EPSON

EPSON RC+ 5.0 Option

VB Guide 5.0

Rev.7 EM135S2521F

EPSON RC+ 5.0 Option　VB Guide 5.0　Rev.7

EPSON RC+ 5.0 Option

VB Guide 5.0

Rev.7

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.

WARRANTY

The robot 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.

Product malfunctions resulting from normal handling or operation will be repaired free of charge during the normal warranty period. (Please ask your Regional Sales Office for warranty period information.)

However, customers will be charged for repairs in the following cases (even if they occur during the warranty period):

1. Damage or malfunction caused by improper use which is not described in the manual, or careless use.
2. Malfunctions caused by customers' unauthorized disassembly.
3. Damage due to improper adjustments or unauthorized repair attempts.
4. Damage caused by natural disasters such as earthquake, flood, etc.

Warnings, Cautions, Usage:

1. If the robot or associated equipment is used outside of the usage conditions and product specifications described in the manuals, this warranty is void.
2. If you do not follow the WARNINGS and CAUTIONS in this manual, we cannot be responsible for any malfunction or accident, even if the result is injury or death.
3. We cannot foresee all possible dangers and consequences. Therefore, this manual cannot warn the user of all possible hazards.

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® XP Operating system
Microsoft® Windows® Vista Operating system
Throughout this manual, Windows XP, and Windows Vista refer to above respective operating systems. In some cases, Windows refers generically to Windows XP, and Windows Vista.

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.

INQUIRIES

Contact the following service center for robot repairs, inspections or adjustments.
If service center information is not indicated below, please contact the supplier office for your region.

Please prepare the following items before you contact us.

- -Your controller model and its serial number
- -Your manipulator model and its serial number
- -Software and its version in your robot system
- -A description of the problem

SERVICE CENTER

MANUFACTURER

SEIKO EPSON CORPORATION
Toyoshina Plant
Industrial Solutions Division
6925 Toyoshina Tazawa,
Azumino-shi, Nagano, 399-8285
JAPAN
TEL : +81-(0)263-72-1530
FAX : +81-(0)263-72-1495

SUPPLIERS

North & South America **EPSON AMERICA, INC.**
Factory Automation/Robotics
18300 Central Avenue
Carson, CA 90746
USA
TEL : +1-562-290-5900
FAX : +1-562-290-5999
E-MAIL : info@robots.epson.com

Europe **EPSON DEUTSCHLAND GmbH**
Factory Automation Division
Otto-Hahn-Str.4
D-40670 Meerbusch
Germany
TEL : +49-(0)-2159-538-1391
FAX : +49-(0)-2159-538-3170
E-MAIL : robot.infos@epson.de

China **EPSON China Co., Ltd**
Factory Automation Division
7F, Jinbao Building No. 89 Jinbao Street
Dongcheng District, Beijing,
China, 100005
TEL : +86-(0)-10-8522-1199
FAX : +86-(0)-10-8522-1120

Taiwan **EPSON Taiwan Technology & Trading Ltd.**
Factory Automation Division
14F, No.7, Song Ren Road, Taipei 110
Taiwan, ROC
TEL : +886-(0)-2-8786-6688
FAX : +886-(0)-2-8786-6677

Southeast Aisa
India

EPSON Singapore Pte Ltd.
Factory Automation System
1 HarbourFrontPlace, #03-02
HarbourFront Tower one, Singapore
098633
TEL : +65-(0)-6586-5696
FAX : +65-(0)-6271-3182

Korea

EPSON Korea Co., Ltd.
Marketing Team (Robot Business)
27F DaeSung D-Polis A, 606,
Seobusaet-gil, Geumcheon-gu, Seoul, 153-803
Korea
TEL : +82-(0)-2-3420-6692
FAX : +82-(0)-2-558-4271

Japan

EPSON SALES JAPAN CORPORATION
Factory Automation Systems Department
Nishi-Shinjuku Mitsui Bldg.6-24-1
Nishishinjuku. Shinjuku-ku. Tokyo. 160-8324
JAPAN
TEL : +81-(0)3-5321-4161

TABLE OF CONTENTS

1. Introduction

The EPSON RC+ VB Guide 5.0 Option 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 VB.

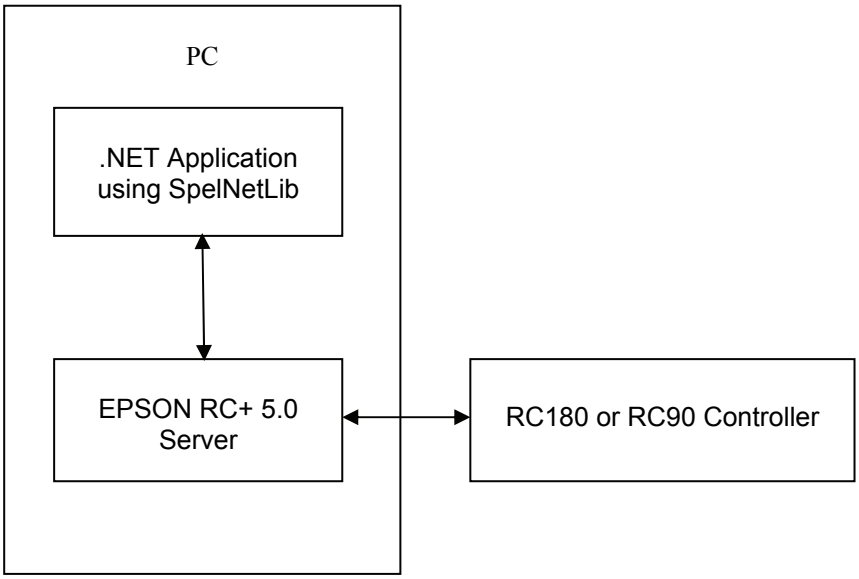
1.1 Features

The following features are supported in the VB Guide 5.0 package:

- SpelNetLib.dll (32-bit) and SpelNetLib_x64.dll (64-bit) .NET class libraries.
- Several EPSON RC+ windows and dialogs can be used by your VB application, including:
  - Robot Manager
  - Command window
  - IO monitor
  - Task manager
  - Controller Tools dialog
  - System configuration

During development, EPSON RC+ 5.0 can be run along with Visual Basic. At production time, EPSON RC+ 5.0 can be run invisibly in the background.

The figure below shows the basic structure of a system using VB Guide 5.0.



VB Guide 5.0 Basic Structure

By default, EPSON RC+ 5.0 is an in-process server for the SpelNetLib library. Each instance of SpelNetLib loads an instance of EPSON RC+ 5.0 into the user application. You can also use EPSON RC+ 5.0 as on out-of-process server. See the ServerOutOfProcess property. For the 64-bit library, EPSON RC+ 5.0 always runs out-of-process, and the ServerOutOfProcess property is not used.

2. Installation

Please follow the instructions in this chapter to help ensure proper installation of the VB Guide 5.0 software.

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

2.1 Step by step instructions

1. Install one of the Visual Studio 2008 or greater Express versions, such as Visual Basic Express, or install Visual Studio 2008 .NET or greater.
2. Install EPSON RC+ 5.0.
3. Ensure that the software key has been enabled for VB Guide 5.0 in the controller(s) you will be connecting to. Refer to the EPSON RC+ 5.0 User's Guide for information on how to enable options in the controller.

This completes the VB Guide 5.0 installation.

2.2 What's installed

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

File	Description
\EPSONRC50\VBGUIDE\VS2008\VB\DEMOS \EPSONRC50\VBGUIDE\VS2010\VB\DEMOS \EPSONRC50\VBGUIDE\VS2012\VB\DEMOS	Visual Basic .NET sample projects
\EPSONRC50\VBGUIDE\VS2008\VCS\DEMOS \EPSONRC50\VBGUIDE\VS2010\VCS\DEMOS \EPSONRC50\VBGUIDE\VS2012\VCS\DEMOS	Visual C# .NET sample projects
\EPSONRC50\VBGUIDE\VS2008\VC\DEMOS \EPSONRC50\VBGUIDE\VS2010\VC\DEMOS \EPSONRC50\VBGUIDE\VS2012\VC\DEMOS	Visual C++ .NET sample projects
\EPSONRC50\PROJECTS\VBGuideDemos	EPSON RC+ 5.0 projects for samples
\EPSONRC50\EXE\SpelNetLib.dll	SpelNetLib 32-bit Class library
\EPSONRC50\EXE\SpelNetLib_x64.dll	SpelNetLib 64-bit Class library

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 VB Guide 5.0. It is recommended that you go through the demonstrations to get familiar with the product.

For LabVIEW users, refer to *13. Using with LabVIEW*.

3.1 Getting started using Visual Basic

To use SpelNetLib in a VB.NET project, declare a Spel Class object, 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.
3. From the Project menu, select Add Reference.
4. In the NET Components tab, browse to the \EpsonRC50\Exe directory and select the SpelNetLib.dll file if your application is 32-bit, or the SpelNetLib_x64.dll file if your application is 64-bit.
5. From the Project menu, create a new module and add the following code.

```
Module Module1
  Public WithEvents g_spel As SpelNetLib.Spel
  Public Sub InitApp()

    g_spel = New SpelNetLib.Spel
    With g_spel
      .Initialize
      .Project = "c:\EpsonRC50\projects\vbnet\vbnet.sprj"
    End With
  End Sub

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

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

NOTE

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 C# project.
3. From the Project menu, select Add Reference.
4. Select the Browse tab and browse to the \EpsonRC50\Exe directory and select the SpelNetLib.dll file if your application is 32-bit, or the SpelNetLib_x64.dll file if your application is 64-bit.
5. In the Form1 class, declare a Spel variable as shown below.

