	4	ID for the Simulator window.

**Outputs***Spel Ref Out* Spel reference output for next VI to use.*Error Out* Error condition output for subsequent Spel nodes.**See Also**

RunDialog, ShowWindow

**HideWindow Example**

**Tool Palette**

Epson Robots | Inputs &amp; Outputs

**Description**

Returns the status of the specified input port. Each port contains 8 input bits (one byte).

**Inputs**

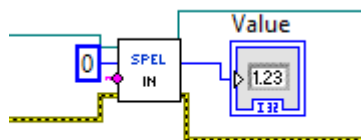
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Port Number</i>	Optional. Integer expressing one of the input ports. Each port contains 8 input bits (one byte). If <i>Label</i> is not specified, then <i>Port Number</i> is used.
<i>Label</i>	Optional. String containing an input byte label. If <i>Label</i> is specified, then <i>Port Number</i> is ignored.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.
<i>Value</i>	Integer from 0 to 255 representing the status of the input port.

**See Also**

InBCD, InW, Sw

**In Example**



## InBCD VI

**Tool Palette**

Epson Robots | Inputs &amp; Outputs

**Description**

Returns the input status of 8 inputs using BCD format. (Binary Coded Decimal)

**Inputs**

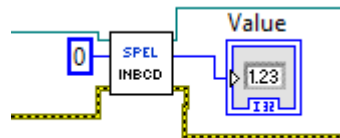
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Port Number</i>	Optional. Integer expressing one of the input ports. Each port contains 8 input bits (one byte). If <i>Label</i> is not specified, then <i>Port Number</i> is used.
<i>Label</i>	Optional. String containing an input byte label. If <i>Label</i> is specified, then <i>Port Number</i> is ignored.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.
<i>Value</i>	Integer from 0 to 9 representing the status of the input port.

**See Also**

In, InW, Sw

**InBCD Example**

InsideBox VI

**Tool Palette**

Epson Robots | Motion

**Description**

Returns whether the current robot end effector is inside a specified box area.

**Inputs**

- Spel Ref In* Spel reference from a previous Spel Ref Out.
- Error In* Error condition from a previous Spel node.
- Area Number* Integer number from 1-15 representing which of the 15 boxes to check.

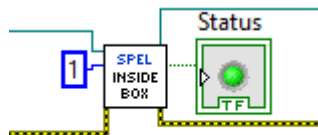
**Outputs**

- Spel Ref Out* Spel reference output for next VI to use.
- Error Out* Error condition output for subsequent Spel nodes.
- Status* Boolean that is True if the robot end effector is inside the box.

**See Also**

Box, InsidePlane, Plane

**InsideBox Example**



## InsidePlane VI

**Tool Palette**

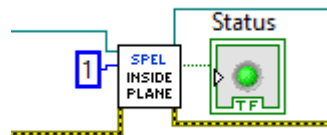
Epson Robots | Motion

**Description**

Returns whether the current robot end effector is inside a specified plane.

**Inputs***Spel Ref In* Spel reference from a previous Spel Ref Out.*Error In* Error condition from a previous Spel node.*AreaNumber* Integer number from 1-15 representing which of the 15 boxes to check.**Outputs***Spel Ref Out* Spel reference output for next VI to use.*Error Out* Error condition output for subsequent Spel nodes.*Status* Boolean that is True if the robot end effector is inside the plane.**See Also**

Box, InsideBox, Plane

**InsidePlane Example**

## InW VI

**Tool Palette**

Epson Robots | Inputs &amp; Outputs

**Description**

Returns the status of the specified input word port. Each word port contains 16 input bits.

**Inputs**

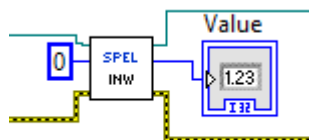
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Port Number</i>	Optional. Integer expressing one of the input ports. Each port contains 8 input bits (one byte). If <i>Label</i> is not specified, then <i>Port Number</i> is used.
<i>Label</i>	Optional. String containing an input byte label. If <i>Label</i> is specified, then <i>Port Number</i> is ignored.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.
<i>Value</i>	Integer value from 0 to 65535 representing the input port.

**See Also**

In, InBCD, Sw

**InW Example**

## Inertia VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Specifies the load inertia and eccentricity for the current robot.

**Inputs**

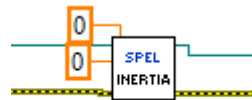
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>LoadInertia</i>	Double value that specifies total moment of inertia in kgm <sup>2</sup> around the center of the end effector joint, including end effector and part.
<i>Eccentricity</i>	Double value that specifies eccentricity in mm around the center of the end effector joint, including end effector and part.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

Weight

**Inertia Example**

Initialize VI

**Tool Palette**

Epson Robots | System

**Description**

Initializes the instance of Spel used by the LabVIEW VI library.

**Inputs**

*Server Product Type* Optional. Specifies which EPSON RC+ product to interface with.

*Connection Number* Optional. Specifies which Controller connection to use.

*Project* Optional. Specifies the EPSON RC+ project to be used.

*ServerInstance* Optional. Specifies which instance of EPSON RC+ server to use.

*ConnectionPassword* Optional. String expression for the connection password. If the controller has a password for its connection and the password was not configured in the RC+ Setup | PC to Controller Communications dialog, then you must specify a password in order to connect with the controller.

**Outputs**

*Spel Ref Out* Spel reference output for next VI to use.

*Error Out* Error condition output for subsequent Spel nodes.

**Remarks**

The Initialize VI must be called for each instance of the library that will be used.

Server Product Type is used to specify which EPSON RC+ product to use. The default is RC70 (EPSON RC+ 7.0).

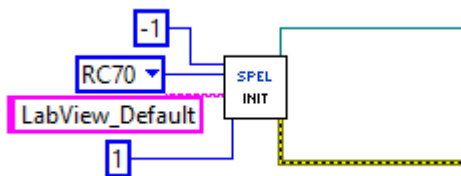
When *Connection Number* is not specified, then the connection last used in the EPSON RC+ 7.0 will be used.

When *Project* is not specified, the default LabVIEW EPSON RC+ 7.0 project will be used. The project must be used in the EPSON RC+ product specified with *Server Product Type*.

**See Also**

Shutdown

**Initialize Example**



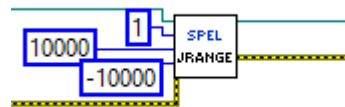
## JRange VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Defines the permissible working range of the specified robot joint in pulses.

**Inputs***Spel Ref In* Spel reference from a previous Spel Ref Out.*Error In* Error condition from a previous Spel node.*JointNumber* Integer number between 1 - 9 representing the joint for which JRange will be specified.*LowerLimitPulses* Integer number representing the encoder pulse count position for the lower limit range of the specified joint.*UpperLimitPulses* Integer number representing the encoder pulse count position for the upper limit range of the specified joint**Outputs***Spel Ref Out* Spel reference output for next VI to use.*Error Out* Error condition output for subsequent Spel nodes.**JRange Example**

JS VI

**Tool Palette**

Epson Robots | Motion

**Description**

Jump Sense detects whether the arm stopped prior to completing a JUMP instruction (which used a SENSE input) or if the arm completed the JUMP move. JS returns the Jump Sense status.

**Inputs**

*Spel Ref In* Spel reference from a previous Spel Ref Out.

*Error In* Error condition from a previous Spel node.

**Outputs**

*Spel Ref Out* Spel reference output for next VI to use.

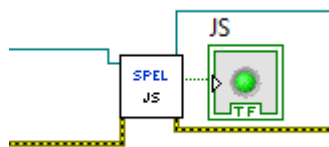
*Error Out* Error condition output for subsequent Spel nodes.

*JS* True if the SENSE input was detected during motion. False if not.

**See Also**

Jump, Sense

**JS Example**





## JTran VI

**Tool Palette**

Epson Robots | Motion

**Description**

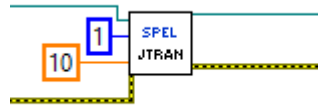
Executes a relative joint move.

**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>JointNumber</i>	The specific joint to move.
<i>Distance</i>	The distance to move. Units are in degrees for rotary joints and millimeters for linear joints.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**JTran Example**

## Jump VI

**Tool Palette**

Epson Robots | Motion

**Description**

Moves the arm from the current position to the specified point using point to point motion while first moving in a vertical direction up, then horizontally and then finally vertically downward to arrive on the final destination point.

**Inputs**

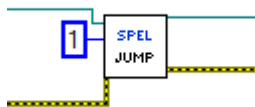
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Point Number</i>	Optional. Specifies the target end point by using the point number for a previously taught point in the Controller's point memory for the current robot. If <i>Point Expression</i> is specified, then <i>Point Number</i> is ignored.
<i>Point Expression</i>	Optional. Specifies the target end point by using a string expression. If <i>Point Expression</i> is not specified, then the <i>Point Number</i> input will be used.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

Accel, Arc, Arc3, BGo, BMove, Go, Jump3, Move, Speed, TGo, TMove

**Jump Example**

## Jump3 VI

**Tool Palette**

Epson Robots | Motion

**Description**

Motion with 3D gate using a combination of two CP motions and one PTP motion. The robot moves to the depart point, then the approach point, and finally the destination point.

**Inputs**

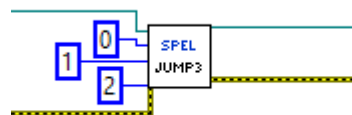
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Depart Point Number</i>	Specifies the depart point by using an integer.
<i>Depart Point Expr</i>	Specifies the depart point by using a string expression. If this input is used, then you must also specify the approach and destination points with string expressions.
<i>Appro Point Number</i>	Specifies the approach point by using an integer.
<i>Appro Point Expr</i>	Specifies the approach point by using a string expression. If this input is used, then you must also specify the depart and destination points with string expressions.
<i>Dest Point Number</i>	Specifies the destination point by using an integer.
<i>Dest Point Expr</i>	Specifies the destination point by using a string expression. If this input is used, then you must also specify the depart and approach points with string expressions.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

Accel, Arc, Arc3, BGo, BMove, Go, Jump, Move, Speed, TGo, TMove

**Jump3 Example**

## Jump3CP VI

**Tool Palette**

Epson Robots | Motion

**Description**

Motion with 3D gate using a combination of three CP motions. The robot moves to the depart point, then the approach point, and finally the destination point.

