Adept ActiveV Developer's Guide

This is a printed version of the Adept ActiveV+ Developer's Guide online documentation. A Table of Contents is provided so that you can locate the desired topics. Because this document is designed for online viewing, cross–references will not have page numbers. For easier navigation use the online version of this document, which can be accessed from the Adept Document Library or from the software CD that came with your system.

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Welcome to the Adept ActiveV Library Developer's Guide

The ActiveV library is an ActiveX (COM) library, which allows you to call V+ or MicroV+ operating system commands from other PC applications developed using tools such as Microsoft® Visual Basic and C++. These applications can be used to control Adept Cobra robots, AdeptSix 6−axis robots, Adept SmartModules and all SmartController products. The ActiveV library (ActiveV2Lib) is delivered as part of Adept DeskTop software version 2.x and later.

NOTE: The ActiveV library was originally delivered through the class library named ACTIVEVLib, in the file ActiveV.dll. Subsequently, additional functionality was added, and a new library named ActiveV2Lib, in the file ActiveV2.dll, was created to supply this new functionality. ActiveV2Lib maintains backwards compatibility, as much as possible, with the original ACTIVEVLib. However, because ActiveV2Lib has more functionality, some incompatibilities exist. Further, ACTIVEVLib is no longer supported. Therefore, all new development should use only ActiveV2Lib.

This documentation provides a programmer's reference for the ActiveV library (ActiveV2Lib), which provides access to V+ operating system functionality. It includes a description of the classes, methods, and properties for the library.

This document also refers to related V+ and MicroV+ commands. In most cases, links to the documentation for these commands are provided, but you may want to review the ***V+ Operating System Reference Guide***, the ***V+ Language Reference Guide***, and the ***MicroV+ Online Help***, so that you are familiar with the operating system environment that will be accessed from your PC applications through the ActiveV library.

What's New in This Version?

The following new functionality was added in ActiveV2Lib:

Methods	Description	Corresponding V+ Keyword	Corresponding MicroV+ Keyword
ListsB	Returns the values of V+ string variables as binary values.	N/A	N/A
XStep2	Execute a single step of a program.	XSTEP	XSTEP
Properties	Description	Corresponding V+ Keyword	Corresponding MicroV+ Keyword

ControllerIPAddress	Returns the IP address of the controller connected to this Communications object.	N/A	N/A
IsGetc	Returns TRUE if the V+ program executed a GETC() instruction to obtain input.	N/A	N/A
Events	Description	Corresponding V+ Keyword	Corresponding MicroV+ Keyword
OnAlive	This event is raised periodically when activity is detected on the network while the application is connecting to a V+ controller.	N/A	N/A
OnAsynchError	Indicates that an asynchronous error has occurred on the V+ system.	N/A	N/A
OnProgramOutput	Indicates that a V+ TYPE or WRITE instruction has been executed.	N/A	N/A
OnProgramStatusChange	Indicates that a V+ program has completed or changed status.	N/A	N/A
OnReadPosted	Indicates that a V+ program has executed a READ instruction and is waiting for input from the keyboard.	N/A	N/A

What would you like to do?

Read the ActiveV library product overview

Install the ActiveV library

View the V+ Operating System Reference Guide

View the V+ Language Reference Guide

View the MicroV+ Online Help

ActiveV Library Product Overview

Adept ActiveV library allows real−time control of Adept robots and motion controllers from native PC languages using Microsoft Component Object Model (COM) technology. Adept ActiveV library enables application developers to control Adept Cobra (s−series and i−series) robots, AdeptSix 6−axis robots, Adept SmartModules and all SmartController −based products from PC hardware using software tools like Visual Basic fi, Visual Basic .NET fi and C# .NET fi.

Adept ActiveV library allows integrators, robot system developers and machine builders to develop application code independent of robot hardware, ensuring the reusability and security of their development effort, and enabling software developers to quickly apply previously developed and debugged PC−based application software to Adept robots and motion control products using standard open architecture tools.

Adept ActiveV library allows you to optimize your control methodology with either a 100% PC−based approach, or a hybrid approach that takes advantage of the additional processing capabilities on the Adept controller.

Features

- PC−based robot control and integration
- 100% Open Architecture
- Serial or Ethernet interface
- Industrial motion controller robustness
- Microsoft ActiveX (COM) compatible
- Does not require learning a new robot programming language. (The programmer should be familiar with the V+ and/or MicroV+ operating system. See the ***V+ Operating System Reference Guide***, the ***V+ Language Reference Guide***, and the ***MicroV+ Online Help*** for details.)

Components

Adept ActiveV library contains the following ActiveX (COM) libraries:

ActiveV version 2.0 (ActiveV2.dll)

- ♦ Available for Adept Cobra robots, AdeptSix 6−axis robots, Adept SmartModules and all SmartController CX controller−based products
- ♦ Provides communication between the PC and V+ controller via a Serial or Ethernet connection.
- ♦ Available with MicroV+
- ♦ Operates in the V+ monitor window buffer
- ♦ Provides real−time access to V+ operating system commands from a PC application.

What would you like to do?

Review the system requirements

Develop a PC application that communicates with the V+ operating system

System Requirements

The ActiveV library runs on Windows 2000 and Windows XP. Unless noted, the term “Windows” refers to these operating systems. The ActiveV library (ActiveV2.dll) is distributed as part of the Adept DeskTop installation.

PC Requirements

In order to use the Adept ActiveV library, your PC should be configured with:

- Microsoft Windows−compatible PC that is capable of running the Windows 2000 or Windows XP operating system.
- Microsoft Windows 2000 or Windows XP operating system

  NOTE: Make sure you have the latest service pack and critical updates for the version of Windows that you are running. Operating system and security updates can be obtained from the Microsoft Windows Update site at http://windowsupdate.microsoft.com.
- For the .NET examples, Microsoft .NET Framework version 1.1 (included in the installer for Adept DeskTop version 2.1 and later)
- For the Cookie Demo application example, ActiveSCANLib ActiveX control is required (included in the installer for Adept DeskTop 2.1 and later)
- Microsoft Internet Explorer version 5.01 or later
- 500 Mb hard disk (recommended minimum). More disk space will be required if you choose to copy the documentation files to your hard disk.
- Minimum of 128 Mb RAM (more memory may be required to run Windows 2000 or Windows XP)
- SVGA monitor
- CD−ROM drive
- 10/100 base−T Ethernet

- For the Ethernet option, the PC requires either the Adept−supplied crossover cable, or an Ethernet card and networking hardware (cabling, hub, etc.).
- For the serial option, a serial−interface cable is required. (This option is only available for the ActiveV library.)

Adept SmartController Requirements

In order to use the Adept ActiveV library, your Adept SmartController should be configured with:

- V+ version 16.2 or later, for full functionality

- An Adept ActiveV Controller License

Adept MicroV+ Systems Requirements

In order to use the Adept ActiveV library, your Adept MicroV+ equipped system should be configured with:

- Micro V+ Version 2.1 or later (for Cobra i−series systems), for full functionality

Network Requirements

Adept ActiveV library users should be aware that a fixed IP address is required for both the PC and the Controller. Typically, IP addresses are allocated by your IT department or network administrator.

Documentation Conventions

This documentation describes the classes, events, methods, and properties included in the Adept ActiveV library. These methods may be called by application software written by a system customizer using software such as Visual Basic 6.0, Visual Basic .NET and C# .NET. For details on the documentation conventions, choose a topic:

ActiveV Library Data Types

Within this documentation, the parameters are documented to be of a specific type. The data types used correspond to the Visual Basic 6.0 data types. The corresponding meaning in Visual Basic .NET and C# .NET are as follows:

Visual Basic	Visual Basic .NET	C# .NET
String	String	System.String
ArrayString	System.Array	System.Array
Long	Integer	System.Int32
ArrayLong	System.Array	System.Array
Single	Single	System.Single
ArraySingle	System.Array	System.Array

Class Documentation

The class documentation topics include a brief description of the class followed by a table summarizing the events, methods, and properties available within that class. The table also indicates the equivalent V+ keyword if applicable and provides links to the corresponding V+ documentation for that keyword.

Method, Property, and Event Documentation

The descriptions of the methods are presented in alphabetical order, with each method starting on a separate page. The description for each method contains the following sections, as applicable.