```
private SpelNetLib.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 SpelNetLib.Spel();
    m_spel.Initialize();

    m_spel.Project =
"c:\\EPSONRC50\\projects\\vcsnet\\vcsnet.sprj";

m_spel.EventReceived += new
   SpelNetLib.Spel.EventReceivedEventHandler(m_spel_EventR
eceived);
```

7. Add the event handler, as shown below.

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

NOTE

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 File | Project.
2. Create a C++ CLR Windows Forms Application project.
3. From the Project menu, select References
4. Click the Add New Reference button.
5. Select the Browse tab and browse to the \EpsonRC50\Exe directory and select the SpelNetLib.dll file if your application is 32-bit, or the SpelNetLib_x64.dll file if your application is 64-bit.
6. In the Form1 class, declare a Spel variable as shown below.

```
private SpelNetLib::Spel^ m_spel;
```

7. 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 SpelNetLib::Spel();
    m_spel->Initialize();
    m_spel->Project =
        "c:\\EPSONRC50\\projects\\vcnet\\vcnet.sprj";
    m_spel->EventReceived += gcnew
        SpelNetLib::Spel::EventReceivedEventHandler(
        this, &Form1::m_spel_EventReceived);
}
```

7. Add the event handler, as shown below.

```
private System::Void m_spel_EventReceived(
    System::Object^ sender, SpelNetLib::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;
```

4. Environments

4.1 Design-Time 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 VB project.
2. Start EPSON RC+ 5.0.
3. Configure EPSON RC+ 5.0 to communicate with one or more robot controllers.
4. Open the desired EPSON RC+ project or create a new EPSON RC+ project.
5. Build the EPSON RC+ project.
6. Add initialization code in VB for the SPEL class instance.
7. Run and debug the VB project.

4.1.2 Spel Class Instance Initialization

After a new instance of the Spel class has been created, it needs to be initalized. When initialization occurs, the underlying EPSON RC+ 5.0 modules are loaded and initialized. Initialization is implicit with the first method call or property access. You can explicity initialize the class by calling the Initialize method.

```
m_spel.Initialize
```

If you want to use the project in the controller which is not on your PC, then you must set the NoProjectSync property to true.

```
m_spel.NoProjectSync = True
```

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 VB anc C#, use the Dispose method:

```
m_spel.Dispose()
```

For VC++, 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 VB code:

1. Stop the VB project.
2. Open EPSON RC+ 5.0.
3. Make changes in the EPSON RC+ 5.0 project.
4. Build the EPSON RC+ 5.0 project.
5. Switch to VB.
6. Run the VB project.

4.2 Production Environment

4.2.1 Opening EPSON RC+ 5.0 at Runtime

Decide if you want to allow the EPSON RC+ 5.0 environment to be opened from your application. This is especially useful for debugging. Set the **OperationMode** property to Program to put RC+ in Program Mode and open the RC+ GUI.

4.2.2 Using EPSON RC+ 5.0 Dialogs and Windows

At runtime, you can open and hide certain EPSON RC+ 5.0 windows from your VB application. You can also run certain EPSON RC+ 5.0 dialogs. See the chapter *EPSON RC+ Windows and Dialogs* for details.

4.2.3 Installation on Target System

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

1. Install EPSON RC+ 5.0.
2. Install your EPSON RC+ 5.0 project. This is not needed if you plan to only run the project in the controller.
3. Install your VB application.

5. Executing Programs

5.1 Executing SPEL+ Programs

You can run any of the eight builtin main functions in the current controller project by using the *Start* method of the Spel class. 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
4	main4
5	main5
6	main6
7	main7

Here is an example that starts function "main":

```
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
```

You can also execute functions in the project using the Xqt method.

Note that when starting using the Start method, global variables are cleared before the function runs. If a function is started with Xqt, the global variables are not cleared.

5.2 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.

For 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
```

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 VB application.

6.2 System Events

There are several system events that are sent to the VB 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 *EventReceived*.

6.3 User Events from SPEL+

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

To fire an event to VB 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 VB. See EPSON RC+ 5.0 Help for details on SPELCom_Event.

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

```
Public WithEvents g_spel As SpelNetLib.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 EventReceived from the procedure list.

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

```
Sub m_spel_EventReceived (ByVal sender As Object, _
            ByVal e As SpelNetLib.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
```

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, Spel 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. For 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 SpelNetLib.SpelException
    MsgBox(ex.Message)
  End Try
End Sub
```

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

```
  Try
    m_spel.Start(0)
  Catch ex As SpelNetLib.SpelException
    MsgBox(ex.ErrorNumber)
  End Try
```

8. Handling Pause and Continue

8.1 Pause state

When a pause occurs, SPEL+ 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+ task executed Pause.

- The safeguard was opened.

8.2 Catching the Pause event

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

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

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

8.3 Executing Pause

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

```
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
```

8.4 Continue after pause

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

```
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
```

8.5 Abort after pause

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

```
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
```

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 you VB application.

```
Imports SpelNetLib.Spel

Private Sub m_spel_EventReceived(ByVal sender As Object, ByVal
e As SpelEventArgs) Handles m_spel.EventReceived
    Select Case EventNumber
        Case SpelEvents.EstopOn
            MsgBox "E-Stop detected"
            gEStop = True
            lblEStop.BackColor = vbRed
            lblEStop.Text = "EStop ON"
        Case SpelEvents.EstopOn
            gEStop = False
            lblEStop.BackColor = vbGreen
            lblEStop.Text = "EStop OFF"
    End Select
End Sub
```

10. EPSON RC+ Windows and Dialogs

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

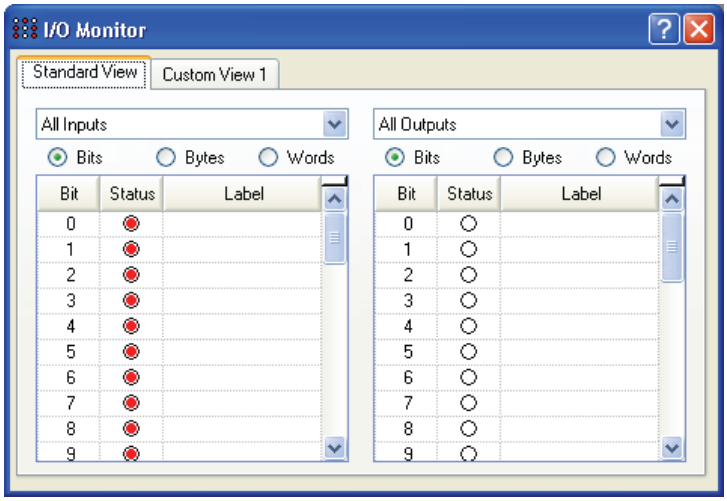
10.1 Windows

Windows are non-modal, meaning that they can remain open while other elements of your VB GUI can be used. You can show and hide EPSON RC+ 5.0 windows from your VB program.

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

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

WindowID	Window
SpelWindows.IOMonitor	IO Monitor
SpelWindows.TaskManager	Task Manager
SpelWindows.Simulator	Simulator



I/O Monitor Window

For 64-bit applications, EPSON RC+ 5.0 runs out-of-process, so you should supply a parent window handle using the ParentWindowHandle property, then call ShowWindow without the parent argument.

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

10.2 Dialogs

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

For example, to open the Robot Manager dialog:

```
m_spel.RunDialog(SpelNetLib.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
SpelDialogs.RobotManager	Robot Manager
SpelDialogs.ControllerTools	Controller Tools

For 64-bit applications, EPSON RC+ 5.0 runs out-of-process, so you should supply a parent window handle using the ParentWindowHandle property, then call RunDialog.

```
m_spel.ParentWindowHandle = Me.Handle
m_spel.RunDialog(SpelNetLib.SpelDialogs.RobotManager)
```