**Inputs**

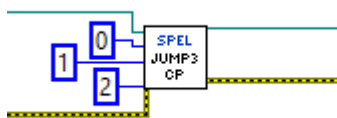
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Depart Point Number</i>	Specifies the depart point by using an integer.
<i>Depart Point Expr</i>	Specifies the depart point by using a string expression. If this input is used, then you must also specify the approach and destination points with string expressions.
<i>Appro Point Number</i>	Specifies the approach point by using an integer.
<i>Appro Point Expr</i>	Specifies the approach point by using a string expression. If this input is used, then you must also specify the depart and destination points with string expressions.
<i>Dest Point Number</i>	Specifies the destination point by using an integer.
<i>Dest Point Expr</i>	Specifies the destination point by using a string expression. If this input is used, then you must also specify the depart and approach points with string expressions.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

Accel, Arc, Arc3, BGo, BMove, Go, Jump, Jump3, Move, Speed, TGo, TMove

**Jump3CP Example**

## LimZ VI

**Tool Palette**

Epson Robots | Motion

**Description**

Sets the default value of the Z axis height for JUMP commands.

**Inputs**

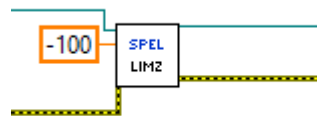
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Z Limit</i>	A coordinate value within the movable range of the Z axis.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

Jump, Jump3

**LimZ Example**

LoadPoints VI

**Tool Palette**

Epson Robots | Points

**Description**

Loads a SPEL<sup>+</sup> point file into the Controller's point memory for the current robot.

**Inputs**

- Spel Ref In* Spel reference from a previous Spel Ref Out.
- Error In* Error condition from a previous Spel node.
- File Name* A valid point file that is in the current Spel project or was previously saved with the SavePoints VI.

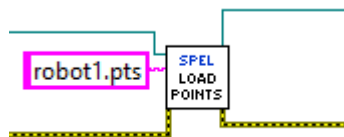
**Outputs**

- Spel Ref Out* Spel reference output for next VI to use.
- Error Out* Error condition output for subsequent Spel nodes.

**See Also**

GetPoint, Robot, SavePoints, SetPoint

**LoadPoints Example**



## MemIn VI

**Tool Palette**

Epson Robots | Inputs &amp; Outputs

**Description**

Returns the status of the specified memory I/O byte port. Each port contains 8 memory I/O bits.

**Inputs**

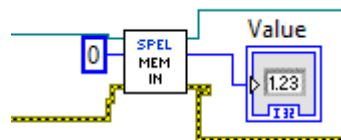
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Port Number</i>	Optional. Integer expressing one of the input ports. Each port contains 8 input bits (one byte). If <i>Label</i> is not specified, then <i>Port Number</i> is used.
<i>Label</i>	Optional. String containing an input byte label. If <i>Label</i> is specified, then <i>Port Number</i> is ignored.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.
<i>Value</i>	Integer from 0 to 255 representing the status of the port.

**See Also**

MemInW, MemOut, MemOutW

**MemIn Example**

## MemInW VI

**Tool Palette**

Epson Robots | Inputs &amp; Outputs

**Description**

Returns the status of the specified memory I/O word port. Each word port contains 16 memory I/O bits.

**Inputs**

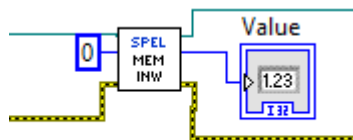
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Port Number</i>	Optional. Integer expressing one of the input ports. Each word port contains 16 input bits. If <i>Label</i> is not specified, then <i>Port Number</i> is used.
<i>Label</i>	Optional. String containing an input byte label. If <i>Label</i> is specified, then <i>Port Number</i> is ignored.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.
<i>Value</i>	Integer from 0 to 255 representing the status of the port.

**See Also**

MemIn, MemOut, MemOutW

**MemInW Example**



## MemOut VI

**Tool Palette**

Epson Robots | Inputs &amp; Outputs

**Description**

Simultaneously sets 8 memory I/O bits based on the 8 bit value specified by the user.

**Inputs**

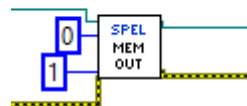
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Port Number</i>	Optional. Integer expressing one of the input ports. Each port contains 8 input bits (one byte). If <i>Label</i> is not specified, then <i>Port Number</i> is used.
<i>Label</i>	Optional. String containing an input byte label. If <i>Label</i> is specified, then <i>Port Number</i> is ignored.
<i>Value</i>	Integer containing the output pattern for the specified byte. Valid values are from 0 - 255.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

MemIn, MemInW, MemOutW

**MemOut Example**

## MemOff VI

**Tool Palette**

Epson Robots | Inputs &amp; Outputs

**Description**

Turns off the specified bit of memory I/O.

**Inputs**

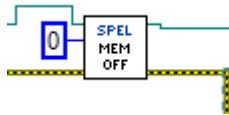
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Bit Number</i>	Optional. Integer representing one of the memory I/O bits. If <i>Label</i> is not specified, then <i>Bit Number</i> is used.
<i>Label</i>	Optional. String containing an input bit label. If <i>Label</i> is specified, then <i>Bit Number</i> is ignored.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

MemOn, MemOut, MemOutW

**MemOff Example**

## MemOn VI

**Tool Palette**

Epson Robots | Inputs &amp; Outputs

**Description**

Turns on the specified bit of memory I/O.

**Inputs**

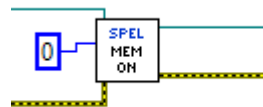
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Bit Number</i>	Optional. Integer representing one of the memory I/O bits. If <i>Label</i> is not specified, then <i>Bit Number</i> is used.
<i>Label</i>	Optional. String containing an input bit label. If <i>Label</i> is specified, then <i>Bit Number</i> is ignored.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

MemOff, MemOut, MemOutW

**MemOn Example**

## MemOut VI

**Tool Palette**

Epson Robots | Inputs &amp; Outputs

**Description**

Simultaneously sets 8 memory I/O bits based on the 8 bit value specified by the user.

**Inputs**

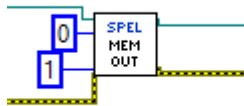
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Port Number</i>	Optional. Integer expressing one of the memory I/O ports. Each port contains 8 memory I/O bits (one byte). If <i>Label</i> is not specified, then <i>Port Number</i> is used.
<i>Label</i>	Optional. String containing a memory I/O byte label. If <i>Label</i> is specified, then <i>Port Number</i> is ignored.
<i>Value</i>	Integer containing the output pattern for the specified byte. Valid values are from 0 - 255.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

MemOn, MemOff, MemOutW

**MemOut Example**

## MemOutW VI

**Tool Palette**

Epson Robots | Inputs &amp; Outputs

**Description**

Simultaneously sets 16 memory I/O bits based on the 16 bit value specified by the user.

**Inputs**

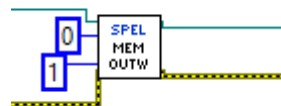
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Port Number</i>	Optional. Integer representing one of the memory I/O ports. Each word port contains 16 input bits. If <i>Label</i> is not specified, then <i>Port Number</i> is used.
<i>Label</i>	Optional. String containing an memory I/O byte label. If <i>Label</i> is specified, then <i>Port Number</i> is ignored.
<i>Value</i>	Integer containing the output pattern for the specified word. Valid values are from 0 - 65535.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

MemOn, MemOff, MemOut

**MemOutW Example**

## MemSw VI

**Tool Palette**

Epson Robots | Inputs &amp; Outputs

**Description**

Returns the status of the specified memory I/O bit.

**Inputs**

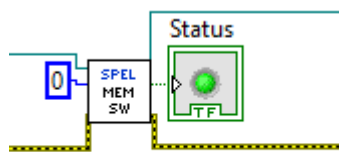
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Bit Number</i>	Optional. Integer representing one of the memory I/O bits. If <i>Label</i> is not specified, then <i>Bit Number</i> is used.
<i>Label</i>	Optional. String containing a memory I/O bit label. If <i>Label</i> is specified, then <i>Bit Number</i> is ignored.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.
<i>Value</i>	Boolean that is True with the memory I/O bit is on.

**See Also**

MemIn, MemInW

**MemSW Example**

## MotorOff VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Turns motors off for the current robot.

**Inputs***Spel Ref In* Spel reference from a previous Spel Ref Out.*Error In* Error condition from a previous Spel node.**Outputs***Spel Ref Out* Spel reference output for next VI to use.*Error Out* Error condition output for subsequent Spel nodes.**See Also**

MotorOn, PowerHigh, PowerLow, Robot

**MotorOff Example**

## MotorOn VI

**Tool Palette**

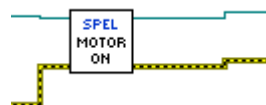
Epson Robots | Robot Settings

**Description**

Turns motors on for the current robot.

**Inputs***Spel Ref In* Spel reference from a previous Spel Ref Out.*Error In* Error condition from a previous Spel node.**Outputs***Spel Ref Out* Spel reference output for next VI to use.*Error Out* Error condition output for subsequent Spel nodes.**See Also**

MotorOff, PowerHigh, PowerLow, Robot

**MotorOn Example**

## Move VI

**Tool Palette**

Epson Robots | Motion

**Description**

Moves the arm from the current position to the specified point using linear interpolation (i.e. moving in a straight line).

**Inputs**

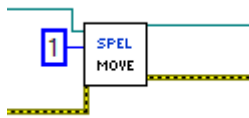
- Spel Ref In* Spel reference from a previous Spel Ref Out.
- Error In* Error condition from a previous Spel node.
- Point Number* Optional. Specifies the target end point by using the point number for a previously taught point in the Controller's point memory for the current robot. If *Point Expression* is specified, then *Point Number* is ignored.
- Point Expression* Optional. Specifies the target end point by using a string expression. If *Point Expression* is not specified, then the *Point Number* input will be used.