NOTE: The variable names used for the method parameters are for explanation purposes only. Your application program can use any variable names you want when calling the method.

Syntax

> ***Example:*** Class.Name ({Parameter1 as Long}, Parameter 2 as Single, ...)

Each documented item includes a syntax statement which includes the following as required:

Class	Indicates the class to which the method, property, or event belongs. To review details about the class, click on the class name. If an item belongs to more than one class, both classes will be listed.
Name	The name of the method, event, or property
Parameters	Parameters are listed along with the associated data type. If a data type is not provided, the parameter uses the VB Variant data type. Optional parameters are enclosed by braces: {}

Description

This is a brief description of the method.

Usage Considerations

This section is used to point out any special considerations associated with use of the event, method, or property.

Input Parameters

Each of the input parameters in the method is described in detail. For any parameter that has a restriction on its acceptable values, or is required to be defined as a double−precision variable, the restriction/requirement is specified.

Output Parameters

Each of the output parameters used in the method is described in detail.

Details

If necessary, this section provides more complete information about the method or property and the circumstances in which it is used.

Examples

In some cases, a code sample is included to clarify the use of the event, method, or property. Most of these samples include Visual Basic 6, Visual Basic .NET and C# .NET code.

Unless noted otherwise, the example code is taken directly from the Cookie Demo sample application.

Corresponding V+ / MicroV+ Keywords

Where applicable, the corresponding V+ and/or MicroV+ keywords are listed with links to those topics.

Related Topics

Other events, methods, or properties with related functions are listed.

What do you want to do?

View the ActiveV Library Reference

How Can I Get Help?

Installing the Adept ActiveV Library

Adept ActiveV library is supplied as part of Adept DeskTop. When Adept DeskTop is installed, the ActiveV library dll file is automatically installed, too.

1. Verify that your system meets the hardware and software requirements for the Adept ActiveV library. More...
2. Run the setup program on the Adept DeskTop software CD. More...

What would you like to do?

Review the system requirements

Learn more about Adept ActiveV software

Add the ActiveV Library to Your Project

To access the Adept ActiveV library functionality from your PC application, you need to add the following COM libraries to your project resources. For details on adding references to your project, refer to the documentation for your program development tool.

- ActiveV2Lib.dll

The sections below describe how to add Imports to your Visual Basic, Visual Basic .NET, and C# .NET development projects. The remainder of this online documentation assumes you have added these Imports.

Visual Basic Projects

For convenience, you should also add the following lines to the top of your Visual Basic program to avoid typing all the ActiveV2Lib prefix information:

```
ActiveV2Lib Imports Statements
```

Visual Basic .NET Projects

For convenience, you can also add the following lines to the top of your Visual Basic .NET program to avoid typing all the ActiveV2Lib prefix information:

```
Imports ActiveV2Lib;
```

or more general

```
Imports ActiveV2Lib.(statement);
```

C# .NET Projects

For convenience, you can also add the following lines to the top of your C# .NET program to avoid typing all the ActiveV2Lib prefix information:

```
using ActiveV2Lib;
```

or more general

```
using ActiveV2Lib.(statement);
```

What would you like to do?

Explore the ActiveV Library Reference

View the Quick Start procedure

ActiveV Library Classes

The ActiveV library methods are organized into the following classes:

- Communications
- EditorFuncs (Editor Functions)
- ErrorHandler (Error Handler)
- Files
- MiscControl (Miscellaneous Control)
- Programs
- Status
- Version

Simply click a link above to access the page describing that class. The class pages are presented in table format. Each table row contains a link to the ActiveV command, a brief description of that command, and links to corresponding V+ and MicroV+ keywords.

The Communications Class

The Communications class provides communication with V+ (through Ethernet) or MicroV+ (through a serial line) for all other classes within the ActiveV library. A simple programming model would be to create a single object of the Communications class and connect it to the desired controller, and to use this object with any method that needs a Communications object. Creating a single Communications object and reusing it provides a performance advantage because it avoids the slight overhead incurred each time an object is created and connected it to V+.

When connecting through Ethernet to V+, multiple Communications objects may be created to connect to multiple V+ controllers. When connecting to MicroV+ through a serial line, only one Communications object may be connected to one MicroV+ controller at a time.

Methods

Properties

Events

Methods	Description	Corresponding V+ Keyword	Corresponding MicroV+ Keyword
Close	Closes the communication channel between the PC and the V+ system.	N/A	N/A
GetStringEx	Read a string output by a V+ TYPE or WRITE instruction.	N/A	N/A
Open	Opens a communication channel to a V+ system.	N/A	N/A
QueryReadRequest	Query V+ for a pending read request on a specified board and task.	N/A	N/A
RequestEvents	Request V+ sends asynchronous output and program output to the ActiveV application.	N/A	N/A

SendStringEx	Send a string to V+ for input to a program running on a specified board/task.	N/A	N/A

Properties	Description	Corresponding V+ Keyword	Corresponding MicroV+ Keyword
ControllerIPAddress	Returns the IP address of the controller connected to this Communications object.	N/A	N/A
IsGetc	Returns TRUE if the V+ program executed a GETC() instruction to obtain input.	N/A	N/A
IsOnline	Determines whether the PC is connected to the V+ system.	N/A	N/A

Events	Description	Corresponding V+ Keyword	Corresponding MicroV+ Keyword
OnAlive	This event is raised periodically when activity is detected on the network while the application is connecting to a V+ controller.	N/A	N/A
OnAsynchError	Indicates that an asynchronous error has occurred on the V+ system.	N/A	N/A
OnProgramOutput	Indicates that a V+ TYPE or WRITE instruction has been executed.	N/A	N/A
OnProgramStatusChange	Indicates that a V+ program has completed or changed status.	N/A	N/A
OnReadPosted	Indicates that a V+ program has executed a READ instruction and is waiting for input from the keyboard.	N/A	N/A

The Editor Functions Class

The Editor Functions class (EditorFuncs) is used to modify programs that reside in V+ or MicroV+ memory. Before calling any other method in this class, you must call the Edit method.

Method	Description	Corresponding V+ Keyword	Corresponding MicroV+ Keyword
ChangeLine	Change the content of a specific line in a V+ program.	V+ Line Editor	MicroV+ Line Editor
CheckProgram	Check program for syntax errors and return first error encountered.	V+ automatically checks syntax during program editing.	V+ automatically checks syntax during program editing.
DeleteLine	Delete a specified line in a specified V+ program.	V+ Line Editor	MicroV+ Line Editor
Edit	Specify the V+ program that will be accessed by subsequent calls to other methods in this EditorFuncs interface.	V+ Line Editor	MicroV+ Line Editor
InsertLine	Insert a new line in the specified V+ program.	V+ Line Editor	MicroV+ Line Editor

The Error Handler Class

The Error Handler class provides access to the methods that translate error codes into descriptive strings.

Method	Description	Corresponding V+ Keyword	Corresponding MicroV+ Keyword
GetErrorString	Return the error message for a given error number.	N/A	N/A
GetVErrorString	Return the error message for a given V+ error number.	$ERROR	N/A

The Files Class

The Files class provides access to methods that access the V+ file system, including files available on a hard disk drive, CompactFlash card, NFS mount, and others.

Method	Description	Corresponding V+ Keyword	Corresponding MicroV+ Keyword
FCopy	Copies a file.	FCOPY	N/A
FDelete	Deletes a File.	FDELETE	N/A
FDirectory	Obtains a directory listing.	FDIRECTORY	N/A
FRename	Renames a file.	FRENAME	N/A
NFSMounts	Returns information about the current NFS Mounts	NET	N/A

The Miscellaneous Control Class

The Miscellaneous Control (MiscControl) class provides methods to modify the state of the connected V+ or MicroV+ system.