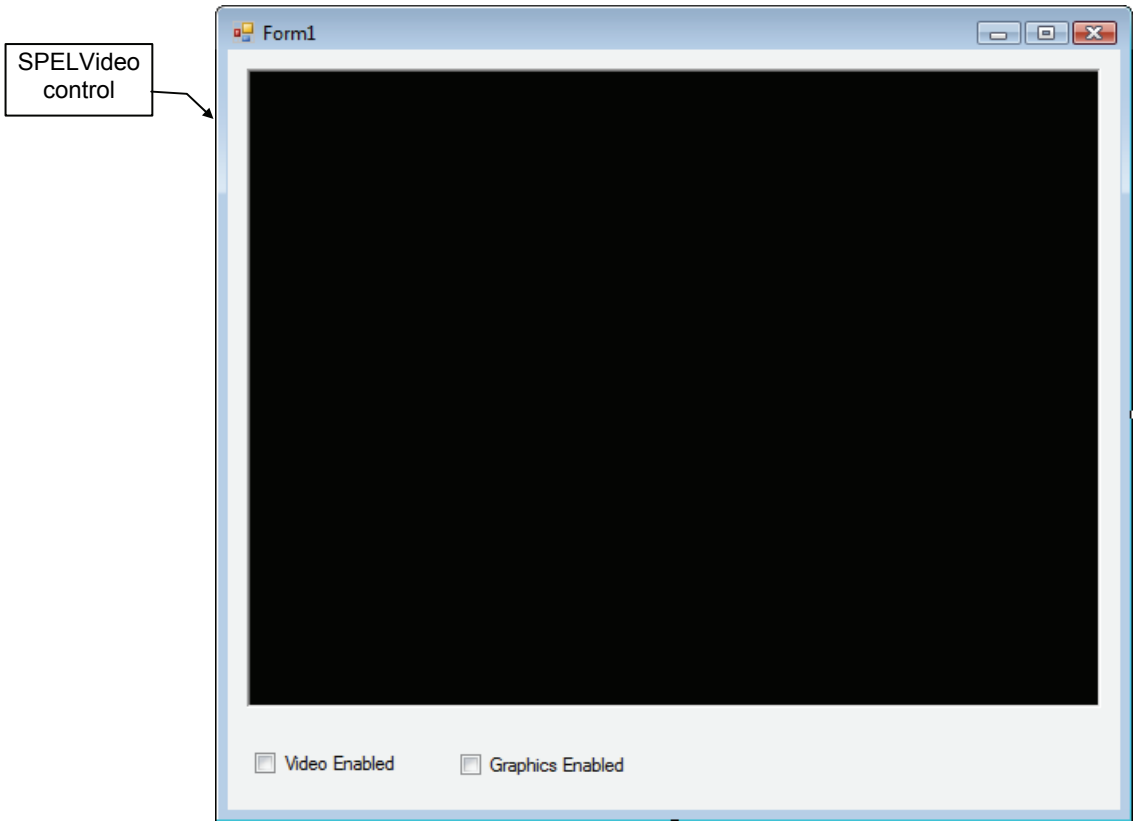
11. Displaying Video

You can easily display a live vision window 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.

Note: For 64-bit applications, you cannot use the SPELVideo control at design time. You need to create the control at runtime. See the section below *Creating a SPELVideo control at runtime*.

Perform the following steps to create a vision window for a 32-bit application:

1. Add the SPELVideo component to your project. To add the control to your VB toolbox, right click on the toolbox and select Choose Items. Select the Browse tab and browse to the \EpsonRC50\Exe directory and select the SpelNetLib.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.

If you use more than one SpelVideo control in your application, then you should disable the video on the non-active controls. There can only be one vision window displayed at one time.

Here is an example showing how to enable video and graphics on a VB 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:\epsonrc50\projects\test\test.sprj"
    SpelVideo1.VideoEnabled = True
    SpelVideo1.GraphicsEnabled = True
    m_spel.SpelVideoControl = SPELVideo1
End Sub
```

Creating a SPELVideo control at runtime

Here is an example showing how to create a SPELVideo control at runtime. This is necessary for 64-bit applications, because Visual Studio 2008 and 2010 are 32-bit and cannot use 64-bit controls at design time.

```
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:\epsonrc50\projects\test\test.sprj"
    m_spelVideo = New SPELVideo
    m_spelVideo.Parent = Me
    m_spelVideo.Left = 10
    m_spelVideo.Top = 10
    m_spelVideo.Width = 320
    m_spelVideo.Height = 240
    m_spelVideo.Visible = True
    m_spelVideo.VideoEnabled = True
    m_spelVideo.GraphicsEnabled = True
    m_spel.SpelVideoControl = m_spelVideo
End Sub
```

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
Curve	Mcal
CVMove	Move
ExecuteCommand	PTran
Go	Pulse
Home	TGo
JTran	TMove
Jump	

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

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

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

13. SpelNetLib Reference

13.1 Spel Class

Description

This class allows you to execute commands and receive events from EPSON RC+ 5.0.

File Name

SpelNetLib.dll (32-bit)

SpelNetLib_x64.dll (64-bit)

13.2 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

WaitCommandComplete

AsyncMode Example

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

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

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

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

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

ErrorOn Property, Spel Class

Description

Returns True if a critical error has occured 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

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

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

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

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 persistant. You must set it after creating a Spel class instance if you want to set it to True.

See Also

Start

Examples

```
m_spel.Initialize
m_spel.NoProjectSync = True
```

OperationMode Property, Spel Class

Description

Reads or sets the EPSON RC+ mode of operation.

Syntax

Property **OperationMode** As SpelOperationMode

Return value

SpelOperationMode value

Remarks

When **OperationMode** is set to Program, the EPSON RC+ 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 VB, the GUI also closes.

OperationMode Example

```
Sub btnSpelProgramMode_Click _
       ByVal sender As System.Object, _
       ByVal e As System.EventArgs) _
       Handles btnHideIOMonitor.Click

  Try
    m_spel.OperationMode = _
                  SpelNetLib.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 = _
                   SpelNetLib.SpelOperationMode.Auto
  Catch ex As SpelNetLib.SpelException
    MsgBox(ex.Message)
  End Try
End If
```

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 when ServerOutOfProcess is set to true. This allows you to specify the parent window from applications that do not have .NET forms, such as LabVIEW.

See Also

ServerOutOfProcess

ParentWindowHandle Example

```
m_spel.ParentWindowHandle = Me.Handle
m_spel.ShowWindow(SpelNetLib.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

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

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

Project Property, Spel Class

Description

Sets / returns the current EPSON RC+ 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+ project make file. The make file is the project name with a .SPRJ extension.

Project Example
```
m_spel.Project = "c:\EpsonRC50\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
```
If m_spel.ProjectBuildComplete Then
    lblBuild.Text = "Project build is Complete"
Else
    lblBuild.Text = "Project build is not Complete"
End If
```

ResetAbortEnabled Property, Spel Class

Description
Sets / returns whether ResetAbort method should be enable 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
```
' Enable reset abort
m_spel.ResetAbortEnabled = 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
```
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
```
Select Case m_spel.RobotType
  Case SpelNetLib.SpelRobotType.Scara
    lblRobotType.Text = "Scara"
  Case SpelNetLib.SpelRobotType.Cartesian
    lblRobotType.Text = "Cartesian"
End Select
```

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

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

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

```
m_spel.SpelVideoControl = SpelVideo1
```

ServerOutOfProcess Property, Spel Class

Description

Sets / returns whether the EPSON RC+ 5.0 should run in-process or out-of-process.

Syntax

ServerOutOfProcess

Type

Boolean

Default Value

False

Remarks

By default, EPSON RC+ 5.0 is used as an in-process server, which means that RC+ is in the same process as your application. You can configure RC+ to run out-of-process using this property. In-process communication is slightly faster than out-of-process communication. Set this property to True for applications that cannot load RC+ in the application process at runtime, such as LabVIEW running in standard execution mode.

You must set ServerOutOfProcess before using any other properties or methods.

Note: This property is only supported in the 32-bit version of SpelNetLib. It is not required in the 64-bit version, where EPSON RC+ must always run out-of-process.

See Also

DisableMsgDispatch, ParentWindowHandle

Version Property, Spel Class

Description

Returns the current EPSON RC+ software version.

Syntax

ReadOnly Property **Version** As String

Return Value

String that contains the current EPSON RC+ 5.0 software version.

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

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

13.3 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

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

```
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 in deg/sec^2 units 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 in deg/sec^2 units to define tool rotation deceleration when ROT is used in motion commands.

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

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

AccelS Method, Spel Class

Description
Sets acceleration and deceleration for linear interpolar (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

Accel	Single expression between 1-5000 represented in mm/sec^2 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.
Decel	Single expression between 1-5000 represented in mm/sec^2 units to define deceleration values for Straight Line and Continuous Path motion. One parameter is used for representing both the Acceleration and Deceleration rates.
JumpDepartAccel	Single expression between 1-5000 representing a percentage of maximum acceleration rate for Jump3CP command Z Axis upward motion.
JumpDepartDecel	Single expression between 1-5000 representing a percentage of maximum deceleration rate for Jump3CP command Z Axis upward motion.
JumpApproAccel	Single expression between 1-5000 representing a percentage of maximum acceleration rate for Jump3CP command Z Axis downward motion.
JumpApproDecel	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
```
Sub MoveToPlace()
    m_spel.AccelS(500)
    m_spel.Move(pick)
    m_spel.AccelS(500, 300)
    m_spel.Move(place)
End Sub
```

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

ArchNumber The Arch number to define. Valid Arch numbers are (0-6) making a total of 7 entries into the Arch table.

DepartDist The depart distance in millimeters moved at the beginning of the Jump instruction before starting horizontal motion.

ApproDist The approach distance in millimeters above the target position of the Jump instruction.

See Also
Jump, Jump3, Jump3CP

Arch Example
```
Sub SetArchs()
    With m_spel
        .Arch(1, 30, 30)
        .Arch(2, 60, 60)
    End With
End Sub
```

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
```
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
```
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

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

ArmSet Method, Spel Class

Description
Specifies auxiliary robot arms.

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

Parameters

ArmNumber Integer number: Valid range from 0-3. The user may select up to 4 different arms. Arm 0 is the standard (default) robot arm. Arm(s) 1-3 are auxiliary arms defined by using the ArmSet instruction.

Param1 (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.

Param2 (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 joint2Offset.)
(For Cartesian Robots) Y axis direction position offset from the original Y position specified in mm.

Param3 (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.)

Param4 (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)

Param5 (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
```
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
```

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.

See Also
Atan2

Atan Example
```
Dim angle As Double

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
```
Dim angle As Double

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

AtHome Method, Spel Class

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

Syntax
Funciton **AtHome** () As Boolean

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

See Also
Home

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

AxisLocked Method, Spel Class

Description
Returns True if specified axis is under servo control.