**Outputs**

- Spel Ref Out* Spel reference output for next VI to use.
- Error Out* Error condition output for subsequent Spel nodes.

**See Also**

AccelS, Arc, Arc3, BGo, BMove, Go, Jump, Jump3, SpeedS, TGo, TMove

**Move Example**



## Off VI

**Tool Palette**

Epson Robots | Inputs &amp; Outputs

**Description**

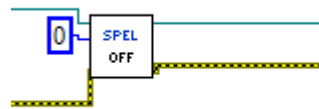
Turns off the specified output bit.

**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Bit Number</i>	Optional. Integer representing one of the output bits. If <i>Label</i> is not specified, then <i>Bit Number</i> is used.
<i>Label</i>	Optional. String containing an input bit label. If <i>Label</i> is specified, then <i>Bit Number</i> is ignored.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**Off Example**

## On VI

**Tool Palette**

Epson Robots | Inputs &amp; Outputs

**Description**

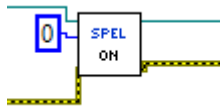
Turns on the specified output bit.

**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Bit Number</i>	Optional. Integer representing one of the output bits. If <i>Label</i> is not specified, then <i>Bit Number</i> is used.
<i>Label</i>	Optional. String containing an input bit label. If <i>Label</i> is specified, then <i>Bit Number</i> is ignored.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**On Example**

## OPort VI

**Tool Palette**

Epson Robots | Inputs &amp; Outputs

**Description**

Returns the status of the specified output bit.

**Inputs**

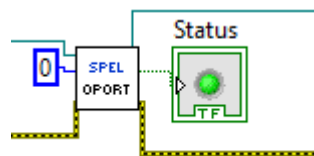
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Bit Number</i>	Optional. Integer representing one of the output bits. If <i>Label</i> is not specified, then <i>Bit Number</i> is used.
<i>Label</i>	Optional. String containing an output bit label. If <i>Label</i> is specified, then <i>Bit Number</i> is ignored.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.
<i>Value</i>	Boolean that is True with the output bit is on.

**See Also**

In, InW, On, Off, Out, Sw

**Oport Example**

OprMode VI

**Tool Palette**

Epson Robots | System

**Description**

Sets the EPSON RC+ 7.0 mode of operation..

**Inputs**

- Spel Ref In* Spel reference from a previous Spel Ref Out.
- Error In* Error condition from a previous Spel node.
- Operation Mode* The mode of operation for the associated EPSON RC+ 7.0 server process.

Mode	ID	Description
Auto	1	EPSON RC+ 7.0 is in auto mode.
Program	2	EPSON RC+ 7.0 is in program mode..

**Outputs**

- Spel Ref Out* Spel reference output for next VI to use.
- Error Out* Error condition output for subsequent Spel nodes.

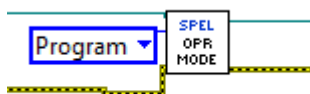
**Remarks**

When *Operation Mode* is set to Program, the EPSON RC+ 7.0 GUI for the associated server process is opened and the Controller operation mode is set to Program. If the user closes the RC+ GUI, *Operation Mode* is set to Auto. If *Operation Mode* is set to Auto, the RC+ GUI also closes.

**See Also**

GetOprMode

**OprMode Example**



## Pause VI

**Tool Palette**

Epson Robots | Tasks

**Description**

Causes all normal SPEL<sup>+</sup> tasks in the Controller to pause. If the robot is moving, it will immediately decelerate to a stop.

**Inputs**

*Spel Ref In* Spel reference from a previous Spel Ref Out.

*Error In* Error condition from a previous Spel node.

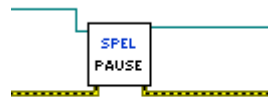
**Outputs**

*Spel Ref Out* Spel reference output for next VI to use.

*Error Out* Error condition output for subsequent Spel nodes.

**See Also**

Continue, Stop

**Pause Example**

## Plane VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Defines a plane.

**Inputs**

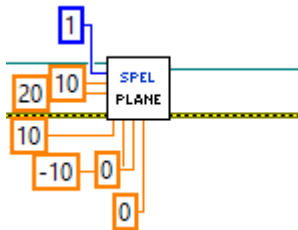
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Plane Number</i>	Integer expression from 1-15 representing which of 15 planes to define.
<i>X</i>	The plane coordinate system origin X coordinate.
<i>Y</i>	The plane coordinate system origin Y coordinate.
<i>Z</i>	The plane coordinate system origin Z coordinate.
<i>U</i>	The plane coordinate system rotation about the Z axis.
<i>V</i>	The plane coordinate system rotation about the Y axis.
<i>W</i>	The plane coordinate system rotation about the X axis.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

Box, InsideBox, InsidePlane

**Plane Example**

## PowerHigh VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Sets motor power to high for the current robot.

**Inputs***Spel Ref In* Spel reference from a previous Spel Ref Out.*Error In* Error condition from a previous Spel node.**Outputs***Spel Ref Out* Spel reference output for next VI to use.*Error Out* Error condition output for subsequent Spel nodes.**See Also**

MotorOff, MotorOn, PowerLow, Robot

**PowerHigh Example**

## PowerLow VI

**Tool Palette**

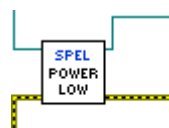
Epson Robots | Robot Settings

**Description**

Sets motor power to low for the current robot.

**Inputs***Spel Ref In* Spel reference from a previous Spel Ref Out.*Error In* Error condition from a previous Spel node.**Outputs***Spel Ref Out* Spel reference output for next VI to use.*Error Out* Error condition output for subsequent Spel nodes.**See Also**

MotorOff, MotorOn, PowerHigh, Robot

**PowerLow Example**

## Quit VI

**Tool Palette**

Epson Robots | Tasks

**Description**

Terminates execution of the specified task.

**Inputs**

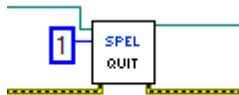
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Task Number</i>	Optional. The task number of the task to be terminated. The range of the task number is 1 to 32. If <i>Task Name</i> is specified, then <i>Task Number</i> is ignored.
<i>Task Name</i>	Optional. Specifies the name of the task to be terminated. If <i>Task Name</i> is not specified, then the <i>Task Number</i> input will be used.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

Halt, Resume

**Quit Example**



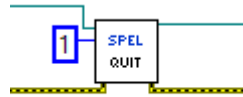
## Reset VI

**Tool Palette**

Epson Robots | System

**Description**

Resets the Controller to the initialized state.

**Inputs***Spel Ref In* Spel reference from a previous Spel Ref Out.*Error In* Error condition from a previous Spel node.**Outputs***Spel Ref Out* Spel reference output for next VI to use.*Error Out* Error condition output for subsequent Spel nodes.**Quit Example**

## Resume VI

**Tool Palette**

Epson Robots | Tasks

**Description**

Resumes a task which was suspended by the Halt VI.

**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Task Number</i>	Optional. The task number of the task to be resumed. The range of the task number is 1 to 32. If <i>Task Name</i> is specified, then <i>Task Number</i> is ignored.
<i>Task Name</i>	Optional. Specifies the name of the task to be resumed. If <i>Task Name</i> is not specified, then the <i>Task Number</i> input will be used.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

Halt, Quit

**Resume Example**

## Robot VI

**Tool Palette**

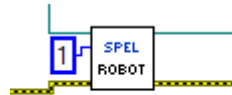
Epson Robots | Robot Settings

**Description**

Selects the current robot.

**Inputs***Spel Ref In* Spel reference from a previous Spel Ref Out.*Error In* Error condition from a previous Spel node.*Robot Number* Integer from 1-16.**Outputs***Spel Ref Out* Spel reference output for next VI to use.*Error Out* Error condition output for subsequent Spel nodes.**See Also**

GetRobot, MotorOff, MotorOn, PowerHigh, PowerLow

**Robot Example**

RunDialog VI

**Tool Palette**

Epson Robots | GUI

**Description**

Runs an EPSON RC+ 7.0 dialog.

**Inputs**

- Spel Ref In*            Spel reference from a previous Spel Ref Out.
- Error In*                Error condition from a previous Spel node.
- Dialog ID*              The ID of the EPSON RC+ 7.0 dialog to run.

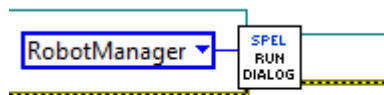
**Dialog name    ID    Description**

RobotManager	1	ID for Tools   Robot Manager dialog
ControllerTools	2	ID for Tools   Controller dialog
VisionGuide	3	ID for Tools   Vision Guide dialog

**Outputs**

- Spel Ref Out*           Spel reference output for next VI to use.
- Error Out*              Error condition output for subsequent Spel nodes.

**RunDialog Example**



## SafetyOn VI

**Tool Palette**

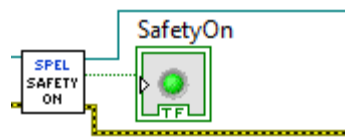
Epson Robots | System

**Description**

Returns the Safety Door open status.

**Inputs***Spel Ref In* Spel reference from a previous Spel Ref Out.*Error In* Error condition from a previous Spel node.**Outputs***Spel Ref Out* Spel reference output for next VI to use.*Error Out* Error condition output for subsequent Spel nodes.*SafetyOn* *True if the SafetyDoor is Open, otherwise is False***See Also**

EStopOn

**SafetyOn Example**

## SavePoints VI

### Tool Palette

Epson Robots | Points

### Description

Save points for the current robot into a file.

### Inputs

- Spel Ref In* Spel reference from a previous Spel Ref Out.
- Error In* Error condition from a previous Spel node.
- File Name* The name of a point file that is in the current Spel project or a new file that will be stored in the Controller.

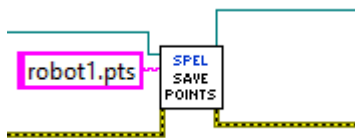
### Outputs

- Spel Ref Out* Spel reference output for next VI to use.
- Error Out* Error condition output for subsequent Spel nodes.