Method	Description	Corresponding V+ Keyword	Corresponding MicroV+ Keyword
Calibrate	Calibrate the robot.	CALIBRATE	CALIBRATE
Parameter	Set the value for a system parameter.	PARAMETER	PARAMETER
SetL	Set the value of a location V+ variable.	SET	SET
SetPP	Set the value of a precision point variable.	#SET.POINT	N/A
SetR	Set the value of a real V+ variable.	N/A	N/A
SetS	Set the value of a V+ string variable.	N/A	N/A
Signal8	Set 8 digital I/O signals at once.	BITS	BITS
SignalOff	Turn off a DIO signal.	SIGNAL	SIGNAL
SignalOn	Turn on a DIO signal.	SIGNAL	SIGNAL
Speed	Set the robot speed.	SPEED	SPEED
SwitchOff	Disable a system switch.	DISABLE	DISABLE
SwitchOn	Enable a system switch.	ENABLE	ENABLE

The Program Class

The Programs class provides access to all methods that manage the storage and execution of V+ or MicroV+ programs in memory.

Method	Description	Corresponding V+ Keyword	Corresponding MicroV+ Keyword
Abort	Stops the specified user task.	ABORT	ABORT
ClearBreakpoint	Clears one or all breakpoints in a V+ program.	BPT	BPT
Delete	Deletes a V+ program including all subroutines or variables referenced by the named program.	DELETE	DELETEP
Deletem	Deletes a V+ module.	DELETEM	N/A
Deletep	Deletes a single V+ program, but does not delete subroutines or variables referenced by the named program.	DELETEP	DELETEP
Directory	Obtains a listing of programs loaded in V+ memory	DIRECTORY	DIRECTORY
Execute	Executes a V+ program.	EXECUTE	EXECUTE
Kill	Kills a task stack	KILL	KILL
ListBreakpoint	Lists all breakpoints in a V+ program.	LISTB	LISTB
Listp	Returns the text of a program that has been loaded in V+ memory.	LISTP	LISTP

Load	Loads programs and variables from a file into V+ memory.	LOAD	LOAD
MDirectory	Lists the modules that are loaded in memory, or alternatively, the programs contained in a specified module.	MDIRECTORY	N/A
Module	Moves a program into a specified module.	MODULE	N/A
ModuleFile	Associates a file name with a module.	N/A	N/A
PCopy	Copy a V+ program to another.	COPY	N/A
Prime	Prepare a V+ program for execution.	PRIME	PRIME
Proceed	Resume execution of an application program.	PROCEED	PROCEED
Rename	Renames a V+ program that is currently in memory.	RENAME	RENAME
Retry	Repeat execution of the last interrupted program instruction and continue execution of the program.	RETRY	RETRY
SetBreakpoint	Sets a breakpoint in a V+ program.	BPT	BPT
SetStackSize	Sets the stack size for the specified task.	STACK	STACK
SStep	Executes one step or an entire program in a specified V+ task.	SSTEP	SSTEP
Store	Saves programs and variables currently in memory to a disk file.	STORE	STORE
Storem	Store specified program module to a disk file.	STOREM	STOREM

Storep	Saves a program to disk	STOREP	N/A
XStep	Executes one step in a specified V+ task.	XSTEP	XSTEP
XStep2	Execute a single step of a program.	XSTEP	XSTEP
ZERO	Clears V+ memory.	ZERO	ZERO

The Status Class

The Status class provides the following methods and properties to retrieve system status information and system variable values.

Methods

Properties

Methods	Description	Corresponding V+ Keyword	Corresponding MicroV+ Keyword
Evaluate	Evaluates an arbitrary V+ expression.	N/A	N/A
Free	Returns the amount of available memory on the V+ controller	FREE	FREE
Here	Sets a variable equal to the current position of the specified robot	HERE	HERE
IO	Samples V+ digital IO signals.	IO	IO
Listl	Returns the value of a location variable.	LISTL	LISTL
Listr	Returns values of real V+ variables.	LISTR	LISTR
Lists	Returns the values of string V+ variables.	LISTS	N/A
ListsB	Returns the values of V+ string variables as binary values.	N/A	N/A
Lvariables	Returns the names of location variables in an array.	LISTL	LISTL
Parameter	Returns the values of system parameters.	PARAMETER	PARAMETER
Rvariables	Returns the names of real variables.	LISTR	LISTR

StackContents	Gets the stack contents for a specified task.	STACK	N/A
Status	Gets the system task/program execution status.	STATUS	STATUS
Svariables	Returns the names of string variables.	LISTS	N/A
Switch	Returns the state of system switches.	SWITCH	SWITCH
Where	Returns the current position of the specified robot.	WHERE	WHERE

Properties	Description	Corresponding V+ Keyword	Corresponding MicroV+ Keyword
NumberOfRobots	Returns the number of robots in the system. This property is not available for MicroV+.	N/A	N/A
SelectedRobot	Returns the currently selected robot. This property is not available for MicroV+.	N/A	N/A

The Version Class

The Version class provides access to the methods that return ActiveV and V+ version information.

Method	Description	Corresponding V+ Keyword	Corresponding MicroV+ Keyword
GetVersion	Returns the version of the ActiveV library.	N/A	N/A
GetVplusVersion	Returns the version of the connected V+ system.	ID	ID

Description of Commands: Overview

This documentation describes the methods and properties included in the Adept ActiveV library. These methods may be called by application software written by a system customizer using software such as Visual Basic or Visual C++.

The descriptions of the methods are presented in alphabetical order, with each method starting on a separate page. The description for each method contains the following sections, as applicable.

NOTE: The variable names used for the method parameters are for explanation purposes only. Your application program can use any variable names you want when calling the method.

Description

This is a brief description of the method.

Usage Considerations

This section is used to point out any special considerations associated with use of the event, method, or property.

Input Parameters

Each of the input parameters in the method is described in detail. For any parameter that has a restriction on its acceptable values, or is required to be defined as a double−precision variable, the restriction/requirement is specified.

Output Parameters

Each of the output parameters used in the method is described in detail.

Details

If necessary, this section provides more complete information about the method or property and the circumstances in which it is used.

Example

In some cases, a code sample is included to clarify the use of the event, method, or property. Most of these samples include Visual Basic 6, Visual Basic .NET and C# .NET code.

Unless noted otherwise, the example code is taken directly from the Cookie Demo sample application.

Corresponding V+ / MicroV+ Keywords

Where applicable, the corresponding V+ and/or MicroV+ keywords are listed with links to those topics.

Related Topics

Other events, methods, or properties with related functions are listed.

Abort Method

Visual Basic 6

```
Programs.Abort(pCommunications as Communications, lTaskNumber as Long,
               pStatus as Long)
```

Visual Basic .NET

```
Programs.Abort(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
               lTaskNumber As Integer, ByRef pStatus As Integer)
```

C# .NET

```
Programs.Abort(ActiveV2Lib.Communications pCommunications, System.Int32
               lTaskNumber, out System.Int32 pStatus)
```

Description

This method aborts a task. Invalid task numbers are ignored.

Input Parameters

lTaskNumber	Task number

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Examples

Visual Basic 6 Visual Basic .NET C# .NET

Visual Basic 6

```
' loop through the query result and stop each task
    On Error Resume Next
    With rec
        Do While Not .EOF
        Set prog = New ActiveV2Lib.Programs

Call prog.Abort(comm, !task, stat)
        If stat <> VE_SUCCESS Or Err.Number <> 0 Then
            If stat <> VE_SUCCESS Then
                Call errh.GetErrorString(stat, str)
            Else
                Call errh.GetErrorString(Err.Number, str)
            End If
            Call MsgBox(str, vbOKOnly, "Error attempting to Stop '" _
            & !Program & "()'")
        End If
```

Visual Basic .NET

```
' Event-handler called by the system when the user clicks the
' Stop button. Abort all configured tasks.
Private Sub StopButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles StopButton.Click

    Dim nStatus As Integer
    Dim i As Integer

    Try
        ' Clear the text box.
        OutputTextBox.Clear()

        ' Go through the arrays to know what to stop
        For i = 0 To mnProgramCount

            ' Start only non-background tasks.
            If mProgramDef(i).bBackGround = False Then
                mPrograms.Abort(mCommunications, mProgramDef(i).nTask, nStatus)
                CheckError(nStatus)
            End If
        Next
    Catch Ex As System.Runtime.InteropServices.COMException
```

```
            ShowError(Ex)
        Catch Ex As Exception
            ShowError(Ex)
        End Try
    End Sub
```