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

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

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

See Also
Local

Base Example
```
Dim originPoint As 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** (*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.
PointExpr	Specifies the end point by using a string expression.

See Also
Go, TGo

BGo 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 a point expression
m_spel.BGo("P0 /L /2")
```

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

See Also

Move, TMove

BMove Example

```
m_spel.Tool(1)
m_spel.BMove(100)
```

Box Method, Spel Class

Description
Specifies and displays the approach check area.

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

Parameters

AreaNumber Integer number from 1-15 representing which of 15 boxes to define.

MinX The minimum X coordinate position which can be set to the approach check area.

MaxX The maximum X coordinate position which can be set to the approach check area.

MinY The minimum Y coordinate position which can be set to the approach check area.

MaxY The maximum Y coordinate position which can be set to the approach check area.

MinZ The minimum Z coordinate position which can be set to the approach check area.

MaxZ The maximum Z coordinate position which can be set to the approach check area.

See Also
BoxClr, BoxDef, Plane

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

BoxDef Method, Spel Class

Description
Returns whether Box has been defined or not.

Syntax
Funciton **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
```
x = m_spel.BoxDef(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
```
x = m_spel.BTst(data, 2)
```

BuildProject Method, Spel Class

Description
Builds the EPSON RC+ project specifed by the Project property.

Syntax
Sub **BuildProject** ()

See Also
Project, ProjectBuildComplete

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

Call Method, Spel Class

Description
Calls (executes) a SPEL+ 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 EPSON RC+ project.
Parameters	Optional. A string expression containing the parameters for the call.

Return Value
The return value of the SPEL+ function. The data type matches the the data type of the function.

Remarks
Use the Call method to call a SPEL+ 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+ code from your VB application.

See Also
Project, Xqt

Call Example
```
'VB Code
Dim errCode As Integer
errCode = m_spel.Call("GetPart")

'SPEL+ function
Function GetPart As Integer
    Long errNum
    OnErr GPErr
    errNum = 0
    Jump P1
    On vacuum
    Wait SW(vacOn) = 1, 2
    If TW(0) = 1 Then
        errNum = VAC_TIMEOUT
    EndIf
GPExit:
    GetPart = errNum
    Exit Function
GPErr:
    errNum = Err
    GoTo GPExit
Fend
```

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

Connect Method, Spel Class

Description
Connects the Spel class instance with a controller.

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

Parameters

ConnectionNumber Integer expression for the connection number.
This currently must be set to 1.

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

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
```
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 SpelNetLib.SpelException
    MsgBox(ex.Message)
  End Try
End Sub
```

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.

See Also
CtReset

Ctr 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

```
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.

For more information, see *Curve Statement* in the Spel+ Language Reference manual.

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. It can also specify the ECP number in the upper four bits.

Mode Setting		Tangential Correction	ECP Number
Hexadecimal	Decimal		
&H00	0	No	0
&H10	16		1
&H20	32		2
…	…		…
&HA0	160		10
&HB0	176		11
&HC0	192		12
&HD0	208		13
&HE0	224		14
&HF0	240		15
&H02	2	Yes	0
&H12	18		1
&H22	34		2
…	…		…
&HA2	162		10
&HB2	178		11
&HC2	194		12
&HD2	210		13
&HE2	226		14
&HF2	242		15

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)

6 - Generate a curve in the XYZ space with U, V, and W axes rotation (6-Axis robots only).

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, O, 0, 4, P1, P2, P3, P4
```

Remarks

Use Curve to define a spline path to be executed with the CVMove method. For more information, see *Curve Statement* in the Spel+ Language Reference manual.

See Also

Curve (SPEL$^+$ Statement), CVMove Method

Curve 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 and stored on a PC hard disk.

OptionList Optional. String expression containing Till specification.

Remarks
Use CVMove to exeute a path defined with the Curve method. See the SPEL$^+$ command **CVMove** for more details.
If you need to execute CVMove with CP, it is recommended that you execute CVMove from a SPEL$^+$ task rather than from VB Guide. 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 VB Guide 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$^+$ Command)

CVMove 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 Methods, Spel Class

Description
Retrieves a coordinate value from a point

CV and CW are for the 6-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

Parameters

PointExpr A string expression specifying the point from which to retrieve the specified coordinate. Any valid point expresion 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)

See Also
GetPoint, SetPoint

CX, CY, CZ, CU, CV, CW Example
```
Dim x As Single, y As Single
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
```
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
```
Dim rad As Double

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

ECP Method, Spel Class

Description
Selects an 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
```
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
```
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
```
x = m_spel.ECPDef(1)
```

ECPSet Method, Spel Class

Description
Defines an ECP (external control point).

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

Parameters

ECPNumber	Integer number from 1-15 representing which of 15 external control points to define.
XCoord	The external control point X coordinate.
YCoord	The external control point Y coordinate.
ZCoord	The external control point Z coordinate.
UCoord	The external control point U coordinate.
VCoord	Optional. The external control point V coordinate.
WCoord	Optional. The external control point W coordinate.

See Also
ArmSet, ECP, GetECP, TLSet

ECPSet 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** (*Event* As 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
```
With m_spel
  .EnableEvent(SpelNetLib.SpelEvents.ProjectBuildStatus,
True)
  .BuildProject()
End With
```

ExecuteCommand Method, Spel Class

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

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

Parameters

Command String containing SPEL+ 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+ 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
```
m_spel.ExecuteCommand("JUMP P1!D50; ON 1!")
```

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

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

```
Dim x As Integer
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

```
saveArm = m_spel.GetArm()
m_spel.Arm(2)
```

GetControllerInfo Method, Spel Class

Description
Returns information about the current controller.

Syntax
Function **GetControllerInfo**() As SpelControllerInfo

Return Value

A SpelControllerInfo instance.

See Also
GetErrorMessage

Remarks

GetControllerInfo returns a new instance of the SpelControllerInfo class, which contains controller information properties.

GetControllerInfo 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)
```

GetECP Method, Spel Class

Description
Returns the current ECP number.

Syntax
Function **GetECP** () As Integer

Return Value

Integer containing the current ECP number.

See Also
ECP, ECPSet

GetECP 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

```
Dim msg As String

If m_spel.ErrorOn Then
  msg = m_spel.GetErrorMessage(m_spel.ErrorCode)
  MsgBox(msg)
End If
```

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

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
Index	Specifies the bit or port number.
Label	Returns the label.
Description	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

```
Dim label As String
Dim desc As String
m_spel.GetIODef(SpelIOLabelTypes.InputBit, 0, label, desc)
```

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

```
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
```
Dim pt As SpelPoint
pt = m_spel.GetPoint("P*")
pt.X = 25.0
m_spel.Go(pt)
```

GetRealTorque Method, Spel Class

Description
Returns the torque for the specified joint.

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 proportion of maximum torque for the current power mode and for the specified joint.

See Also
GetRobotPos

GetRealTorque Example
```
Dim j1Torque As Double
j1Torque = m_spel.GetRealTorque(1)
```

GetRobotPos Method, Spel Class

Description
Returns the current actual position of the robot.

Syntax
Function **GetRobotPos** (*PosType* As SpelRobotPosType, Local As Integer) As Single()

Parameters

PosType	Specifies the data to retrieve. You can specify World, Joint, or Pulse.
Local	Integer expression that specifies the local coordinate system to use when retrieving World coordinates.

Return Value

Single array of six elements containing the position data.

See Also
GetRealTorque

GetRobotPos Example

```
Dim posData As Single(6)
posData = m_spel.GetRobotPos(SpelRobotPosType.World, 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

See Also
Speed

GetSpeed Example

```
Dim x As Integer
x = m_spel.GetSpeed(1)
```

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
```
saveTool = m_spel.GetTool()
m_spel.Tool(2)
```

GetVar Method, Spel Class

Description
Returns the value of a SPEL+ global preserve variable in the controller.

Syntax
Function **GetVar**(*VarName* As String) As Object

Parameters

VarName The name of the SPEL+ 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+ 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 sucessfully 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+ project, the variable is declared:

```
Global Preserve Integer g_myIntVar
Global Preserve Real g_myRealArray(10)
Global Preserve String g_myStringVar$
Function main
   ...
Fend
```

In the VB project:

Since g_myIntVar is declared as in integer, the VB variable used to retrieve the value of g_myInVar must be declared as an Integer. For g_myRealArray, the VB 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$")
```

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** (*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.

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

See Also
Jump, Move, TGo

Go Example
```
m_spel.Go(1)
m_spel.Go("P1 :Z-20")
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.

See Also
Resume, Xqt

Halt 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
```
m_spel.Here("P20")
```

HideWindow Method, Spel Class

Description

Hides an EPSON RC+ window that was previously displayed with ShowWindow.

Syntax
Sub **HideWindow** (*WindowID* As SpelWindows)

Parameters

WindowID The ID of the EPSON RC+ window to hide.

See Also
RunDialog, ShowWindow

HideWindow Example
```
Sub btnHideIOMonitor_Click _
       ByVal sender As System.Object, _
       ByVal e As System.EventArgs) _
       Handles btnHideIOMonitor.Click

    m_spel.HideWindow(SpelNetLib.SpelWindows.IOMonitor)
End Sub
```

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
```
With m_spel
    .MotorsOn = True
    .Home()
End With
```

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

Parameters

J1Pulses – J9Pulses The Home position encoder pulse value for each joint.