### See Also

GetPoint, LoadPoints, Robot, SetPoint

### SavePoints Example



## Sense VI

**Tool Palette**

Epson Robots | Motion

**Description**

Specifies input condition that, if satisfied, completes the Jump in progress by stopping the robot above the target position.

**Inputs**

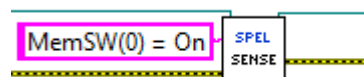
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Condition</i>	Specifies the I/O condition using a string expression. For details see the Sense Statement in the SPEL+ Language Reference manual.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

JS, Jump

**Sense Example**

SetPoint VI

**Tool Palette**

Epson Robots | Points

**Description**

Sets the coordinate data for a point for the current robot.

**Inputs**

- Spel Ref In*      Spel reference from a previous Spel Ref Out.
- Error In*        Error condition from a previous Spel node.
- Point Number*    Optional. Specifies the target end point by using the point number for a previously taught point in the Controller's point memory for the current robot. If *Point Name* is specified, then *Point Number* is ignored.
- Point Name*      Optional. Specifies the point by using a string expression for the point name. If *Point Name* is not specified, then the *Point Number* input will be used.
- X – W*            X, Y, Z, U, V, W coordinates of the specified point.

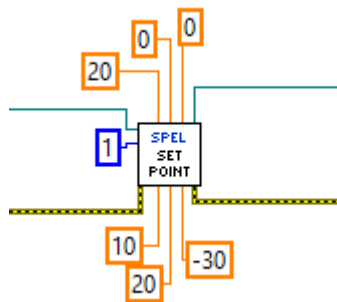
**Outputs**

- Spel Ref Out*     Spel reference output for next VI to use.
- Error Out*        Error condition output for subsequent Spel nodes.

**See Also**

GetPoint, LoadPoints, Robot, SavePoints

**SetPoint Example**





## SetVar VI

**Tool Palette**

Epson Robots | Variables

**Description**Sets the value of a SPEL<sup>+</sup> global preserve variable in the Controller.**Inputs**

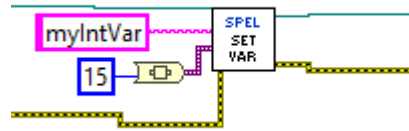
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Var Name</i>	The name of the SPEL <sup>+</sup> global preserve variable.
<i>Value</i>	A variant containing the value.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

GetVar

**SetVar Example**

## SFree VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Free joint the specified robot axes.

**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Axes</i>	Optional. Integer array specifying which axes to free. If omitted, all axes are freed.

**Outputs**

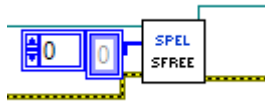
<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**Remarks**

If Axes is omitted, then all axes are freed.

**See Also**

MotorOff, MotorOn, SLock

**SFree Example**

## ShowWindow VI

**Tool Palette**

Epson Robots | GUI

**Description**

Displays an EPSON RC+ 7.0 window.

**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Window ID</i>	The ID of the EPSON RC+ 7.0 window to show.

**Window name ID Description**

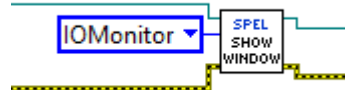
IOMonitor	1	ID for the I/O Monitor window.
TaskManager	2	ID for the Task Manager window.
ForceMonitor	3	ID for the Force Monitor window.
Simulator	4	ID for the Simulator window.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

HideWindow, RunDialog

**ShowWindow Example**

Shutdown VI

**Tool Palette**

Epson Robots | System

**Description**

Shuts down the EPSON RC+ 7.0 server process that was started when the Initialize VI was called.

**Inputs**

*Spel Ref In* Spel reference from a previous Spel Ref Out.

*Error In* Error condition from a previous Spel node.

**Outputs**

*Error Out* Error condition output for subsequent Spel nodes.

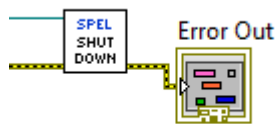
**Remarks**

The Shutdown VI must be called for each instance of the library. This will shutdown the associated EPSON RC+ 7.0 server process.

**See Also**

Initialize

**Shutdown Example**



## SLock VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Release the free joint state for the specified robot axes.

**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Axes</i>	Optional. Integer array specifying which axes to lock. If omitted, all axes are locked.

**Outputs**

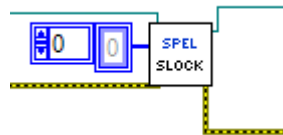
<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**Remarks**

If Axes is omitted, then all axes are locked.

**See Also**

MotorOff, MotorOn, SFree

**SLock Example**

## Speed VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Specifies the arm speed for use with the point to point instructions Go, Jump and Pulse.

**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>PointToPoint Speed</i>	Specifies the arm speed for use with the point to point instructions Go, Jump and Pulse.
<i>Depart Speed</i>	Integer number between 1-100 representing the Z axis upward motion speed for the Jump instruction.
<i>Appro Speed</i>	Integer number between 1-100 representing the Z axis downward motion speed for the Jump instruction.

**Outputs**

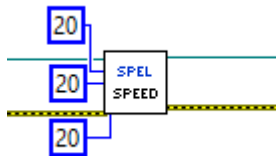
<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**Remarks**

Use Speed to set the point to point speed for the current robot. All values can be from 1 to 100%. If *Depart Speed* is specified, then *Appro Speed* must also be specified.

**See Also**

Accel, AccelS, SpeedS

**Speed Example**

## SpeedS VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Specifies the arm speed for use with the Continuous Path instructions Jump3CP, Move, Arc, and CVMove.

**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Linear Speed</i>	Specifies the arm speed for use with the Continuous Path instructions Jump3CP, Move, Arc, and CVMove.
<i>Depart Speed</i>	Double value between 1-5000 representing the Z axis upward motion speed for the Jump3CP instruction.
<i>Appro Speed</i>	Double value between 1-5000 representing the Z axis downward motion speed for the Jump3CP instruction.

**Outputs**

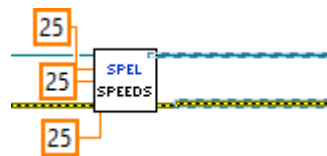
<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**Remarks**

Use Speed to set the linear speed for the current robot in millimeters / sec. If *Depart Speed* is specified, then *Appro Speed* must also be specified.

**See Also**

Accel, AccelS, Speed

**SpeedS Example**

Start VI

**Tool Palette**

Epson Robots | Tasks

**Description**

Starts a program that will run in the Controller.

**Inputs**

*Spel Ref In* Spel reference from a previous Spel Ref Out.

*Error In* Error condition from a previous Spel node.

*ProgramNumber* The program number to start, corresponding to the 8 main functions in SPEL+ as shown in the table below. The range is 0 to 7.

Program Number	SPEL+ Function Name
0	main
1	main1
2	main2
3	main3
4	main4
5	main5
6	main6
7	main7

**Outputs**

*Spel Ref Out* Spel reference output for next VI to use.

*Error Out* Error condition output for subsequent Spel nodes.

**Remarks**

When **Start** is executed, control will return immediately to the calling VI. You cannot start a program that is already running. Note that Start causes global variables in the Controller to be cleared and default robot points to be loaded.

**See Also**

Continue, Pause, Stop, Xqt

**Start Example**





## Stop VI

**Tool Palette**

Epson Robots | Tasks

**Description**

Stops all normal SPEL<sup>+</sup> tasks running in the Controller and optionally stops all background tasks.

**Inputs**

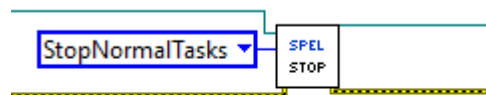
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Stop Type</i>	Optional. Specify StopNormalTasks (default) or StopAllTasks (also stop background tasks).

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

Continue, Pause, Start, Xqt

**Stop Example**

## Sw VI

**Tool Palette**

Epson Robots | Inputs &amp; Outputs

**Description**

Returns the status of the specified input bit.

**Inputs**

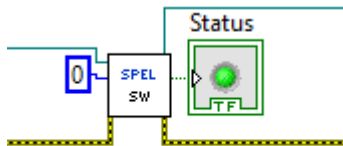
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Bit Number</i>	Optional. Integer representing one of the input bits. If <i>Label</i> is not specified, then <i>Bit Number</i> is used.
<i>Label</i>	Optional. String containing an input bit label. If <i>Label</i> is specified, then <i>Bit Number</i> is ignored.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.
<i>Value</i>	Boolean that is True with the output bit is on.

**See Also**

In, InW, On, Off, OPort, Out

**SW Example**

## TargetOK VI

**Tool Palette**

Epson Robots | Motion

**Description**

Returns a status indicating whether or not the PTP (Point to Point) motion from the current position to a target position is possible.

**Inputs**

*Spel Ref In* Spel reference from a previous Spel Ref Out.

*Error In* Error condition from a previous Spel node.

*Point Number* Optional. Specifies the target end point by using the point number for a previously taught point in the Controller's point memory for the current robot. If *Point Expression* is specified, then *Point Number* is ignored.

*Point Expression* Optional. Specifies the target end point by using a string expression. If *Point Expression* is not specified, then the *Point Number* input will be used.

**Outputs**

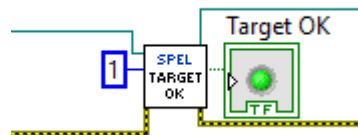
*Spel Ref Out* Spel reference output for next VI to use.

*Target OK* The robot can move to the target position.

*At Home* Boolean indicating if the current robot is at the home position.

**See Also**

BGo, Go, Jump, TGo

**TargetOK Example**

## TGo VI

**Tool Palette**

Epson Robots | Motion

**Description**

Executes Point to Point relative motion, in the current tool coordinate system.

**Inputs**

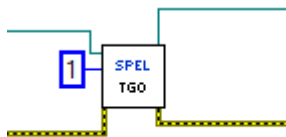
- Spel Ref In* Spel reference from a previous Spel Ref Out.
- Error In* Error condition from a previous Spel node.
- Point Number* Optional. Specifies the target end point by using the point number for a previously taught point in the Controller's point memory for the current robot. If *Point Expression* is specified, then *Point Number* is ignored.
- Point Expression* Optional. Specifies the target end point by using a string expression. If *Point Expression* is not specified, then the *Point Number* input will be used.