C# .NET

```
    /// <summary>
    /// Called when the user clicks the Stop button.
    /// Stop all the foreground tasks.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void StopButton_Click(object sender, System.EventArgs e)
    {
         int nStatus;

         try
         {

              // Go through the arrays to know what to stop.
              for(int i = 0; i < mnProgramCount; i++)
              {
                   // Stop only non-background tasks.
                   if (mProgramDef[i].bBackGround == false)
                   {
                        mPrograms.Abort(mCommunications, mProgramDef[i].nTask, out nStatus);
                        CheckError(nStatus);
                   }
              }
         }
         catch (System.Runtime.InteropServices.COMException Ex)
         {
              ShowError(Ex);
         }
         catch (Exception Ex)
         {
              ShowError(Ex);
         }
    }
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
ABORT	ABORT

Related Topics

Execute

Kill

Prime

Proceed

ZERO

Calibrate Method

Visual Basic 6

```
MiscControl.Calibrate(pCommunications as Communications, pStatus as Long)
```

Visual Basic .NET

```
MiscControl.Calibrate(ByVal pCommunications As ActiveV2Lib.Communications,
                ByRef pStatus As Integer)
```

C# .NET

```
MiscControl.Calibrate(ActiveV2Lib.Communications pCommunications, out
                System.Int32 pStatus)
```

Description

This method calibrates the robot.

Input Parameters

None.

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Examples

Visual Basic 6 Visual Basic .NET C# .NET

Visual Basic 6

```
Private Sub cmdCalibrateHome_Click()
    Dim mctrl As ActiveV2Lib.MiscControl
    Dim stat As Long
```

```
    Dim str As String

    Set mctrl = New ActiveV2Lib.MiscControl

    ' request calibrate
    On Error Resume Next
    Call mctrl.Calibrate(comm, stat)
    If stat <> VE_SUCCESS Or Err.Number <> 0 Then
        If stat <> VE_SUCCESS Then
            Call errh.GetErrorString(stat, str)
        Else
            Call errh.GetErrorString(Err.Number, str)
        End If
        Call MsgBox(str, vbOKOnly, "Error attempting to Calibrate robot(s)")
    End If

End Sub
```

Visual Basic .NET

```
' Event-handler called by the system when the user clicks the
' Calibrate button. Call the Calibrate command.
Private Sub CalibrateButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CalibrateButton.Click

    Dim nStatus As Integer

    Try
        mMisc.Calibrate(mCommunications, nStatus)
        CheckError(nStatus)
    Catch Ex As System.Runtime.InteropServices.COMException
        ShowError(Ex)
    Catch Ex As Exception
        ShowError(Ex)
    End Try

End Sub
```

C# .NET

```
/// <summary>
/// Called when the user clicks the Calibrate button.
/// Start the calibration task.
```

```
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void CalibrateButton_Click(object sender, System.EventArgs e)
{
     int nStatus;

     try
     {
          mMisc.Calibrate(mCommunications, out nStatus);

          CheckError(nStatus);
     }
     catch (System.Runtime.InteropServices.COMException Ex)
     {
          ShowError(Ex);
     }
     catch (Exception Ex)
     {
          ShowError(Ex);
     }

}
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
CALIBRATE	CALIBRATE

ChangeLine Method

Visual Basic 6

```
EditorFuncs.ChangeLine(pCommunications as Communications, lLineNumber as Long,
                pBstrLineText as String, pStatus as Long)
```

Visual Basic .NET

```
EditorFuncs.ChangeLine(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                lLineNumber As Integer, ByRef pBstrLineText As String, ByRef
                pStatus As Integer)
```

C# .NET

```
EditorFuncs.ChangeLine(ActiveV2Lib.Communications pCommunications, System.Int32
                lLineNumber, ref System.String pBstrLineText, out System.Int32
                pStatus)
```

Description

This method changes the contents of a program line.

Usage Considerations

Before calling this method, you must call the Edit method.

Input Parameters

lLineNumber	Line number of line to change.
pBstrLineText	Text for new line.

Output Parameters

pBstrLineText	Text for new line as formatted by V+.

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
V+ Line Editor	MicroV+ Line Editor

Related Topics

CheckProgram

DeleteLine

Edit

InsertLine

CheckProgram Method

Visual Basic 6

```
EditorFuncs.CheckProgram(pCommunications as Communications, pStatus as Long)
```

Visual Basic .NET

```
EditorFuncs.CheckProgram(ByVal pCommunications As ActiveV2Lib.Communications,
                ByRef pStatus As Integer)
```

C# .NET

```
EditorFuncs.CheckProgram(ActiveV2Lib.Communications pCommunications,
                out System.Int32 pStatus)
```

Description

This method checks the V+ program for syntax errors and returns the first error encountered.

Usage Considerations

This method does not return the number of the line containing the error.

Input Parameters

None.

Output Parameters

pStatus	V+ error code

Related Topics

ChangeLine

DeleteLine

Edit

InsertLine

ClearBreakpoint Method

Visual Basic 6

```
Programs.ClearBreakpoint(pCommunications as Communications, bstrProgram as String,
                nLine as Long, pStatus as Long)
```

Visual Basic .NET

```
Programs.ClearBreakpoint(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                strProgram As String, ByVal nLine As Integer, ByRef pStatus As Integer)
```

C# .NET

```
Programs.ClearBreakpoint(ActiveV2Lib.Communications pCommunications, System.String
                strProgram, System.Int32 nLine, out System.Int32 pStatus)
```

Description

This method clears one or all breakpoints in a V+ program.

Input Parameters

bstrProgram	Name of V+ program
nLine	line number of the breakpoint (must be positive) if >0, a single breakpoint at that line is cleared if 0, all breakpoints in the specified V+ program are cleared.

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
BPT	BPT

Related Topics

ListBreakpoints

SetBreakpoint

Close Method

Visual Basic 6

```
Communications.Close()
```

Visual Basic .NET

```
Communications.Close()
```

C# .NET

```
Communications.Close()
```

Description

Closes a connection.

Input Parameters

None.

Output Parameters

None.

Examples

Visual Basic 6 Visual Basic .NET C# .NET

Visual Basic 6

```
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    Dim stat As Long

    ' close communications to controller
    On Error Resume Next
    comm.RequestEvents 0, stat
```

```
    comm.Close

    End ' End application and free-up all memory
End Sub
```

Visual Basic .NET

```
'Check all cases to properly close communications with the controller.
Protected Sub CloseConnection()
    Dim nStatus As Integer

    If Not (mCommunications Is Nothing) Then
        If (mCommunications.IsOnline <> 0) Then
            mCommunications.RequestEvents(0, nStatus)
            mCommunications.Close()
        End If
    End If
    mCommunications = Nothing

    mnuLocationPanel.Enabled = False
    mnuDIOPanel.Enabled = False
    mnuAbout.Enabled = False

End Sub
```

C# .NET

```
/// Close the connection to the controller, if any exist.
/// <summary>
/// Check all cases to properly close communications with the controller.
/// </summary>
protected void CloseConnection()
{
     if (mCommunications != null)
     {
          if (mCommunications.IsOnline != 0)
          {
               int nStatus;
               mCommunications.RequestEvents(0, out nStatus);
               mCommunications.Close();
           }
     }
```

```
        mCommunications = null;

        mnuLocationPanel.Enabled = false;
        mnuDIOPanel.Enabled = false;
        mnuAbout.Enabled = false;
    }
```

Related Topics

ControllerIPAddress

IsOnline

OnAlive

Open

ControllerIPAddress Property

Visual Basic 6

```
Communications.ControllerIPAddress
```

Visual Basic .NET

```
Communications.ControllerIPAddress()
```

C# .NET

```
Communications.ControllerIPAddress
```

Description

Returns the IP address of the controller connected to this Communications object. The IP address is returned as a string that is formatted in dotted−decimal notation: 192.168.1.100.

Usage Considerations

This property is read−only.

The return type is a string with the format ###.###.###.### where # is a number.

Input Parameters

None

Output Parameters

None

Related Topics

Close

IsOnline

OnAlive

Open

Delete Method

Visual Basic 6

```
Programs.Delete(pCommunications as Communications, bstrProgName as String,
                pStatus as Long)
```

Visual Basic .NET

```
Programs.Delete(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                bstrProgName As String, ByRef pStatus As Integer)
```

C# .NET

```
Programs.Delete(ActiveV2Lib.Communications pCommunications, System.String
                bstrProgName, ref System.Int32 pStatus)
```

Description

This method deletes a V+ program using the V+ DELETE monitor command.