See Also
Home, MCal

HomeSet 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
```

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

Parameters

Step 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.

See Also
Home, HomeSet, Mcordr

Hordr 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

Hour Example
```
Dim hoursRunning As Single
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

SourcePath	String expression containing the specific path and file to import into the current project. The extension must be .PTS.
ProjectFileName	String expression containing the specific file to be imported to in the current project for the current robot or specified robot if *RobotNumber* is supplied. The extension must be .PTS.
RobotNumber	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
```
With m_spel
    .ImportPoints("c:\mypoints\model1.pts", "robot1.pts")
End With
```

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.

See Also
InBCD, Out, OpBCD, Sw

In Example
```
Dim port1Value As Integer
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.

See Also
In, Out, OpBCD, Sw

InBCD Example

```
Dim port1Value As Integer
port1Value = m_spel.InBCD(1)
```

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+ loads into memory. So in some cases, you may want to call initialize first in your application during startup.

See Also
Connect, Disconnect

Initialize Example

```
m_spel.Initiialize()
```

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
```
x = 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
```
x = 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
```
Dim data As Integer
data = m_spel.InW(0)
```

JRange Method, Spel Class

Description
Defines the permissible working range of the specified axis in pulses.

Syntax
Sub **JRange** (*JointNumber* As Integer, *LowerLimitPulses* As Integer, *UpperLimitPulses* As Integer)

Parameters

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

See Also
XYLim

JRange 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, Sense

JS Example
```
With m_spel
    .Sense("Sw(1) = On")
    .Jump("P1 SENSE")
    stoppedOnSense = .JS()
End With
```

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

See Also
Go, Move, TGo

Jump Example
```
Const pick = 1

m_spel.Jump(1)
m_spel.Jump("P1 :Z-20")
m_spel.Jump("pick")
```

Jump3 Method, Spel Class

Description
Motion with 3D gate using a combination of two CP motions and one PTP motion.

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

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
Go, Jump, Jump3CP

Jump3 Example
```
m_spel.Jump3(1, 2, 3)
```

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
Go, Jump, Jump3

Jump3CP Example
```
m_spel.Jump3CP(1, 2, 3)
```

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
```
saveLimZ = m_spel.GetLimZ()
m_spel.LimZ(-22)
```

LoadPoints Method, Spel Class

Description
Loads a SPEL$^+$ point file into the controller's point memory for the current robot.

Syntax
Sub **LoadPoints** (*FileName* As String)

Parameters

FileName	A valid point file in the current project.

See Also
ImportPoints, SavePoints

LoadPoints Example
```
With m_spel
    .LoadPoints("part1.pts")
End With
```

Local Method, Spel Class

Description
Defines local coordinate systems.

Syntax

Sub **Local** (*LocalNumber* As Integer, *OriginPoint* As SpelPoint, [*XAxisPoint* As SpelPoint], [*YAxisPoint* As SpelPoint])

Parameters

LocalNumber	The local coordinate system number. A total of 15 local coordinate systems (of the integer value from 1 to 15) may be defined.
OriginPoint	SpelPoint variable for the origin of the local coordinate system.
XAxisPoint	Optional. SpelPoint variable for a point along the X axis of the local corrdinate system.
YAxisPoint	Optional. SpelPoint variable for a point along the Y axis of the local corrdinate system.

See Also
Base

Local Example
```
Dim originPoint As 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
```
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 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
```
Dim localExists As Boolean
localExists = m_spel.LocalDef(1)
```

MemIn Method, Spel Class

Description
Returns the status of the specified memory I/O byte port. Each port contains 8 memory I/O its.

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
```
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 containing the port value.

See Also
In, InBCD, MemIn, MemSw, Sw, Off, On, Oport

MemInW 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
```
m_spel.MemOff(500)
```

MemOn Method, Spel Class

Description
Turns On the specified bit of the S/W 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
```
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
```
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 (integers from 0 to 65535) using an expression or numeric value.

See Also
In, InBCD, MemIn, MemSw, Sw, Off, On, Oport

MemOutW 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
```
If m_spel.MemSw(10) Then
    m_spel.On(2)
End If
```

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

See Also
AccelS, Go, Jump, SpeedS, TGo

Move Example
```
m_spel.Move(1)
m_spel.Move("P1 :Z-20")
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
```
m_spel.Off(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. This tells the **On** instruction which output to turn on

Label String expression containing an output bit label.

See Also
Off, Oport, Out, OutW

On 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
```
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 and expansion discrete 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
```
If m_spel.Oport(1) Then
    m_spel.On(2)
End If
```

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 containing the port value.

See Also
InBCD, OpBCD, Oport, OutW, Sw

Out Example
```
m_spel.Out(1, 240)
```

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 containing the port value.

See Also
InBCD, OpBCD, Oport, Out, Sw

OutW 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 ~ 9.

Return Value

Single containing the angle for the specified joint in degrees or millimeters.

See Also
Agl, Pls, CX – CT

PAgl Example
```
Dim t1Angle As Single
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

PalletNumber	Pallet number represented by an integer number from 0 to 15.
Point1	Point variable which defines first pallet position.
Point2	Point variable which defines second pallet position.
Point3	Point variable which defines third pallet position.
Point4	Optional. Point variable which defines fourth pallet position.
Rows	Numbers of points on lateral side of the pallet. Each number is an integer from 1 to 32767.
Columns	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
```
m_spel.Pallet(1, 1, 2, 3, 4, 3, 4)
```

Pause Method, Spel Class

Description
Causes all SPEL+ 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
```
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
```

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
```
x = 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 that specifies the last point in range to delete. If omitted, only the pont specified in *FirstPointNumber* is deleted.

See Also
PDef, LoadPoints, Clear, SavePoints

PDel Example
```
m_spel.PDel(1, 10)
m_spel.SavePoints("model1.pts")
```

Plane Method, Spel Class

Description
Defines a Plane.

Syntax
Sub **Plane** (*PlaneNumber* As Integer, *Point* As SpelPoint)

Parameters

PlaneNumber	Integer number from 1-15 representing which of the 15 Planes to define.
Point	Point data representing the coordinate data of the approach check plane.

See Also
PlaneClr, PlaneDef

Plane 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
```
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
```
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
```
j1Pulses = m_spel.Pls(1)
```

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
```
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
```
If m_spel.PTPBoostOK(1) Then
    m_spel.Go(1)
End If
```

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
```
' Move joint 1 5000 pulses in the plus direction.
m_spel.PTran(1, 5000)
```

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
```
m_spel.Pulse(5000, 1000, 0, 0)
```

Quit Method, Spel Class

Description
Terminates execution of the task which is specified by task number.

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
```
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
```
Dim deg As Double

deg = m_spel.RadToDeg(1)
```

RebuildProject Method, Spel Class

Description
Completely rebuilds the current EPSON RC+ project specifed in the Project property.

Syntax
Sub **RebuildProject** ()

See Also
BuildProject, EnableEvent, EventReceived, Project, ProjectBuildComplete

RebuildProject Example
```
With m_spel
    .Project = "c:\epsonrc50\projects\myproject\myproject.sprj"
    .RebuildProject()
End With
```

Reset Method, Spel Class

Description
Resets the controller to the initialized state.

Syntax
Sub **Reset** ()

See Also
ResetAbort

Reset 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
```
Sub btnMcal_Click() Handles btnMcal.Click
  m_spel.ResetAbort()
  m_spel.MCal()
End Sub
```

Resume Method, Spel Class

Description
Continues 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
```
m_spel.Resume(2)
```

RunDialog Method, Spel Class

Description
Runs an EPSON RC+ dialog.

Syntax
Sub **RunDialog** (*DialogID* As SpelDialogs)

Parameters

DialogID The ID of the EPSON RC+ dialog to run.

See Also
ShowWindow

RunDialog Example

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

    m_spel.RunDialog(SpelDialogs.RobotManager)
End Sub
```

SavePoints Method, Spel Class

Description
Save points for the current robot.

Syntax
Sub **SavePoints** (*FileName* As String)

Parameters

FileName The filename to save the points in the current EPSON RC+ project.

See Also
LoadPoints

SavePoints Example

```
With m_spel
    .SavePoints("part1.pts")
End With
```

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 the Sense Statement in the SPEL+ Language Reference manual.

See Also
Jump, JS

Sense Example

```
With m_spel
    .Sense("Sw(1) = On")
    .Jump("P1 SENSE")
    stoppedOnSense = .JS()
End With
```

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 SpelLabelTypes, *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
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
```
Dim label, desc As String
label = "StartCycle"
desc = "Starts the robot cycle"
m_spel.SetIODef(SpelLabelTypes.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**(*PointNumber* As Integer, *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**(*PointNumber* As Integer, *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**(*PointNumber* As Integer, *X* As Single, *Y* As Single, *Z* As Single, *U* As Single, *V* As Single, *W* As Single)

Parameters

PointNumber	Integer expression that specifies the point number for a point in the current robot's point memory.
X	The X coordinate for the specified point.
Y	The Y coordinate for the specified point.
Z	The Z coordinate for the specified point.
U	The U coordinate for the specified point.
V	The V coordinate for the specified point.
W	The W coordinate for the specified point.
Local	The Local Number for the specified point. Use 0 when there is no local.
Hand	The hand orientiation of the specified point.
Elbow	The elbow orientiation of the specified point.
Wrist	The wrist orientiation of the specified point.

See Also
GetPoint, LoadPoints, SavePoints

SetPoint Example
```
' Get coordinates of P1
m_spel.GetPoint(1, x, y, z, u, localNum, orient)
' Set it with changes
m_spel.SetPoint(1, x, y, z - 10.5, u, localNum, orient)
```

SetVar Method, Spel Class

Description
Sets the value of a SPEL+ global preserve variable.

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

```
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))
```

SFree Method, Spel Class

Description
Frees the specified robot axes from servo control.

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 specifiy axis numbers from 1 – 6.

See Also
SLock

SFree Example

```
' Free Axes 1 & 2
m_spel.SFree(1, 2)
```

ShowWindow Method, Spel Class

Description
Shows an EPSON RC+ window.

Syntax
Sub **ShowWindow** (*WindowID* As SpelWindows)
Sub **ShowWindow** (*WindowID* As SpelWindows, *Parent* As Form)

Parameters

WindowID The ID of the EPSON RC+ window to show.

Parent Optional. The .NET parent form.

Remarks

If *Parent* is ommited, then you should set the parent window handle using the ParentWindowHandle property.

See Also
HideWindow, ParentWindowHandle, RunDialog

ShowWindow Example

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

    m_spel.ShowWindow(SpelNetLib.SpelWindows.IOMonitor, Me)
End Sub
```

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
```
' Restart Windows
m_spel.Shutdown(1)
```

SLock Method, Spel Class

Description
Returns specified axes to servo control.