**Outputs**

- Spel Ref Out* Spel reference output for next VI to use.
- Error Out* Error condition output for subsequent Spel nodes.

**See Also**

Accel, Arc, Arc3, BGo, BMove, Go, Jump, Jump3, Move, Speed, TMove

**TGo Example**

## Till VI

**Tool Palette**

Epson Robots | Motion

**Description**

Specifies event condition that, if satisfied, completes the motion command (Jump, Go, Move, etc.) in progress by decelerating and stopping the robot at an intermediate position.

**Inputs**

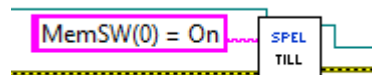
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Condition</i>	Specifies the I/O condition using a string expression. For details see the Sense Statement in the SPEL+ Language Reference manual.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

Accel, Arc, Arc3, BGo, BMove, Jump, Jump3, Move, Speed, TGo, TillOn, TMove

**Till Example**

TillOn VI

**Tool Palette**

Epson Robots | Motion

**Description**

Returns True if a stop has occurred from a till condition during the last Go/Jump/Move statement.

**Inputs**

*Spel Ref In* Spel reference from a previous Spel Ref Out.

*Error In* Error condition from a previous Spel node.

**Outputs**

*Spel Ref Out* Spel reference output for next VI to use.

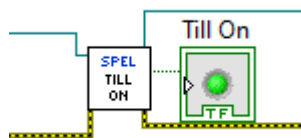
*Error Out* Error condition output for subsequent Spel nodes.

*Till On* True if the till condition was detected during motion. False if not.

**See Also**

Accel, Arc, Arc3, BGo, BMove, Jump, Jump3, Move, Speed, TGo, Till, TMove

**TillOn Example**



## TLSet VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

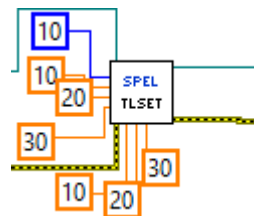
Defines an ECP (external control point).

**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>ToolNumber</i>	Integer expression from 1-15 representing which of 15 tools to define. (Tool 0 is the default tool and cannot be modified.)
<i>X</i>	The tool coordinate system origin X coordinate.
<i>Y</i>	The tool coordinate system origin Y coordinate.
<i>Z</i>	The tool coordinate system origin Z coordinate.
<i>U</i>	The tool coordinate system rotation about the Z axis.
<i>V</i>	The tool coordinate system rotation about the Y axis.
<i>W</i>	The tool coordinate system rotation about the X axis.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**TLSet Example**

## TMove VI

**Tool Palette**

Epson Robots | Motion

**Description**

Executes linear interpolation relative motion, in the current tool coordinate system.

**Inputs**

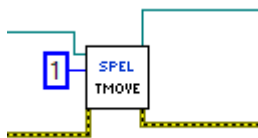
- Spel Ref In* Spel reference from a previous Spel Ref Out.
- Error In* Error condition from a previous Spel node.
- Point Number* Optional. Specifies the target end point by using the point number for a previously taught point in the Controller's point memory for the current robot. If *Point Expression* is specified, then *Point Number* is ignored.
- Point Expression* Optional. Specifies the target end point by using a string expression. If *Point Expression* is not specified, then the *Point Number* input will be used.

**Outputs**

- Spel Ref Out* Spel reference output for next VI to use.
- Error Out* Error condition output for subsequent Spel nodes.

**See Also**

AccelS, Arc, Arc3, BGo, BMove, Go, Jump, Jump3, Move, SpeedS, TGo

**TMove Example**



## Tool VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Selects the current robot tool.

**Inputs***Spel Ref In* Spel reference from a previous Spel Ref Out.*Error In* Error condition from a previous Spel node.*Tool Number* Integer number from 0-15 representing which of 16 tool definitions to use with subsequent motion instructions.**Outputs***Spel Ref Out* Spel reference output for next VI to use.*Error Out* Error condition output for subsequent Spel nodes.**See Also**

Arm, Armset, GetTool, TLSet

**Tool Example**

TW VI

**Tool Palette**

Epson Robots | Inputs & Outputs

**Description**

Returns the status of the wait condition.

**Inputs**

*Spel Ref In* Spel reference from a previous Spel Ref Out.

*Error In* Error condition from a previous Spel node.

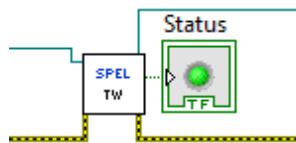
**Outputs**

*Spel Ref Out* Spel reference output for next VI to use.

*Error Out* Error condition output for subsequent Spel nodes.\

*Status* Returns the status of the wait condition.

**TW Example**



## VGetBool VI

**Tool Palette**

Epson Robots | Vision

**Description**

Retrieves a vision property or result that returns a Boolean value.

**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Sequence</i>	The name of a vision sequence in the current project.
<i>Object</i>	Optional. The name of a vision object in the specified sequence. Omit when retrieving a property or result for a sequence.
<i>Property Code</i>	The property or result code.
<i>Result Index</i>	Optional. The index of the result.

**Outputs**

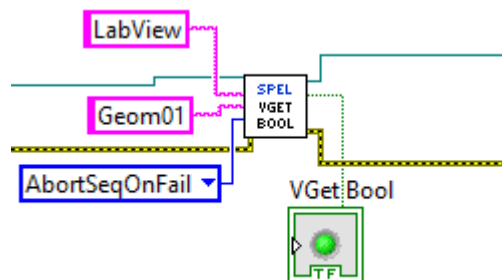
<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.
<i>Value</i>	The Boolean value.

**Remarks**

See the Vision Guide 7.0 Properties and Results Reference manual for details on Vision Guide properties and results.

**See Also**

VRun, VGetDbl, VGetInt, VGetStr, VSetBool, VSetDbl, VSetInt, VSetStr

**VGetBool Example**

## VGetDbI VI

**Tool Palette**

Epson Robots | Vision

**Description**

Retrieves a vision property or result that returns a double value.

**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Sequence</i>	The name of a vision sequence in the current project.
<i>Object</i>	Optional. The name of a vision object in the specified sequence. Omit when retrieving a property or result for a sequence.
<i>Property Code</i>	The property or result code.
<i>Result Index</i>	Optional. The index of the result.

**Outputs**

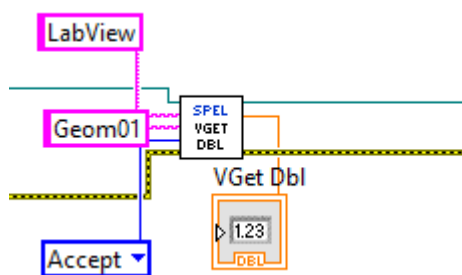
<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.
<i>Value</i>	The double value.

**Remarks**

See the Vision Guide 7.0 Properties and Results Reference manual for details on Vision Guide properties and results.

**See Also**

VRun, VGetBool, VGetInt, VGetStr, VSetBool, VSetDbI, VSetInt, VSetStr

**VGetDbI Example**

## VGetInt VI

**Tool Palette**

Epson Robots | Vision

**Description**

Retrieves a vision property or result that returns an integer value.

**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Sequence</i>	The name of a vision sequence in the current project.
<i>Object</i>	Optional. The name of a vision object in the specified sequence. Omit when retrieving a property or result for a sequence.
<i>Property Code</i>	The property or result code.
<i>Result Index</i>	Optional. The index of the result.

**Outputs**

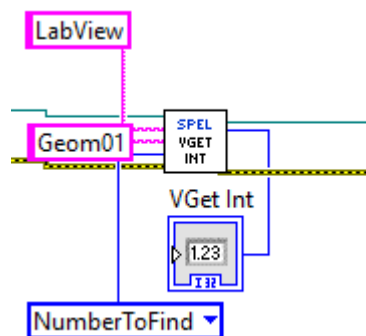
<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.
<i>Value</i>	The integer value.

**Remarks**

See the Vision Guide 7.0 Properties and Results Reference manual for details on Vision Guide properties and results.

**See Also**

VRun, VGetBool, VGetDbl, VGetStr, VSetBool, VSetDbl, VSetInt, VSetStr

**VGetInt Example**

## VGetStr VI

**Tool Palette**

Epson Robots | Vision

**Description**

Retrieves a vision property or result that returns a string value.

**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Sequence</i>	The name of a vision sequence in the current project.
<i>Object</i>	Optional. The name of a vision object in the specified sequence. Omit when retrieving a property or result for a sequence.
<i>Property Code</i>	The property or result code.
<i>Result Index</i>	Optional. The index of the result.

**Outputs**

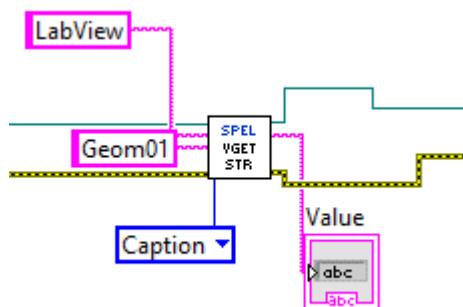
<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.
<i>Value</i>	The string value.

**Remarks**

See the Vision Guide 7.0 Properties and Results Reference manual for details on Vision Guide properties and results.

**See Also**

VRun, VGetBool, VGetDbl, VGetInt, VSetBool, VSetDbl, VSetInt, VSetStr

**VGetStr Example**

## VideoControl VI

**Tool Palette**

Epson Robots | Vision

**Description**

Controls a SPEL Video control.