Usage Considerations

This method is not available for MicroV+.

An executing program cannot be deleted.

This method also deletes subroutines or variables referenced by the named program. To delete the named program but retain its variables and called subroutines, use the Deletep method.

Input Parameters

bstrProgName	Name of program to delete.

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
DELETE	DELETEP

Related Topics

Deletem

Deletep

DeleteLine

Visual Basic 6

```
EditorFuncs.DeleteLine(pCommunications as Communications, lLineNumber as Long,
                lNumberOfLines as Long, pStatus as Long)
```

Visual Basic .NET

```
EditorFuncs.DeleteLine(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                lLineNumber As Integer, ByVal lNumberOfLines As Integer, ByRef
                pStatus As Integer)
```

C# .NET

```
EditorFuncs.DeleteLine(ActiveV2Lib.Communications pCommunications, System.Int32
                lLineNumber, System.Int32 lNumberOfLines, out System.Int32 pStatus)
```

Description

This method deletes a line from a program.

Input Parameters

lLineNumber	Line number of line to delete
lNumberOfLines	Number of lines to delete. May be 0.

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword

V+ Line Editor	MicroV+ Line Editor

Related Topics

ChangeLine

CheckProgram

Edit

InsertLine

Deletem Method

Visual Basic 6

```
Programs.Deletem(pCommunications as Communications, bstrModuleName as String,
                pStatus as Long)
```

Visual Basic .NET

```
Programs.Deletem(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                bstrModuleName As String, ByRef pStatus As Integer)
```

C# .NET

```
Programs.Deletem(ActiveV2Lib.Communications pCommunications, System.String
                bstrModuleName, out System.Int32 pStatus)
```

Description

This method deletes a V+ module using the V+ DELETEM monitor command.

Usage Considerations

This method is not available for MicroV+.

Input Parameters

bstrModuleName	Name of module to delete

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
DELETEM	N/A

Related Topics

Delete

Deletep

Deletep Method

Visual Basic 6

```
Programs.Deletep(pCommunications as Communications, bstrProgName as String,
                pStatus as Long)
```

Visual Basic .NET

```
Programs.Deletep(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                bstrProgName As String, ByRef pStatus As Integer)
```

C# .NET

```
Programs.Deletep(ActiveV2Lib.Communications pCommunications, System.String
                bstrProgName, ref System.Int32 pStatus)
```

Description

This method deletes a single V+ program using the V+ DELETEP monitor command.

Usage Considerations

An executing program cannot be deleted.

This method does not delete subroutines or variables referenced by the named program. To remove those components, use the Delete method.

Input Parameters

bstrProgName	Name of program to delete.

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
DELETEP	DELETEP

Related Topics

Delete

Deletem

Directory Method

Visual Basic 6

```
Programs.Directory(pCommunications as Communications, bstrProgName as String,
                pCount as Long, aProgram as ArrayLong, aAttribute as ArrayLong )
```

Visual Basic .NET

```
Programs.Directory(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                bstrProgName As String, ByRef pCount As Integer, ByRef aProgram As
                System.Array, ByRef aAttribute As System.Array)
```

C# .NET

```
Programs.Directory(ActiveV2Lib.Communications pCommunications, System.String
                bstrProgName, ref System.Int32 pCount, out System.Array aProgram,
                out System.Array aAttribute)
```

Description

This method obtains a listing of programs loaded in V+ memory. The V+ monitor command executed is DIRECTORY.

Usage Considerations

Wildcard matching can be specified using the * character. For instance, "ldb.*" selects all programs that begin with the characters "ldb.". Only simple wildcard−matching may be specified; for instance, the wildcard string "ldb.*.2" would be too complex. See below for examples.

```
.dir
     .PROGRAM a.mcp_step()
     .PROGRAM err.reacte()
     .PROGRAM err.reacti()
     .PROGRAM err.txt()
     .PROGRAM mcp.menu($line1, $line2, chr.start, chr.stop, keys, button)
.dir e*.
.dir e*
     .PROGRAM err.reacte()
     .PROGRAM err.reacti()
     .PROGRAM err.txt()
.dir err.*
     .PROGRAM err.reacte()
     .PROGRAM err.reacti()
     .PROGRAM err.txt()
.dir *.*eac*
     .PROGRAM err.reacte()
     .PROGRAM err.reacti()
.dir *.*eac?
     .PROGRAM err.reacte()
     .PROGRAM err.reacti()
.dir *.?eac?
     .PROGRAM err.reacte()
     .PROGRAM err.reacti()
.dir ?.?eac?
     .PROGRAM err.reacte()
     .PROGRAM err.reacti()
.
```

Input Parameters

bstrProgName	Program name to use for the DIRECTORY monitor command. Its main purpose is to allow simple wildcard matching (see Usage Considerations), and will usually be an empty string (meaning return all program names).
pCount	Number of program names to be returned.

Output Parameters

pCount	Actual number of program names returned.
aProgram	ARRAY of program names
aAttribute	ARRAY of program attribute bit flags: bit 0 set => is modified bit 1 set => invalid program

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
DIRECTORY	DIRECTORY

Related Topics

MDirectory

Edit Method

Visual Basic 6

```
EditorFuncs.Edit(pCommunications as Communications, bstrProgName as String,
                 pStatus as Long)
```

Visual Basic .NET

```
EditorFuncs.Edit(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                  bstrProgName As String, ByRef pStatus As Integer)
```

C# .NET

```
EditorFuncs.Edit(ActiveV2Lib.Communications pCommunications, System.String
                 bstrProgName, out System.Int32 pStatus)
```

Description

This method initializes the editor interface for editing a particular program. See the links to the corresponding V+ or MicroV+ keywords for more information.

Input Parameters

bstrProgName	Name of program to be edited.

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
V+ Line Editor	MicroV+ Line Editor

Related Topics

ChangeLine

CheckProgram

DeleteLine

InsertLine

Evaluate Method

Visual Basic 6

```
Status.Evaluate(pCommunications as Communications, lTask as Long,
                bstrProgram as String, bstrExpression as String,
                pType as Long, bstrValue as String, pStatus as Long)
```

Visual Basic .NET

```
Status.Evaluate(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                lTask As Integer, ByVal bstrProgram As String, ByVal bstrExpression
                As String, ByRef pType As Integer, ByRef bstrValue As String,
                ByRef pStatus As Integer)
```

C# .NET

```
Status.Evaluate(ActiveV2Lib.Communications pCommunications, System.Int32 lTask,
                System.String bstrProgram, System.String bstrExpression,
                out System.Int32 pType, out System.String bstrValue,
                out System.Int32 pStatus)
```

Description

This method evaluates an arbitrary V+ expression. An expression can be a variable, a literal value or a function. Only one result is returned at a time.

Usage Considerations

This method is not available for MicroV+.

Input Parameters

bstrExpression	String containing expression to evaluate.
lTask	Task number for context. A value of −1 indicates that no task is specified.
bstrProgram	Program name for context (can be an empty string).

Output Parameters

pType	0 if invalid expression 1 if real expression 2 if string expression 3 if trans expression 4 if ppoint expression
bstrValue	Resulting value of expression as a string.
pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Examples

Visual Basic .NET

Visual Basic .NET

```
Dim mStatus As ActiveV2Lib.StatusClass = New ActiveV2Lib.StatusClass
Dim nReturnedType As Integer
Dim sReturnedValue As String
Dim fResult As Double
Dim nStatus As Integer

mStatus.Evaluate(mCommunications, -1, "", "SQRT(144)", nReturnedType, sReturnedValue, nStatus)


Select Case nReturnedType
    Case 1  ' Real
        fResult = Double.Parse(sReturnedValue)
    Case Else
        Throw New Exception("Invalid returned type")
End Select
```

Execute Method

Visual Basic 6

```
Programs.Execute(pCommunications as Communications, bstrProgramName as String,
                lTaskNumber as Long, pStatus as Long)
```

Visual Basic .NET

```
Programs.Execute(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                bstrProgramName As String, ByVal lTaskNumber As Integer, ByRef
                pStatus As Integer)
```

C# .NET

```
Programs.Execute(ActiveV2Lib.Communications pCommunications, System.String
                bstrProgramName, System.Int32 lTaskNumber, out System.Int32
                pStatus)
```

Description

This method executes a V+ program. See the links to the corresponding V+ or MicroV+ keywords for more information.