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 specifiy axis numbers from 1 – 9.

See Also
SFree

SLock Example
```
' Return Axes 1 and 2 to servo control
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 or displays 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
```
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, TMove

SpeedR 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	Single expression between 1-5000 representing the Z axis upward motion speed for the Jump3CP instruction.
JumpApproSpeed	Single expression between 1-5000 representing the Z axis downward motion speed for the Jump3CP instruction.

See Also
AccelS, Jump3CP, Move, TMove

SpeedS Example
```
m_spel.SpeedS(500)
```

Start Method, Spel Class

Description
Start one SPEL+ program.

Syntax
Sub **Start** (*ProgramNumber* As Integer)

Parameters

ProgramNumber The program number to start, corresponding to the 64 built-in main functions in SPEL+ 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
…	…
7	main7

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 to be cleared and default robot points to be loaded.

See Also
Continue, Pause, Stop, Xqt

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

    m_spel.Start(0)
End Sub
```

Stop Method, Spel Class

Description

Stops all SPEL+ tasks running in the controller.

Syntax

Sub **Stop** ()

See Also

Continue, Pause, Start

Stop Example

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

    m_spel.Stop()
End Sub
```

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

```
If m_spel.Sw(1) Then
    m_spel.On(2)
End If
```

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
```
If m_spel.TargetOK("P1 /F") Then
    m_spel.Go("P1 /F")
End If
```

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
Stat, TaskState, Xqt

TasksExecuting Example
```
tasksRunning = m_spel.TasksExecuting()
```

TaskState Method, Spel Class

Description
Returns the status of a task.

Syntax
Function **TaskStatus** (*TaskNumber* As Integer) As SpelTaskState
Function **TaskStatus** (*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
Stat, TasksExecuting, Xqt

TaskStatus Example
```
Dim taskStatus As SpelTaskState
taskStatus = 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

Parameters

PointFile	A string containing the name of the point file.
PointNumber	The point number to teach.
Prompt	A string containing the instructional text that is displayed on the bottom of the teach dialog.

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 from 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

```
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
```

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 the Till Statement in the SPEL+ Language Reference manual.

See Also
Go, Jump, JS, Sense, TillOn

Till Example
```
With m_spel
    .Till("Sw(1) = On")
    .Go("P1 TILL")
End With
```

TillOn Method, Spel Class

Description
Returns True if a stop has occured 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, Stat, Till

TillOn Example
```
If m_spel.TillOn() Then
    m_spel.Jump(2)
End If
```

TGo Method, Spel Class

Description
Executes Point to Point relative motion, in the selected tool coordinate system.

Syntax
Sub **TGo** (*PointNumber* As Integer)
Sub **TGo** (*Point* As SpelPoint)
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.
PointExpr	Specifies the end point by using a string expression.

See Also
Go, Jump, Move, TMove

TGo Example
```
m_spel.Tool(1)
m_spel.TGo(100)
```

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, ToolDef

ToolClr Example
```
m_spel.ToolClr(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, ToolClr

ToolDef Example
```
m_spel.ToolDef(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

ToolNumber	Integer expression from 1-15 representing which of 15 tools to define. (Tool 0 is the default tool and cannot be modified.)
Point	A SpelPoint containing the point data.
XCoord	The tool coordinate system origin X coordinate.
YCoord	The tool coordinate system origin Y coordinate.
ZCoord	The tool coordinate system origin Z coordinate.
UCoord	The tool coordinate system rotation about the Z axis.
VCoord	The tool coordinate system rotation about the Y axis.
WCoord	The tool coordinate system rotation about the X axis.

See Also
Arm, Armset, GetTool, Tool

TLSet Example
```
m_spel.TLSet(1, .5, 4.3, 0, 0, 0, 0)
```

TMove Method, Spel Class

Description
Executes linear interpolation relative motion, in the selected tool coordinate system

Syntax
Sub **TMove** (*PointNumber* As Integer)
Sub **TMove** (*Point* As SpelPoint)
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.
PointExpr	Specifies the end point by using a string expression.

See Also
Go, Jump, Move, TGo

TMove Example
```
m_spel.Tool(1)
m_spel.TMove(10)
```

Tool Method, Spel Class

Description
Selects a tool definition.

Syntax
Sub **Tool** (*ToolNumber* As Integer)

Parameters

ToolNumber	Integer number from 0-3 representing which of 4 tool definitions to use with the upcoming motion instructions.

See Also
TLSet, Arm, TGo, TMove

Tool 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
```
If m_spel.TrapStop() Then
    MsgBox "Robot stopped by Trap"
End If
```

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
```
Const PartPresent = 1
m_spel.WaitSw(PartPresent, True, 5)
If m_spel.TW() Then
    MsgBox "Part present time out occurred"
End If
```

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 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 Double)

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 String)

Parameters

Sequence	String expression containing the name of a vision sequence in the current EPSON RC+ project.
Object	String expression containing the name of an object in sequence *Sequence*. If the property is for a sequence, then this string must be empty.
PropCode	A SpelVisionProps value that specifies the property code.
Value	Variable containing property or result value. The type of the variable must match the property or result type.

See Also
VSet, VRun

VGet 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
```

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

Sequence	String expression containing the name of a vision sequence in the current EPSON RC+ project.
Object	String expression containing the name of an object in sequence *Sequence*.
Result	Integer expression representing the result number.
Found	Boolean variable that will contain whether or not the object was found.
X	Real variable that will contain x coordinate in millimeters.
Y	Real variable that will contain y coordinate in millimeters.
U	Real variable that will contain angle in degrees.

See Also
VGetPixelXYU, VGetRobotXYU

VGetCameraXYU 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)
```

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

Sequence	String expression containing the name of a vision sequence in the current EPSON RC+ project.
Object	String expression containing the name of an object in sequence *Sequence*.
Result	Integer expression representing the result number.
MinX	Real variable that will contain minimum x coordinate in pixels.
MaxX	Real variable that will contain maximum x coordinate in pixels.
MinY	Real variable that will contain minimum y coordinate in pixels.
MaxY	Real variable that will contain maximum y coordinate in pixels.

See Also
VGet

VGetExtrema 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
```

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

Sequence	String expression containing the name of a vision sequence in the current EPSON RC+ project.
Object	String expression containing the name of an object in sequence *Sequence*.
Left	Integer variable that will contain left coordinate in pixels.
Top	Integer variable that will contain top coordinate in pixels.
Width	Integer variable that will contain width in pixels.
Height	Integer variable that will contain height in pixels.

See Also
VSetModelWin, VGetSearchWin, VSetSearchWin

VGetModelWin 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
```

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

Sequence	String expression containing the name of a vision sequence in the current EPSON RC+ project.
Object	String expression containing the name of an object in sequence *Sequence*.
Result	Integer expression representing the result number.
Found	Boolean variable that will contain whether or not the object was found.
X	Real variable that will contain x coordinate in pixels.
Y	Real variable that will contain y coordinate in pixels.
U	Real variable that will contain the angle in degrees.

See Also
VGetCameraXYU, VGetRobotXYU

VGetPixelXYU 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)
```

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

Sequence	String expression containing the name of a vision sequence in the current EPSON RC+ project.
Object	String expression containing the name of an object in sequence *Sequence*.
Result	Integer expression representing the result number.
Found	Integer variable that will contain boolean found status. If found is false, then *x, y,* and *u* are undefined.
X	Real variable that will contain x coordinate in millimeters.
Y	Real variable that will contain y coordinate in millimeters.
U	Real variable that will contain the angle in degrees.

See Also
VGetCameraXYU, VGetPixelXYU

VGetRobotXYU 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)
```

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

Sequence	String expression containing the name of a vision sequence in the current EPSON RC+ project.
Object	String expression containing the name of an object in sequence *Sequence*.
Left	Integer variable that will contain left coordinate in pixels.
Top	Integer variable that will contain top coordinate in pixels.
Width	Integer variable that will contain width in pixels.
Height	Integer variable that will contain height in pixels.

See Also
VGetModelWin, VSetModelWin, VSetSearchWin

VGetSearchWin 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
```

VRun Method, Spel Class

Description
Run a vision sequence in the current EPSON RC+ project.