**Inputs**

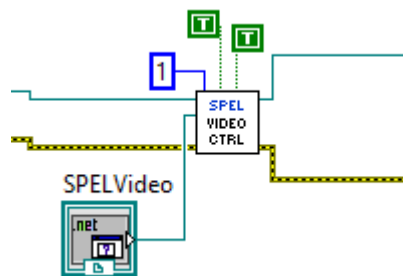
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Sequence</i>	The name of a vision sequence in the current project.
<i>VideoRef In</i>	Reference from a SPEL Video control.
<i>Camera</i>	Sets which camera video to display. Default is 0, which displays any camera.
<i>Graphics Enabled</i>	Sets whether graphics should be displayed.
<i>Video Enabled</i>	Sets whether video should be displayed.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

Displaying Video, VGet, VRun

**VideoControl Example**

**VRun VI**

**Tool Palette**

Epson Robots | Vision

**Description**

Run a vision sequence in the current project.

**Inputs**

- Spel Ref In* Spel reference from a previous Spel Ref Out.
- Error In* Error condition from a previous Spel node.
- Sequence* The name of a vision sequence in the current project.

**Outputs**

- Spel Ref Out* Spel reference output for next VI to use.
- Error Out* Error condition output for subsequent Spel nodes.

**Remarks**

Refer to the Vision Guide 7.0 software manual for information on running vision sequences.

**See Also**

VGetBool, VGetDbl, VGetInt, VGetStr, VSetBool, VSetDbl, VSetInt, VSetStr

**VRun Example**





## VSetBool VI

**Tool Palette**

Epson Robots | Vision

**Description**

Sets the value of a vision property whose data type is Boolean.

**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Sequence</i>	The name of a vision sequence in the current project.
<i>Object</i>	Optional. The name of a vision object in the specified sequence. Omit when setting a property for a sequence.
<i>Property Code</i>	The property code.
<i>Value</i>	The new Boolean value of the property.

**Outputs**

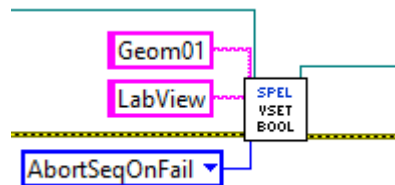
<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**Remarks**

See the Vision Guide 7.0 Properties and Results Reference manual for details on Vision Guide properties and results.

**See Also**

VGetBool, VGetDbl, VGetInt, VGetStr, VRun, VSetDbl, VSetInt, VSetStr

**VSetBool Example**

VSetDbl VI

**Tool Palette**

Epson Robots | Vision

**Description**

Sets the value of a vision property whose data type is real or double.

**Inputs**

- Spel Ref In*      Spel reference from a previous Spel Ref Out.
- Error In*        Error condition from a previous Spel node.
- Sequence*        The name of a vision sequence in the current project.
- Object*            Optional. The name of a vision object in the specified sequence. Omit when setting a property for a sequence.
- Property Code*    The property code.
- Value*             The new double value of the property.

**Outputs**

- Spel Ref Out*     Spel reference output for next VI to use.
- Error Out*        Error condition output for subsequent Spel nodes.

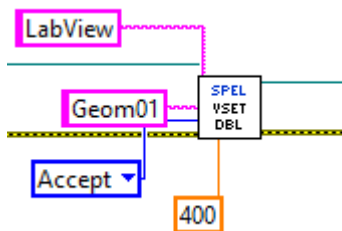
**Remarks**

See the Vision Guide 7.0 Properties and Results Reference manual for details on Vision Guide properties and results.

**See Also**

VGetBool, VGetDbl, VGetInt, VGetStr, VRun, VSetBool, VSetInt, VSetStr

**VSetDbl Example**



## VSetInt VI

**Tool Palette**

Epson Robots | Vision

**Description**

Sets the value of a vision property whose data type is integer.

**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Sequence</i>	The name of a vision sequence in the current project.
<i>Object</i>	Optional. The name of a vision object in the specified sequence. Omit when setting a property for a sequence.
<i>Property Code</i>	The property code.
<i>Value</i>	The new integer value of the property.

**Outputs**

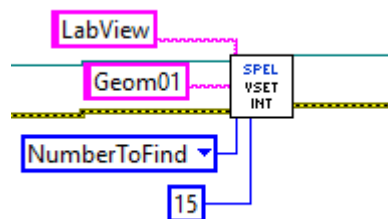
<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**Remarks**

See the Vision Guide 7.0 Properties and Results Reference manual for details on Vision Guide properties and results.

**See Also**

VGetBool, VGetDbl, VGetInt, VGetStr, VRun, VSetBool, VSetDbl, VSetStr

**VSetInt Example**

VSetStr VI

**Tool Palette**

Epson Robots | Vision

**Description**

Sets the value of a vision property whose data type is string.

**Inputs**

- Spel Ref In*      Spel reference from a previous Spel Ref Out.
- Error In*        Error condition from a previous Spel node.
- Sequence*        The name of a vision sequence in the current project.
- Object*           Optional. The name of a vision object in the specified sequence. Omit when setting a property for a sequence.
- Property Code*    The property code.
- Value*            The new string value of the property.

**Outputs**

- Spel Ref Out*     Spel reference output for next VI to use.
- Error Out*        Error condition output for subsequent Spel nodes.

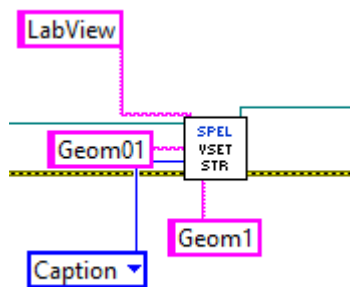
**Remarks**

See the Vision Guide 7.0 Properties and Results Reference manual for details on Vision Guide properties and results.

**See Also**

VGetBool, VGetDbl, VGetInt, VGetStr, VRun, VSetBool, VSetDbl, VSetInt

**VSetStr Example**



## WaitTaskDone VI

**Tool Palette**

Epson Robots | Tasks

**Description**

Waits for a task to finish and returns the status.

**Inputs**

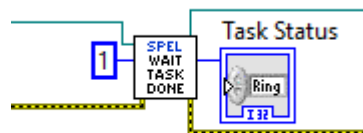
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Task Number</i>	Optional. The task number of the task to be suspended. The range of the task number is 1 to 32. If <i>Task Name</i> is specified, then <i>Task Number</i> is ignored.
<i>Task Name</i>	Optional. Specifies the name of the task to be suspended. If <i>Task Name</i> is not specified, then the <i>Task Number</i> input will be used.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.
<i>Task State</i>	Indicates the final status of the task (Quit, Aborted, Finished).

**See Also**

Xqt

**WaitTaskDone Example**

Weight VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Specifies the weight parameters for the current robot.

**Inputs**

- Spel Ref In* Spel reference from a previous Spel Ref Out.
- Error In* Error condition from a previous Spel node.
- Payload Weight* The weight of the end effector to be carried in Kg units.
- Arm Length* The distance from the rotational center of the second arm to the center of the gravity of the end effector in mm units.

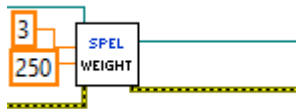
**Outputs**

- Spel Ref Out* Spel reference output for next VI to use.
- Error Out* Error condition output for subsequent Spel nodes.

**See Also**

Inertia

**Weight Example**



## Xqt VI

**Tool Palette**

Epson Robots | Tasks

**Description**Start one SPEL<sup>+</sup> task.**Inputs**

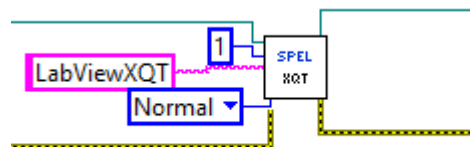
<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Task Number</i>	Optional. The task number of the task to execute. The range of the task number is 1 to 32. If <i>Task Number</i> is omitted, then a task number will automatically be assigned.
<i>Func Name</i>	Specifies the name of the function to be executed.

**Outputs**

<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

Halt, Quit, Resume, WaitForTaskDone

**XQT Example**

## XYLim VI

**Tool Palette**

Epson Robots | Robot Settings

**Description**

Sets the permissible motion range limits for the Manipulator.

**Inputs**

<i>Spel Ref In</i>	Spel reference from a previous Spel Ref Out.
<i>Error In</i>	Error condition from a previous Spel node.
<i>Min X</i>	The minimum X coordinate position to which the Manipulator may travel. (The Manipulator may not move to a position with the X Coordinate less than min X.)
<i>Max X</i>	The maximum X coordinate position to which the Manipulator may travel. (The Manipulator may not move to a position with the X Coordinate greater than max X.)
<i>Min Y</i>	The minimum Y coordinate position to which the Manipulator may travel. (The Manipulator may not move to a position with the Y Coordinate less than min Y.)
<i>Max Y</i>	The maximum Y coordinate position to which the Manipulator may travel. (The Manipulator may not move to a position with the Y Coordinate greater than max Y.)
<i>Min Z</i>	The minimum Z coordinate position to which the Manipulator may travel. (The Manipulator may not move to a position with the Z Coordinate less than min Z.)
<i>Max Z</i>	The maximum Z coordinate position to which the Manipulator may travel. (The Manipulator may not move to a position with the Z Coordinate greater than max Z.)

**Remarks**

XYLim is used to define motion range limits. Many robot systems allow users to define joint limits but the SPEL+ language allows both joint limits and motion range limits to be defined. In effect this allows users to create a work envelope for their application. (Keep in mind that joint range limits are also definable with SPEL.)

The motion range established with XYLim values applies to motion command target positions only, and not to motion paths from starting position to target position. Therefore, the arm may move outside the XYLim range during motion. (i.e. The XYLim range does not affect Pulse.)

To turn off motion range limits, specify 0 for the range limit parameters.

**Outputs**

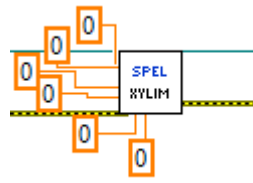
<i>Spel Ref Out</i>	Spel reference output for next VI to use.
<i>Error Out</i>	Error condition output for subsequent Spel nodes.

**See Also**

Box



### XYLim Example



## 17. Using LabVIEW with RCNetLib

### 17.1 Overview

The LabVIEW VI library described in the chapter Using the LabVIEW VI Library is a high level interface that uses the RCAPINet.dll. Some users may want to interface with RCAPINet.dll directly instead of using the high level library. This chapter contains information for using LabVIEW with RCAPINet.dll. The following topics are described.