Input Parameters

bstrProgramName	Name of program, and parameters if any (specified within parenthesis)
lTaskNumber	Task number to use

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Examples

Visual Basic 6 Visual Basic .NET C# .NET

Visual Basic 6

```
' start each program in the associated task
On Error Resume Next
With rec
Do While Not .EOF
    Set prog = New ActiveV2Lib.Programs
    Call prog.Execute(comm, !Program, !task, stat)
    If stat <> VE_SUCCESS Or Err.Number <> 0 Then
        If stat <> VE_SUCCESS Then
            Call errh.GetErrorString(stat, str)
        Else
            Call errh.GetErrorString(Err.Number, str)
        End If
        Call MsgBox(str, vbOKOnly, "Error attempting to Start'" _
        & !Program & "()'")
    End If
    .MoveNext
    Loop
End With
```

Visual Basic .NET

```
' Event-handler called by the system when the user clicks the
' Start button. This will start all configured programs.
Private Sub StartButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles StartButton.Click

    Dim nStatus As Integer
    Dim i As Integer

    Try
        ' Clear the text box.
        OutputTextBox.Clear()

        ' Go through the arrays to know what to start.
        For i = 0 To mnProgramCount
```

```
            ' Start only non-background tasks.
            If mProgramDef(i).bBackGround = False Then
                mPrograms.Execute(mCommunications, mProgramDef(i).sName, mProgramDef(i).nTask, nStatus)
                CheckError(nStatus)
            End If
        Next
    Catch Ex As System.Runtime.InteropServices.COMException
        ShowError(Ex)
    Catch Ex As Exception
        ShowError(Ex)
    End Try

End Sub
```

C# .NET

```
    /// <summary>
    /// Called when the user clicks the Start button.
    /// Start all programs flagged to run in the foreground.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void StartButton_Click(object sender, System.EventArgs e)
    {
         int nStatus;

         try
         {
              // Clear the text box
              OutputTextBox.Clear();


              // Go through the arrays to know what to start
              for(int i = 0; i < mnProgramCount; i++)
              {
                   // Start only non-background tasks
                   if (mProgramDef[i].bBackGround == false)
                   {
                        mPrograms.Execute(mCommunications, mProgramDef[i].sName, mProgramDef[i].nTask, out nStatus);
                        CheckError(nStatus);
                   }
              }
         }
```

```
        catch (System.Runtime.InteropServices.COMException Ex)
        {
             ShowError(Ex);
        }
        catch (Exception Ex)
        {
             ShowError(Ex);
        }
    }
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
EXECUTE	EXECUTE

Related Topics

Kill

Prime

Proceed

ZERO

FCopy Method

Visual Basic 6

```
Files.FCopy(pCommunications as Communications, bstrSourceFile as String,
                bstrDestFile as String, pStatus as Long)
```

Visual Basic .NET

```
Files.FCopy(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                bstrSourceFile As String, ByVal bstrDestFile As String, ByRef
                pStatus As Integer)
```

C# .NET

```
Files.FCopy(ActiveV2Lib.Communications pCommunications, System.String
                bstrSourceFile, System.String bstrDestFile, ref System.Int32
                pStatus)
```

Description

This method copies one file to another. If the destination file already exists, an error is returned.

Usage Considerations

This method is not available for MicroV+.

The V+ file and path name specifications must followed. Otherwise, the operation will fail. For details on V+ file and directory name specifications, see the ***V+ Operating System User's Guide***.

Input Parameters

bstrSourceFile	name of source file
bstrDestFile	name of destination file

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
FCOPY	N/A

Related Topics

FDelete

FDirectory

FRename

NFSMounts

FDelete Method

Visual Basic 6

```
Files.FDelete(pCommunications as Communications, bstrFileName as String,
               pStatus as Long)
```

Visual Basic .NET

```
Files.FDelete(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
               bstrFileName As String, ByRef pStatus As Integer)
```

C# .NET

```
Files.FDelete(ActiveV2Lib.Communications pCommunications, System.String
               bstrFileName, ref System.Int32 pStatus)
```

Description

This method deletes a file.

Usage Considerations

This method is not available for MicroV+.

Input Parameters

bstrFileName	Name of the file to delete.

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
FDELETE	N/A

Related Topics

FCopy

FDirectory

FRename

NFSMounts

FDirectory Method

Visual Basic 6

```
Files.FDirectory(pCommunications as Communications, bstrPathName as String,
                pCount as Long, aFileName as ArrayString , aAttribute as
                ArrayLong, aDate as ArrayDate, aSize as ArrayLong)
```

Visual Basic .NET

```
Files.FDirectory(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                bstrPathName As String, ByRef pCount As Integer, ByRef aFileName
                As System.Array, ByRef aAttribute As System.Array, ByRef aDate As
                System.Array, ByRef aSize As System.Array)
```

C# .NET

```
Files.FDirectory(ActiveV2Lib.Communications pCommunications, System.String
                bstrPathName, ref System.Int32 pCount, out System.Array aFileName,
                out System.Array aAttribute, out System.Array aDate, out
                System.Array aSize)
```

Description

This method obtains a directory listing.

Usage Considerations

This method is not available for MicroV+.

Input Parameters

bstrPathName	pathname to use for FDIRECTORY monitor command
pCount	number of file/directory names to be returned

Output Parameters

pCount	actual number of file/directory names returned
aFileName	ARRAY of file/directory names
aAttribute	ARRAY of file attribute bit flags:

Bit	Description
0	item is a directory
1	item is read−only
2	item is a link
3	item is protected
4	item is a system file
5	item is a volume label

aDate	ARRAY of file/directory creation dates
aSize	ARRAY of file sizes

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
FDIRECTORY	N/A

Related Topics

FCopy

FDelete

FRename

NFSMounts

FRename Method

Visual Basic 6

```
Files.FRename(pCommunications as Communications, bstrOldName as String,
               bstrNewName as String, pStatus as Long)
```

Visual Basic .NET

```
Files.FRename(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
               bstrOldName As String, ByVal bstrNewName As String, ByRef
               pStatus As Integer)
```

C# .NET

```
Files.FRename(ActiveV2Lib.Communications pCommunications, System.String
               bstrOldName, System.String bstrNewName, ref System.Int32
               pStatus)
```

Description

This method renames a file. An error is returned if:

- the existing file specified is not found
- the new name specified is already assigned to another file
- the new name contains illegal characters or too many characters. For details on V+ file and directory name specifications, see the ***V+ Operating System User's Guide***.

Usage Considerations

This method is not available for MicroV+.

Input Parameters

bstrOldName	Name of existing file
bstrNewName	New file name

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
FRENAME	N/A

Related Topics

FCopy

FDelete

FDirectory

NFSMounts

Free

Visual Basic 6

```
Status.Free(pCommunications as Communications, PrgMem as Single,
                GrMem as Single, VisMem as Single, ModMem as Single)
```

Visual Basic .NET

```
Status.Free(ByVal pCommunications As ActiveV2Lib.Communications,
                ByRef PrgMem As Single, ByRef GrMem As Single, ByRef VisMem
                As Single, ByRef ModMem As Single)
```

C# .NET

```
Status.Free(ActiveV2Lib.Communications pCommunications, out System.Single
                PrgMem, out System.Single GrMem, out System.Single VisMem,
                out System.Single ModMem)
```

Description

This method returns the amount of available memory on the V+ controller.

Output Parameters:

PrgMem	Program memory
GrMem	Graphics memory
VisMem	Vision memory
ModMem	Memory for models

Corresponding V+ Keyword	Corresponding MicroV+ Keyword

FREE	FREE

Related Topics

ZERO

GetErrorString Method

Visual Basic 6

```
ErrorHandler.GetErrorString(lErrorCode as Long, pbstrErrorString as String)
```

Visual Basic .NET

```
ErrorHandler.GetErrorString(ByVal lErrorCode As Integer, ByRef
                pbstrErrorString As String)
```

C# .NET