Syntax
Sub **VRun** (*Sequence* As String)

Parameters

Sequence String containing the name of a sequence in the current EPSON RC+ 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

```
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", SpelVisionProps.CameraX, x)
      .VGet("seq01", "corr01", SpelVisionProps.CameraY, y)
      .VGet("seq01", "corr01", SpelVisionProps.Angle, angle)
      FindPart = True
    End If
  End With
End Function
```

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 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 Double)

Sub **VSet** (*Sequence* As String, *Object* As String, *PropCode* As SpelVisionProps, *Value* As String)

Parameters

Sequence	String expression containing the name of a vision sequence in the current EPSON RC+ project.
Object	String expression containing the name of an object in sequence *Sequence*. If the property is for a sequence, then this string must be empty.
PropCode	A SpelVisionProps value that specifies the property code.
Value	Expression containing the new value. The expression type must match the property type.

See Also
VGet, VRun

VSet 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

Sequence	String expression containing the name of a vision sequence in the current EPSON RC+ project.
Object	String expression containing the name of an object in sequence *Sequence*.
Left	Integer expression representing left coordinate in pixels.
Top	Integer expression representing top coordinate in pixels.
Width	Integer expression representing width in pixels.
Height	Integer expression representing height in pixels.

See Also
VGetModelWin, VGetSearchWin, VSetSearchWin

VSetModelWin 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
```

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

Sequence	String expression containing the name of a vision sequence in the current EPSON RC+ project.
Object	String expression containing the name of an object in sequence *Sequence*.
Left	Integer expression representing left coordinate in pixels.
Top	Integer expression representing top coordinate in pixels.
Width	Integer expression representing width in pixels.
Height	Integer expression representing height in pixels.

See Also
VGetModelWin, VSetModel, VGetSearchWin

VSetSearchWin 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
```

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

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)

Parameters

BitNumber	Integer expression representing the memory bit number.
Condition	Boolean expression representing the memory bit status.
Timeout	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
```
' 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
```

WaitSw Method, Spel Class

Description
Waits for input bit status to change.

Syntax
Sub **WaitSw** (*BitNumber* As Integer, *Condition* As Boolean, *Timeout* As Single)

Parameters

BitNumber	Integer expression representing the input bit number.
Condition	Boolean expression representing the input bit status.
Timeout	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

```
Const PartPresent = 1
m_spel.WaitSw(PartPresent, True, 5)
If m_spel.TW() Then
    MsgBox "Part present time out occurred"
End If
```

Weight Method, Spel Class

Description
Specifies the weight parameters for the robot arm.

Syntax
Sub **Weight** (*PayloadWeight* As Single, *ArmLength* As Single)
Sub **Weight** (*PayloadWeight* As Single, *Axis* As SpelAxis, [Axis])

Parameters

PayloadWeight	The weight of the end effector to be carried in Kg units.
ArmLength	The distance from the rotational center of the second arm to the center of the gravity of the end effector in mm units.
Axis	Specifies which additional axis (S or T) is assign the payload weight.

See Also
JRange, Tool

Weight Example
```
m_spel.Weight(2, 2.5)
```

Xqt Method, Spel Class

Description
Start one SPEL+ 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+ 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 status event to wait for a task to finish.

See Also
Call, EnableEvent, EventReceived

Xqt Example
```
m_spel.Xqt(2, "conveyor")

' Supply an argument to the RunPart function
m_spel.Xqt(3, "RunPart(3)")

Dim funcCall As String
funcCall = "RunPart(" & partNum & ")"
m_spel.Xqt(3, funcCall)
```

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

XLowerLimit 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.)

XUpperLimit 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.)

YLowerLimit 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.)

YUpperLimit 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.)

ZLowerLimit 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.)

ZUpperLimit 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.)

See Also
JRange

XYLim 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
```
m_spel.XLLimClr()
```

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
```
m_spel.XLLimDef()
```

13.4 Spel Class Events

EventReceived Event, Spel Class

Description

Occurs when EPSON RC+ 5.0 sends a system event or when a program running in SPEL+ sends an event using a SPELCom_Event statement.

Syntax

EventReceived (ByVal *sender* As Object, ByVal *e* As SpelNetLib.SpelEventArgs)

Parameters

e.Event Number representing a specific user-defined event.

e.Message String containing event data.

Remarks

There are several system events that EPSON RC+ 5.0 issues. The following table describes them.

System Events

Some events are disabled by default. To use these events you need to enable them by 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	"ALL TASKS STOPPED"	SpelEvents.AllTasksStopped	Occurs when all tasks have been stopped. Enabled 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+ task. Enabled by default.
7	"ESTOP ON"	SpelEvents.EStopOn	Occurs when emergency stop condition changes to ON. Enabled by default.
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.

Event Number	Event Message	Constant	Description
10	"MOTOR ON"	SpelEvents.MotorOn	Occurs when robot motors are turned on. Disabled by default.
11	"MOTOR OFF"	SpelEvents.MotorOff	Occurs when robot motors are turned off. Disabled by default.
12	"POWER HIGH"	SpelEvents.PowerHigh	Occurs when robot power is set to high. Disabled by default.
13	"POWER LOW"	SpelEvents.PowerLow	Occurs when robot power is set to low. Disabled by default.
14	"TEACH MODE"	SpelEvents.TeachMode	Occurs when Teach Mode is activated. Disabled by default.
15	"AUTO MODE"	SpelEvents.AutoMode	Occurs when Auto Mode is activated. Disabled by default.
16	"<TaskID>,<Status>, <FuncName>" Status: "RUN", "HALT", "PAUSE", “FINISHED”, “ABORTED”	SpelEvents.TaskStatus	Occurs when task status changes. Disabled by default.
17	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+ GUI. This event must be enabled with the EnableEvent method. Disabled by default.

Handling Events

When **EventReceived** is called from the Spel class instance, it is waiting for the routine to finish. Therefore, you should never try to execute any VB Guide 5.0 commands from within the **EventReceived** routine. If you want to execute commands based on an event that occurred, set a flag in **EventReceived** that can be read by an event loop in your program.

For example, in your VB main form Load procedure, you can create an event loop that services events received from SPEL+. In the EventReceived routine, set global flags to indicate which events were received. Then, in your ServiceEvents routine, you can check which events occurred and issue the appropriate commands.

User Events

You can also send events from your SPEL+ programs to your Visual Basic application.

To display print event data

Add a TextBox to a form.

Each time the event is received, use the SelText property of the TextBox to append each message. See the example.

To send an event from SPEL+ to VB

Use the SPELCom_Event command in a SPEL+ program statement. For example:

```
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+ 5.0 Help for details on SPELCom_Event.

```
Sub m_spel_EventReceived ( _
        ByVal sender As Object, _
        ByVal e As SpelNetLib.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
```

See Also

EnableEvent (Spel Class)

13.5 SPELVideo Control

Description

This control allows you to display video from an EPSON Smart Camera or Compact Vision system.

File Name

SpelNetLib.dll (32-bit)

SpelNetLib_x64.dll (64-bit)

13.6 SPELVideo Control Properties

This control supports the properties listed below in addition to all standard .NET component properties. They are documented in this manual. See the Visual Basic on-line Help for documentation on the standard properties.

- 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

Camera

Type

Integer

Default Value

0 – any camera is displayed

See Also

VideoEnabled, GraphicsEnabled

Examples

```
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

GraphicsEnabled

Type

Boolean

Default Value

False

See Also

Camera, VideoEnabled

Examples

```
SpelVideo1.GraphicsEnabled = True
```

VideoEnabled Property, SPELVideo Control

Description

Determines whether video is displayed.

Syntax

VideoEnabled

Type

Boolean

Default Value

False

See Also

Camera, GraphicsEnabled

Examples

```
SpelVideo1.VideoEnabled = True
```

13.7 SPELVideo Control Events

All of the events for this control are standard .NET events. See the Visual Basic on-line Help for details.

13.8 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.

Here is an eample.

```
Dim info As SpelNetLib.SpelControllerInfo
info = m_spel.GetControllerInfo()
Label1.Text = info.ProjectID + " " + info.ProjectName
```

13.9 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.

```
Try
  m_spel.Go(1)
Catch (ex As SpelNetLib.SpelException)
  MsgBox(ex.ErrorNumber & " " & ex.Message)
End Try
```

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.

13.10 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 examples:

```
1:
Dim pt As New SpelNetLib.SpelPoint(25.5, 100.3, -21, 0)
m_spel.Go(pt)

2:
Dim pt As New SpelNetLib.SpelPoint
pt.X = 25.5
pt.Y = 100.3
pt.Z = -21
m_spel.Go(pt)

3:
Dim pt As New SpelNetLib.SpelPoint
pt = m_spel.GetPoint("P*")
pt.Y = 222
m_spel.Go(pt)
```

SpelPoint Properties

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

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+.

Format can be:

Empty	Returns the entire point with all coordinates and attributes.
"XY"	Returns "XY(…)"
"XYST"	Returns "XY(…) :ST(…)"

13.11 Enumerations

13.11.1 SpelBaseAlignment Enumeration

Member name	Value	Description
XAxis	0	Align with X axis
YAxis	1	Align with Y axis

13.11.2 SpelDialogs Enumeration

Member name	Value	Description
RobotManager	1	ID for Robot Manager dialog
ControllerTools	2	ID for Tools | Controller dialog

13.11.3 SpelElbow Enumeration

Member name	Value	Description
Above	1	Elbow orientation is above.
Below	2	Elbow orientation is below.