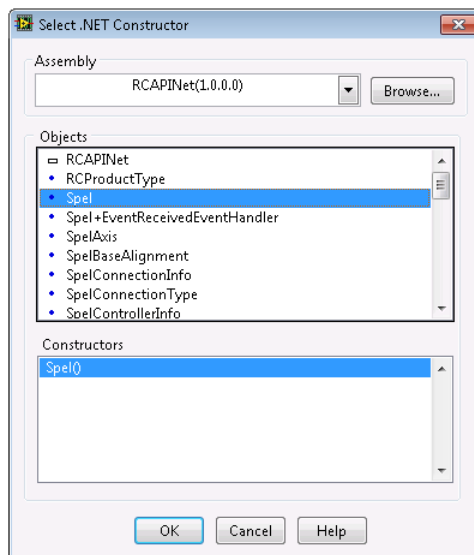
- Initialization
- Use Spel properties and methods in your application
- Shutdown
- Using dialogs and windows

### 17.2 Initialization

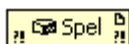
#### 17.2.1 Add a constructor node for the Spel class

Before you can call methods or use properties from the Spel class, you must create an instance of the Spel class using a Constructor Node. You should use one Spel class instance in your application.

In the Block Diagram view of the VI that will contain the Spel class instance, add a Constructor Node from the [RC+ API] – [.NET palette]. The [Select .NET Constructor] dialog will appear. Select “RCAPINet” in the [Assembly] list and select “Spel” in the [Objects] list, as shown below.

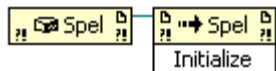


Click <OK> to create a constructor node for Spel in the block diagram.



### 17.2.2 Initialize the Spel class instance

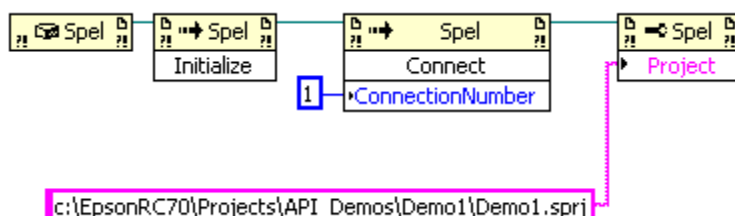
Add an Invoke node for the Spel class Initialize method. When Initialize executes, it will configure and start RC+ as a server in the background.



### 17.2.3 Connect to Controller and set project

Add an Invoke node for the Spel class Connect method. Set the ConnectionNumber parameter for the Controller connection you want to use. To view the connection numbers, start EPSON RC+ 7.0, then select [Setup]-[PC to Controller Communications].

Add a Property node for the Spel class Project property. Set the Project parameter to the desired project file.



## 17.3 Use Spel properties and methods

Add more nodes to use the Spel properties and methods for your application. You must wire the reference output from the previous node to the reference input of the current node. This allows each property or method to use the Spel class instance you created and initialized in the steps above. Refer to the RCAPINet Reference chapter for information on the properties and methods that can be used.

## 17.4 Shutdown

When you are finished using the Spel class instance, you need to invoke the Dispose method. This will shutdown the EPSON RC+ 7.0 server that is associated with the Spel class instance. Normally, you should call Dispose at the end of your application.

If your application is aborted without calling Dispose, then the RC+ process continues to run, because LabVIEW (the client process) continues to run. If you start your application again, the RC+ process is restarted if it was running. But if you try to run the RC+ GUI, it will ask if you want to run another instance of RC+. In this case, you can terminate the RC+ process (erc70.exe) from the Windows Task Manager first, then run the EPSON RC+ 7.0 GUI.

## 17.5 Using Dialogs and Windows

When used with .NET applications, a .NET parent form is normally used as the parent for dialogs and windows that are displayed from the Spel class instance. But LabVIEW does not use .NET forms, so to display windows and dialogs from LabVIEW, use the ParentWindowHandle property. Set it to the window handle of your VI. You can call the Windows API FindWindow method to get the window handle.

When using ParentWindowHandle, you must call Spel.ShowWindow without the Parent parameter.

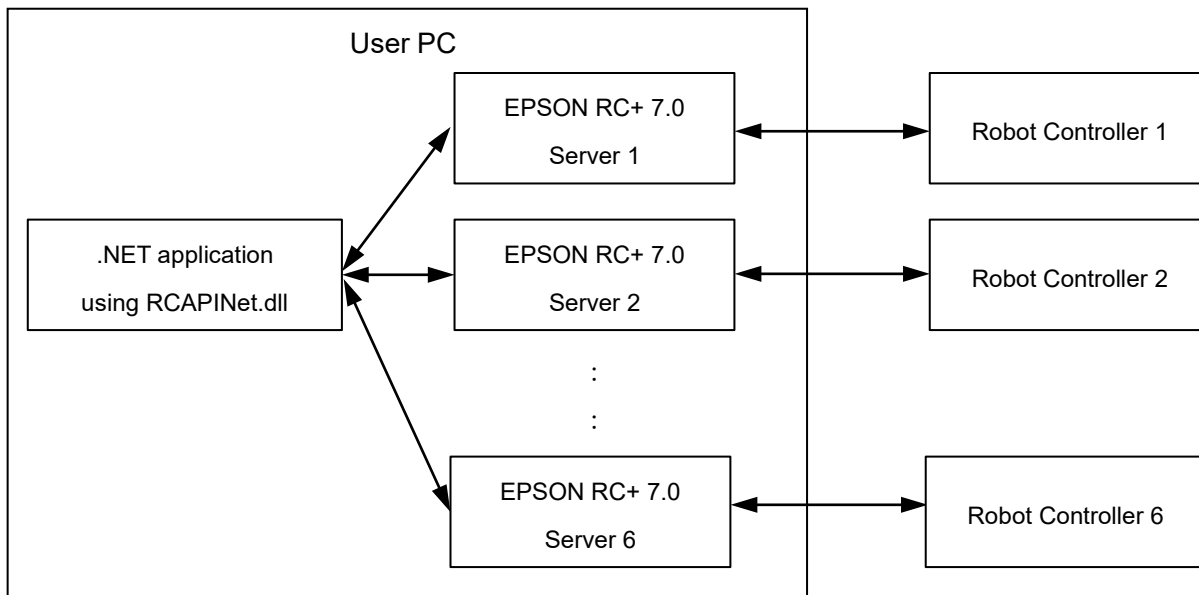
# 18. How to Control Multiple Controllers from One PC

## 18.1 Overview

Using the RC+ API, you can control up to six Robot Controllers from one PC.

To control multiple Controllers, an RCAPINet Spel class needs to be instantiated for each Controller.

The figure below shows the basic system configuration diagram for controlling multiple Controllers using the RC+ API.




The application controls the multiple Controllers via the servers (RCAPINet Spel class) prepared for each Controller.

### 18.1.1 System Requirements

We recommend a PC that satisfies the following requirements.

OS	Windows 8.1 Pro 64-bit version Windows 10 Pro 64-bit version Windows 11 Pro 64-bit version
CPU	Core i5 or greater
Memory	2 GB or more



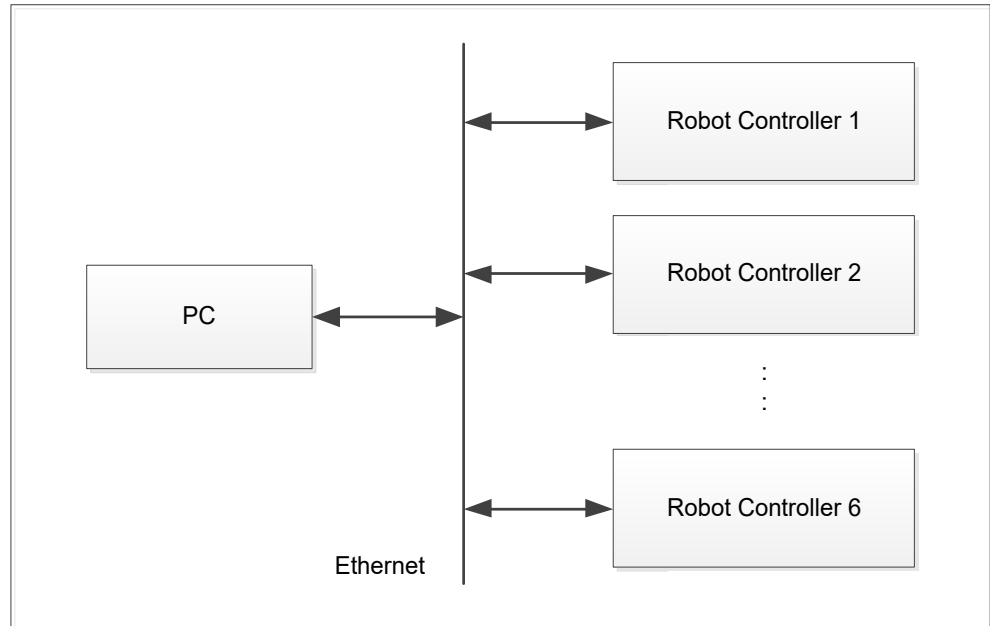
CAUTION

- When using a PC that does not satisfy the above requirements, make sure you check operation adequately before using the system in production. If a PC that does not satisfy the above requirements is used, the Controllers may not be controlled reliably.

### 18.1.2 Connection of PC and Controllers

The connection type for the first Controller can be USB or Ethernet. The connection type for the remaining Controllers must be Ethernet.

The figure below shows the basic configuration diagram of the PC and the multiple Controllers.



Robot Controller supports the Controllers that connect to EPSON RC+ 7.0.



Controllers that connect to EPSON RC+ 7.0 can be connected at the same time.



One virtual Controller can be selected.



One Controller can be connected by the USB communication.  
 When using the USB communication, connect only one Controller by the USB communication and connect other Controllers by the Ethernet.



- If the anti-virus software is installed on the PC, communication with the Controllers may be disconnected abnormally when running a virus scan. To run a virus scan, disconnected the communication with the Controllers beforehand.