```
ErrorHandler.GetErrorString(System.Int32 lErrorCode, out System.String
                pbstrErrorString)
```

Description

This method returns the error string when given an error number. Errors are categorized by type:

- Positive number = Success
- Negative number = Error
- 0 = Undefined error

Any error code outside the legal range is considered an "Unknown error".

Input Parameters

lErrorCode	error number

Output Parameters

pbstrErrorString	error description string

Examples

Visual Basic 6 Visual Basic .NET C# .NET

Visual Basic 6

```
Set mctrl = New ActiveV2Lib.MiscControl

On Error Resume Next
Call mctrl.Speed(comm, hscMonitorSpeed.Value, stat)
If stat <> VE_SUCCESS Or Err.Number <> 0 Then
    If stat <> VE_SUCCESS Then
        Call errh.GetErrorString(stat, str)
    Else
        Call errh.GetErrorString(Err.Number, str)
    End If
    Call MsgBox(str, vbOKOnly, "Error attempting to set Monitor Speed")
End If
```

Visual Basic .NET

```
' Check the value of the status to display the proper error, if required.
Private Sub CheckError(ByVal nStatus As Long)
    If nStatus <> 0 Or nStatus <> ActiveV2Lib.tagERRORS.VE_SUCCESS Then
        Dim message As String
        mError.GetErrorString(nStatus, message)
        MessageBox.Show(Me, message, "Error returned by V+", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End If
End Sub
```

C# .NET

```
   /// <summary>
   /// Check the value of the status to display the proper error, if required.
   /// </summary>
   /// <param name="nStatus">status code returned by the controller</param>
   protected void CheckError(int nStatus)
   {
        if (nStatus != 0 ||
             nStatus != (int) ActiveV2Lib.tagERRORS.VE_SUCCESS)
        {
```

```
            string sMsg;
            mError.GetErrorString(nStatus, out sMsg);
            MessageBox.Show(this, sMsg, "Application Error");
        }
    }
```

Related Topics

GetVErrorString

GetStringEx Method

Visual Basic 6

```
Communications.GetStringEx(nBoard as Long, nChannel as Long, bstrExpected as String,
                pbstrString as String, pCount as Long, pContinue as Long)
```

Visual Basic .NET

```
Communications.GetStringEx(ByVal nBoard As Integer, ByVal nChannel As Integer, ByVal
                bstrExpected As String, ByRef pbstrString As String, ByRef
                pCount As Integer, ByRef pContinue As Integer)
```

C# .NET

```
Communications.GetStringEx(System.Int32 nBoard, System.Int32 nChannel, System.String
                bstrExpected, out System.String pbstrString, out System.Int32 pCount,
                out System.Int32 pContinue)
```

Description

Reads a string that is output by a V+ TYPE or WRITE instruction. The string is read from the specified V+ board/task.

Usage Considerations

This method should only be called to read program output within the OnProgramOutput event processing method. Calling this method at other times will block program execution. The OnProgramOutput event passes the board and task number when invoked, and these numbers should be passed to this method.

Input Parameters

nBoard	Specifies the board number (processor number), which is usually 1 to indicate the main CPU, for the device to read from.
nChannel	Specifies the user channel number (task number) to read from.
bstrExpected	Specifies the terminator string to wait for. Set to the empty string (i.e., "").

Output Parameters

pbstrString	Pointer to string from V+.
pCount	Pointer to number of bytes received from V+.
pContinue	Pointer to flag that is true if more data must be read from V+ as part of the same transaction. In other words, the caller must continue to GetString until pContinue returns false.

Examples

Visual Basic 6 Visual Basic .NET C# .NET

Visual Basic 6

```
Private Sub comm_OnProgramOutput(ByVal nBoard As Long, ByVal nTask As Long)
    Dim pbstrString As String
    Dim pcount As Long
    Dim pContinue As Long

    ' loop until no more text associated with this event (bContinue false)
    Do
        Call comm.GetStringEx(nBoard, nTask, "", pbstrString, pcount, pContinue)


        ' truncate string to nearest CR+LF after BUFFER_SIZE
        txtProgramOutput.Text = Right(txtProgramOutput.Text + pbstrString, BUFFER_SIZE)
        txtProgramOutput.Text = Mid(txtProgramOutput.Text, InStr(txtProgramOutput.Text, Chr(13)))


    Loop While pContinue

End Sub
```

Visual Basic .NET

```
' Event-handler called by ActiveV when a V+ program generates
' some text output. Display it in the Output text box.
Private Sub CommProgramOutput(ByVal nBoard As Integer, ByVal nTask As Integer) Handles mCommunications.OnProgramOutput
```

```
    Dim sMsg As String
    Dim count As Integer
    Dim continue As Integer

    Do
        mCommunications.GetStringEx(nBoard, nTask, "", sMsg, count, continue)
        sMsg = OutputTextBox.Text + sMsg
        If sMsg.Length > 500 Then
            sMsg = sMsg.Substring(sMsg.IndexOf(vbCrLf) + 2)
        End If
        OutputTextBox.Text = sMsg
    Loop While continue

End Sub
```

C# .NET

```
/// <summary>
/// Called by the Communication component. This handler displays the information
/// from the controller.
/// </summary>
/// <param name="nBoard">Board Number</param>
/// <param name="nTask">Task Number</param>
private void mCommunications_OnProgramOutput(int nBoard, int nTask)
{
     string sMsg;
     int nCount;
     int nContinue;

     do
     {
          mCommunications.GetStringEx(nBoard, nTask, "", out sMsg, out nCount, out nContinue);
          sMsg = OutputTextBox.Text + sMsg;
          if (sMsg.Length > 500)
          {
               sMsg = sMsg.Substring(sMsg.IndexOf("\n") + 2);
          }
          OutputTextBox.Text = sMsg;
     }
     while (nContinue != 0);
}
```

Related Topics

OnProgramOut Event

SendStringEx

GetVErrorString Method

Visual Basic 6

```
ErrorHandler.GetVErrorString(lVErrorCode as Long, bstrErrorString as String)
```

Visual Basic .NET

```
ErrorHandler.GetVErrorString(ByVal lVErrorCode As Integer, ByRef
                bstrErrorString As String)
```

C# .NET

```
ErrorHandler.GetVErrorString(System.Int32 lVErrorCode, out System.String
                bstrErrorString)
```

Description

This method converts a V+ error code into a descriptive string. Unlike GetErrorString which interprets all error codes, this method accepts only V+ error codes.

Input Parameters

lVErrorCode	V+ error code

Output Parameters

bstrErrorString	V+ error string

Examples

Visual Basic 6

Visual Basic 6

```
' instantiate Communications object
Set comm. = New ActiveV2lib.Communications

' Calibrate robot
```

```
Call mctrl.Calibrate(comm, stat)

' Check returning status for error condition
If stat <> VE_SUCCESS Then
     ' Extract error description
     Call errh.GetVErrorString(stat, str)
     Call MsgBox(str, vbOKOnly, "Error attempting to Calibrate Robot")
End if
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
$ERROR	N/A

Related Topics

GetErrorString

V+ Error Messages

GetVersion

Visual Basic 6

```
Version.GetVersion(pVersion as Long, pRevision as Long, pBstrEdit as String,
                pBstrVersion$tring as String
```

Visual Basic .NET

```
Version.GetVersion(ByRef pVersion As Integer, ByRef pRevision As Integer,
                ByRef pBstrEdit As String, ByRef pBstrVersionString As String)
```

C# .NET

```
Version.GetVersion(out System.Int32 pVersion, out System.Int32 pRevision,
                out System.String pBstrEdit, out System.String
                pBstrVersionString)
```

Description

This method returns the version of the ActiveV library. The version appears in the form of:

<version>.<revision> <edit>

For example:

1.1 A1

2.0 B

Output Parameters:

pVersion	version
pRevision	revision

pBstrEdit	edit (e.g., "A1" or "B")
pBstrVersionString	entire edit string (e.g., "1.1 (Edit A1, 30−Sep−2001)")

Related Topics

GetVplusVersion

GetVplusVersion

Visual Basic 6

```
Version.GetVplusVersion(pCommunications as Communications, pVersion as Long,
                pRevision as Long, pBstrEdit as String, pBstrEditString as String
```

Visual Basic .NET

```
Version.GetVplusVersion(ByVal pCommunications As ActiveV2Lib.Communications,
                ByRef pVersion As Integer, ByRef pRevision As Integer, ByRef pBstrEdit
                As String, ByRef pBstrEditString As String)
```