13.11.4 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.
AllTasksStopped	4	ID for all tasks stopped 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.
TaskStatus	16	ID for task status event.
ProjectBuildStatus	17	ID for project build status event.

13.11.5 SpelHand Enumeration

Member name	Value	Description
Righty	1	Hand orientation is righty.
Lefty	2	Hand orientation is lefty.

13.11.6 SpelIOLabelTypes 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.

13.11.7 SpelOperationMode Enumeration

Member name	Value	Description
Auto	1	EPSON RC+ is in auto mode.
Program	2	EPSON RC+ is in program mode.

13.11.8 SpelRobotPosType Enumeration

Member name	Value	Description
World	0	Specifies world coordinates.
Joint	1	Specifies joint coordinates.
Pulse	2	Specifies pulses.

13.11.9 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.

13.11.10 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.

13.11.11 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.

13.11.12 SpelVisionProps Enumeration

This enumeration is for all vision properties and results. Refer to the Vision Guide Reference manual for details.

13.11.13 SpelWrist Enumeration

Member name	Value	Description
NoFlip	1	Wrist orientation is no flip.
Flip	2	Wrist orientation is flip.

13.11.14 SpelWindows Enumeration

Member name	Value	Description
IOMonitor	1	ID for the I/O Monitor window.
TaskManager	2	ID for the Task Manager window.
TaskManager	3	ID for the Simulator window.

13.12 Spel Error Numbers and Messages

For error numbers and error messages, see the *SPEL+ Language Reference*.

14. Using With LabVIEW

14.1 Overview

This chapter contains information for using LabVIEW with VB Guide 5.0. The following topics are described.

- Set the VI execution mode
- Initialization
- Use Spel properties and methods in your application
- Shutdown
- Using dialogs and windows

14.2 Setting VI Execution Mode

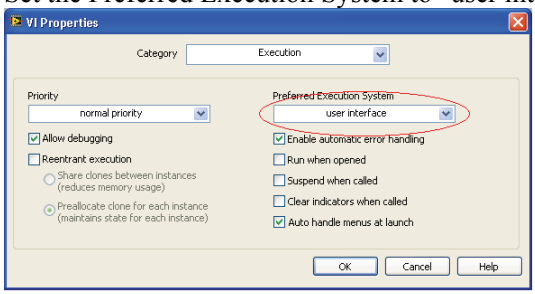
You must first decide whether to run EPSON RC+ 5.0 in-process or out-of-process.

If you are using RC+ as an **out-of-process** server (recommended, ServerOutOfProcess = True), you can set the VI execution mode to "standard" (default) so that Spel methods will not run in the VI GUI thread.

If you are using RC+ as an **in-process** server (default, ServerOutOfProcess = False), the Spel class needs to run in a single thread apartment, so the VI in which it is used needs its execution mode set to "user interface".

To set the execution mode to "user interface" (if using RC+ as an in-process server):

1. In the LabView VI where you will use the Spel class, select VI Properties from the File menu, then select the Execution Category.
2. Set the Preferred Execution System to "user interface".



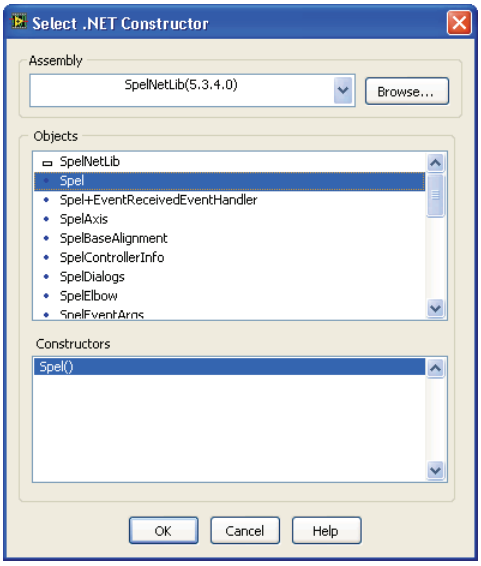
Note: It is recommended that you use EPSON RC+ as an out-of-process server when using LabVIEW. Also, when using the 64-bit library, EPSON RC+ is always out-of-process.

14.3 Initialization

14.3.1 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 Connectivity - .NET palette. The Select .NET Constructor dialog will appear. Select SpelNetLib for 32-bit LabVIEW or SpelNetLib_x64 for 64-bit LabVIEW in the Assembly list and select Spel in the Objects list, as shown below.

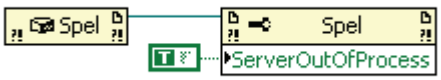


This will create a constructor node for Spel in the block diagram.

14.3.2 Add a property node to set out-of-process operation

If using EPSON RC+ out-of-process with LabVIEW 32-bit, then in the Block Diagram view, add a Property Node, then select the ServerOutOfProcess Property. Right click and select Change to Write. Use a Boolean constant to set the property to True. Wire the reference output from the Spel contructor to the reference in of the property node.

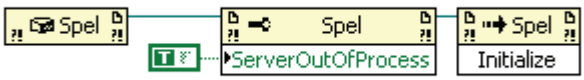
Note: The ServerOutOfProcess property must be set before any other properties or methods are used.



Note: ServerOutOfProcess is not used in the 64-bit version (SpelNetLib_x64).

14.3.3 Initialize the Spel class instance

Add an Invoke node for the Spel class Initialize method and wire the reference ouput from the ServerOutOfProcess node to the reference input for the Initialize node. When Initialize executes, it will configure and start RC+. If ServerOutOfProcess is true, then RC+ is started in a separate process.



14.4 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 SpelNetLib Reference chapter for information on the properties and methods that can be used.

14.5 Shutdown

When you are finished using the Spel class instance, you need to invoke the Dispose method. This will shutdown the EPSON RC+ 5.0 server. Normally, you should call Dispose at the end of your application.

When using RC+ as an out-of-process server: If your application is aborted without calling Dispose, then the RC+ 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 (erc50.exe) from the Windows Task Manager first, then run the RC+ GUI.

14.6 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.

15. Using With VS 2010 or VS 2012

15.1 Overview

If you created an application using Visual Studio 2010, the default target .NET Framework is v4.0. If you created an application using Visual Studio 2012, the default target .NET Framework is v4.5. By default, applications that target .NET Framework 4.0 or greater cannot load assemblies from previous .NET Framework versions. The VB Guide 5.0 SpelNetLib.dll and SpelNetLib_x64.dll assemblies were created with .NET Framework 3.5. If you want to use SpelNetLib with your VS 2010 or VS 2012 application, there are two choices:

- Change the application target .NET Framework to v3.5
  OR
- Modify the application configuration

15.2 Change the .NET target framework to v3.5

For VB and C#:
In the project properties page, change the target .NET Framework to 3.5. Now your application can be used with SpelNetLib.

For VC++:
1. Ensure that VS 2008 is installed on the same system.
2. In Configuration Properties | General, change the platform toolset to V90.
3. Close Visual Studio.
4. Modify the vcxproj file for your project using Notepad. In the <PropertyGroup Label = "Globals"> element, change the value for TargetFrameworkVersion to 3.5. Add TargetFrameworkVersion if is it missing.

   <TargetFrameworkVersion>v3.5</TargetFrameworkVersion>

5. Open Visual Studio and build the project.

15.3 Modify the application configuration

Follow the procedures below if your application uses .NET Framework 4.0 or greater:

For VB and C#:
1. In Solution Explorer, check if app.config exists. If it does, go to step 4.
2. From the Project menu, select Add New Item.
3. Select Application Configuration File and click Add. The app.config XML file will appear.
4. In the app.config file, add the following XML code inside the configuration property.

```
<configuration>
    <startup useLegacyV2RuntimeActivationPolicy="true">
      <supportedRuntime version="v4.0"/>
    </startup>
</configuration>
```

For VC++:
1. Create a config file for the application that uses the same name as the exe file with ".config" appended. For example, if you exe file name is myapp.exe, then the config file would be named myapp.exe.config.

2. In the config file, add the configuration code:

```
<configuration>
     <startup useLegacyV2RuntimeActivationPolicy="true">
       <supportedRuntime version="v4.0"/>
     </startup>
</configuration>
```

Copy the config file to both the debug and release folders. You can use a post build event that performs this automatically. For an example, see the VB Guide 5.0 VC++ demo project.

16. 32 Bit and 64 Bit Applications

Two libraries are supplied with VB Guide 5.0. For your applications that use VB Guide to run correctly, you need to reference the correct library and also use the correct solution platform.

SpelNetLib.dll	Compiled as a 32 bit library. Applications using this library should be compiled to run as a 32 bit application.
SpelNetLib_x64.dll	Compiled as a 64 bit library. Applications using this library should be compiled to run as a 64 bit application.

16.1 Using 32 Bit Windows

VB and C# 32 Bit Applications
Add a reference to SpelNetLib.dll. The solution platform should be set to x86.

VC++ 32 Bit Applications
Add a reference to SpelNetLib.dll. The default solution platform win32 should be used.

16.2 Using 64 Bit Windows

VB and C# 32 Bit Applications
Add a reference to SpelNetLib.dll. The solution platform should be set to x86.

VC++ 32 Bit Applications
Add a reference to SpelNetLib.dll. The default solution platform win32 should be used.

VB and C# 64 Bit Applications
Add a reference to SpelNetLib_x64.dll. The solution platform should be set to x64.

VC++ 64 Bit Applications
Add a reference to SpelNetLib_x64.dll. The solution platform should be x64. You may also need to install 64 bit tools.