### 18.2 Restrictions on controlling multiple Controllers

Controlling multiple Controllers has restrictions as described in the following sections.

#### 18.2.1 Restrictions on Controller options

The following Controller options controlled by each Controller have restrictions.

- PC vision
- Fieldbus master
- Force sensing  
(Force sensing and Force Sensor are different.)

When one of the above three options is already connected to the active Controller, these options cannot be used for other (the second or later) Controllers.

#### 18.2.2 Restrictions on simulator

##### Simulator window display

EPSON RC+ 7.0 simulator window can be used from the .NET application.  
For details, refer to *10.1 Windows* in this manual.

If the simulator window is opened for each Controller when multiple Controllers are connected, the cycle time may increase by 100 to 200 msec compared to not displaying the simulator windows.

Also, if the program is executed with the simulator windows open, the CPU utilization increases near 100 % and a huge load may be put on the PC.

It is recommended to use the system with the simulator windows closed, except when debugging the program.

##### Collision detection

To avoid collision with peripherals using the simulator, set 15 mm or more margins around the simulator object to avoid collision detection.

Collision detection in the simulator does not guarantee the accuracy. When applying to the actual system, make sure to set the margins and check operation adequately.

For details on each restriction, refer to *8.4 Simulator Specifications and Restrictions* in *EPSON RC+ User's Guide*.

### 18.3 Sample Program for connecting multiple Controllers

The following sections describe sample programs to connect the PC with Controller 1 and Controller 2 using a .NET application.



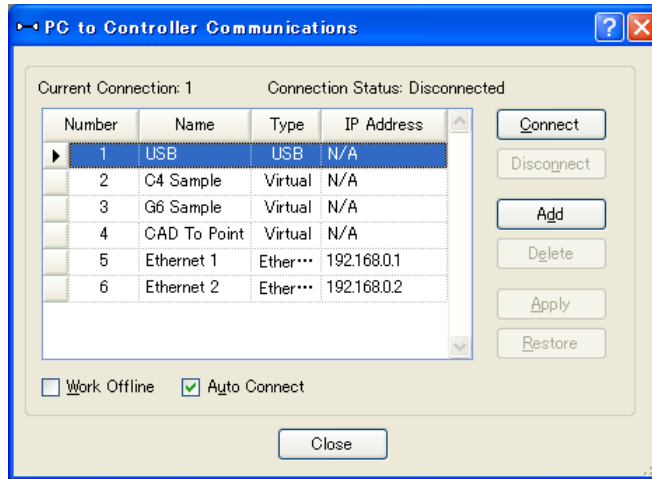
For details on available properties and methods, refer to *14. RCAPINet Reference* in this manual.

#### 18.3.1 Controller connection setting

When connecting the multiple Controllers at the same time, specify the connection using the Connect method of Spel class.

```
m_spel.Connect(1)
```

The parameter in the Connect method indicates the connection number. This number is same as the one shown in “Number” in the dialog box below (EPSON RC+ 7.0 menu-[Setup]-[PC to Controller Communications]). If a value of -1 is used, it means to use the most recent connection.



From RC+7.5.0 or later, you can connect to Connect method by the connection name (“Name” in the above window).

### 18.3.2 Project setting

To connect the multiple controllers, specify the project using the Project property of the Spel class. Each controller must use a separate project.

```
m_spel.Project = "c:\EpsonRC70\projects\API_Demos\Demo1\Demo1.sprj"
```

### 18.3.3 Sample program using Visual Basic

- (1) Select menu-[File]-[New]-[Project] in Visual Studio .NET.
- (2) Create a Visual Basic project.
- (3) Select [Project]-[Add Reference].
- (4) Select the [Browse] tab, reference “\EpsonRC70\Exe” directory, and then select the “RCAPINet.dll” file.
- (5) Add two buttons (btnController1, btnController2) to the Form1 class.
- (6) Add quick events of each button and create the thread to control each Robot Controller.

```
Private trd1 As System.Threading.Thread ' for Robot Controller 1
Private trd2 As System.Threading.Thread ' for Robot Controller 2

Private Sub btnController1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnController1.Click
    ' Start thread for Robot Controller 1
    trd1 = New System.Threading.Thread( _
        New System.Threading.ThreadStart(AddressOf StartController1))
    trd1.Start()
End Sub
Private Sub StartController1()
    ' Control Robot Controller 1
    Try
        Dim frm1 As New frmDemo1
        frm1.ShowDialog()
        frm1.Dispose()
    Catch ex As Exception
        MsgBox(ex.Message)
    End Try
End Sub
```

```

        End Try
    End Sub
    Private Sub btnController2_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnController2.Click
        ' Start thread for Robot Controller 2
        trd2 = New System.Threading.Thread( _
            New System.Threading.ThreadStart(AddressOf StartController2))
        trd2.Start()
    End Sub
    Private Sub StartController2()
        ' Control Robot Controller 2
        Try
            Dim frm2 As New frmDemo2
            frm2.ShowDialog()
            frm2.Dispose()
        Catch ex As Exception
            MsgBox(ex.Message)
        End Try
    End Sub

```

- (7) Add a form (frmDemo1) for Controller 1.

```

    Private WithEvents m_spell As New Spel
    Private Sub frmDemo1_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load

        Try
            m_spell.ServerInstance=1
            m_spell.Initialize()
            m_spell.Connect(5)
            m_spell.Project = "
c:\\EpsonRC70\\Projects\\Demo1\\Demo1.sprj "
        Catch ex As SpelException
            MsgBox(ex.Message)
        End Try
    End Sub
    Private Sub m_spell_EventReceived(ByVal sender As Object, ByVal e
    As
        SpelEventArgs) _Handles m_spell.EventReceived
        ' for Robot Controller 1
    End Sub
    Private Sub frmDemo1_FormClosed(ByVal sender As System.Object, _
        ByVal e As System.Windows.Forms.FormClosedEventArgs)
        -
        Handles MyBase.FormClosed
        m_spell.Dispose()
    End Sub

```

- (8) Add a form (frmDemo2) for Controller 2.

```

    Private WithEvents m_spell2 As New Spel
    Private Sub frmDemo2_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load

        Try
            m_spell2.ServerInstance=2
            m_spell2.Initialize()
            m_spell2.Connect(6)
            m_spell2.Project = "
c:\\EpsonRC70\\Projects\\Demo2\\Demo2.sprj "
        Catch ex As SpelException
            MsgBox(ex.Message)
        End Try
    End Sub

    Private Sub m_spell2_EventReceived(ByVal sender As Object, ByVal e As
        SpelEventArgs) _Handles m_spell2.EventReceived
        ' for Robot Controller 2
    End Sub
    Private Sub frmDemo2_FormClosed(ByVal sender As System.Object, _
        ByVal e As

```



```

System.Windows.Forms.FormClosedEventArgs) _
        Handles MyBase.FormClosed
    m_spell2.Dispose()
End Sub

```

### 18.3.4 Sample program using Visual C#

- (1) Select menu-[File]-[New]-[Project] in Visual Studio .NET.
- (2) Create a Visual C# project.
- (3) Select menu-[Project]-[Add Reference].
- (4) Select the [Browse] tab, reference “\EpsonRC70\Exe” directory, and then select the “RCAPINet.dll” file.
- (5) Add two buttons (btnController1, btnController2) to the Form1 class.
- (6) Add quick events of each button and create the thread to control each Robot Controller.

```

private System.Threading.Thread trd1; // for robot controller1
private System.Threading.Thread trd2; // for robot controller2

private void btnController1_Click(object sender, EventArgs e)
{
    // Start thread for robot controller 1
    trd1 = new System.Threading.Thread(new _
        System.Threading.ThreadStart(StartController1));
    trd1.Start();
}
private void StartController1()
{
    // Control robot controller 1
    try
    {
        frmDemo1 frm1 = new frmDemo1();
        frm1.ShowDialog();
        frm1.Dispose();
    }
    catch (System.Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
private void btnController2_Click(object sender, EventArgs e)
{
    // Start thread for robot controller 2
    trd2 = new System.Threading.Thread(new _
        System.Threading.ThreadStart(StartController2));
    trd2.Start();
}
private void StartController2()
{
    // Control robot controller 2
    try
    {
        frmDemo2 frm2 = new frmDemo2();
        frm2.ShowDialog();
        frm2.Dispose();
    }
    catch (System.Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

```

- (7) Add a form (frmDemo1) for Controller 1.

```

private Spell m_spell1;
private void frmDemo1_Load(object sender, EventArgs e)
{
    m_spell1 = new Spell();
}

```

```

try
{
    m_spell1.ServerInstance = 1;
    m_spell1.Initialize();
    m_spell1.Connect(5);
    m_spell1.Project = "c:\\EpsonRC70\\Projects\\Demo1\\Demo1.sprj";
    m_spell1.EventReceived += new _
        Spel.EventReceivedEventHandler(m_spell1_EventReceived);
}
catch (SpelException ex)
{
    MessageBox.Show(ex.Message);
}
}
public void m_spell1_EventReceived(object sender, SpelEventArgs e)
{
    // for robot controller 1
}
private void frmDemo1_FormClosed(object sender, FormClosedEventArgs e)
{
    m_spell1.Dispose();
}

```

(8) Add a form (frmDemo2) for Controller 2.

```

private Spel m_spell2;
private void frmDemo2_Load(object sender, EventArgs e)
{
    m_spell2 = new Spel();
    try
    {
        m_spell2.ServerInstance = 2;
        m_spell2.Initialize();
        m_spell2.Connect(6);
        m_spell2.Project =
"c:\\EpsonRC70\\Projects\\Demo2\\Demo2.sprj";
        m_spell2.EventReceived += new _
            Spel.EventReceivedEventHandler(m_spell2_EventReceived);
    }
    catch (SpelException ex)
    {
        MessageBox.Show(ex.Message);
    }
}
public void m_spell2_EventReceived(object sender, SpelEventArgs e)
{
    // for robot controller 2
}
private void frmDemo2_FormClosed(object sender, FormClosedEventArgs
e)
{
    m_spell2.Dispose();
}

```