C# .NET

```
Version.GetVplusVersion(ActiveV2Lib.Communications pCommunications, out System.Int32
                pVersion, out System.Int32 pRevision, out System.String pBstrEdit,
                out System.String pBstrEditString )
```

Description

This method returns the version of the connected V+ system. The version appears in the form of:

<version>.<revision> <edit>

For example:

14.1 A1

15.0 B

Output Parameters:

pVersion	version

pRevision	revision
pBstrEdit	edit (e.g. "A1" or "B")
pBstrEditString	entire edit string (e.g. "14.1 (Edit A1, 30−Sep−2001)")

Examples

Visual Basic 6 Visual Basic .NET C# .NET

Visual Basic 6

```
    ' display Cookie Demo V+ application software version number
    Call status.Lists(comm, 0, "", "$ckd.version", 1, strVarName(), strVarContent(), stat)
    lblVAppVersion.Caption = strVarContent(1)


    Set version = New ActiveV2Lib.version
    version.GetVplusVersion comm, stat, stat, versionString, fullVersion
    lblID.Caption = fullVersion
```

Visual Basic .NET

```
    'Display the V+ version number.
    Dim Version As ActiveV2Lib.Version

    Version = New ActiveV2Lib.Version

    Dim sFullVersion As String
    Dim sVersion As String
    Dim nRev As Integer
    Dim nVersion As Integer

Version.GetVplusVersion(mCommunications, nVersion, nRev, sVersion, sFullVersion)
    VVersion.Text = sFullVersion
```

C# .NET

```
// Display the V+ version number.
ActiveV2Lib.Version Version = new ActiveV2Lib.Version();
string sFullVersion;
string sVersion;
int nRev;
int nVersion;

Version.GetVplusVersion(mCommunications, out nVersion, out nRev, out sVersion, out sFullVersion);
VVersion.Text = sFullVersion;
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
ID	ID

Related Topics

GetVersion

Here Method

Visual Basic 6

```
Status.Here(pCommunications as Communications, lRobotNumber as Long,
                bstrVariableName as String)
```

Visual Basic .NET

```
Status.Here(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                lRobotNumber As Integer, ByVal bstrVariableName As String)
```

C# .NET

```
Status.Here(ActiveV2Lib.Communications pCommunications, System.Int32
                lRobotNumber, System.String bstrVariableName)
```

Description

This method sets a variable to the value of the current location.

Input Parameters

lRobotNumber	Robot number to use for current location
bstrVariableName	Variable name

Output Parameters

None.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
HERE	HERE

Related Topics

Where

InsertLine Method

Visual Basic 6

```
EditorFuncs.InsertLine(pCommunications as Communications, lLineNumber as Long,
                pBstrLineText as String, pStatus as Long)
```

Visual Basic .NET

```
EditorFuncs.InsertLine(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                lLineNumber As Integer, ByRef pBstrLineText As String, ByRef
                pStatus As Integer)
```

C# .NET

```
EditorFuncs.InsertLine(ActiveV2Lib.Communications pCommunications, System.Int32
                lLineNumber, ref System.String pBstrLineText, out System.Int32
                pStatus)
```

Description

This method inserts a new line in the current program at the specified line number. The contents of the specified line number and following are moved down one line to make room for the inserted line.

Input Parameters

lLineNumber	line number at which to insert new line
pBstrLineText	text for new line

Output Parameters

pBstrLineText	text for new line as formatted by V+
pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
V+ Line Editor	MicroV+ Line Editor

Related Topics

ChangeLine

CheckProgram

DeleteLine

Edit

InsertLine

IO Method

Visual Basic 6

```
Status.IO(pCommunications as Communications, lNumber as Long,
                aValid as ArrayLong, aState as ArrayLong)
```

Visual Basic .NET

```
Status.IO(ByVal pCommunications As ActiveV2Lib.Communications,
                ByVal lNumber As Integer, ByRef aValid As System.Array,
                ByRef aState As System.Array)
```

C# .NET

```
Status.IO(ActiveV2Lib.Communications pCommunications, System.Int32
                lNumber, out System.Array aValid, out System.Array
                aState)
```

Description

This method samples V+ digital IO signals.

Input Parameters

lNumber	0 means get signals from 1 to 512 1 means get signals from 1001 to 1512 2 means get signals from 2001 to 2512 3 means get signals from 3001 to 3512

Output Parameters

aValid	Array of 512 longs. Element <n> non−zero means that the corresponding signal is valid. For instance, if lNumber is 1 and <n> is 3, then aValid[3] being non−zero means that signal number 1003 is valid.
aState	Array of 512 longs. Element <n> non−zero means that the corresponding signal is on,

else the signal is off.

Examples

Visual Basic 6 Visual Basic .NET C# .NET

Visual Basic 6

```
 Private Sub tmrRefresh_Timer()
     Dim status As ActiveV2Lib.status
     Dim lngIOValid() As Long
     Dim lngIOState() As Long


     Set status = New ActiveV2Lib.status


 ' read and buffer outputs
     Call status.IO(comm, 0, lngIOValid(), lngIOState()) 'sigs 1 to 512
     For ii = 1 To 512
        iovalid(ii) = lngIOValid(ii)
        iostate(ii) = lngIOState(ii)
     Next ii
```

Visual Basic .NET

```
' Read the address from 1001 to 1512 (Input).
mStatus.IO(mCommunications, 0, IOValid, IOState)
For i = IOValid.GetLowerBound(0) To IOValid.GetUpperBound(0)
    ' Only add the valid one in the combo box.
    If IOValid(i) <> 0L Then
        nAddress = i + 1000
        If ((i - 1) Mod nBit) = 0 Then
            cmbAddress.Items.Add(nAddress.ToString())
        End If
    End If
Next
```

C# .NET

```
// Read the address from 1001 to 1512 (Input).
mStatus.IO(mCommunications, 0, out IOValid, out IOState);
for(i = IOValid.GetLowerBound(0); i <= IOValid.GetUpperBound(0); i++)
{
     // Only add the valid one in the combo box.
     if ((int) IOValid.GetValue(i) != 0)
     {
          nAddress = i + 1000;
          if (((i - 1) % nBit) == 0)
          {
               cmbAddress.Items.Add(nAddress.ToString());
          }
     }
}
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
IO	IO

Related Topics

Signal8

SignalOff

SignalOn

IsGetc Property

Visual Basic 6

```
Communications.IsGetc
```

Visual Basic .NET

```
Communications.IsGetc() As Integer
```

C# .NET

```
Communications.IsGetc
```

Description

Returns TRUE if the V+ program executed a GETC() instruction to obtain input.

Usage Considerations

This property is read–only.

Input Parameters

None

Output Parameters

None

Details

The IsGetc property returns TRUE if the V+ program executed a GETC() instruction to obtain input. This knowledge is needed because a V+ program behaves differently depending on whether a READ( ) or a GETC( ) instruction is executed. In the case of a READ( ) instruction, input is not considered complete until a <return> is typed. However, in the case of a GETC( ) instruction, every character is handled by the V+ program as soon as it is typed. This property is useful for interacting with a legacy V+ utility.

IsOnline Property

Visual Basic 6

```
Communications.IsOnline as Long
```

Visual Basic .NET

```
Communications.IsOnline() As Integer
```

C# .NET

```
Communications.IsOnline
```

Description

This property returns a value to indicate whether the PC is connected to V+:

1 = connected

0 = not connected

Usage Considerations

This property is read–only.

Input Parameters

None

Output Parameters

None

Examples

Visual Basic .NET C# .NET

Visual Basic .NET

```
'Check all cases to properly close communications with the controller.
Protected Sub CloseConnection()
    Dim nStatus As Integer

    If Not (mCommunications Is Nothing) Then
        If (mCommunications.IsOnline <> 0) Then
            mCommunications.RequestEvents(0, nStatus)
            mCommunications.Close()
        End If
    End If
    mCommunications = Nothing

End Sub
```

C# .NET

```
/// Check all cases to properly close communications with the controller.
/// </summary>
protected void CloseConnection()
{
     if (mCommunications != null)
     {
          if (mCommunications.IsOnline != 0)
          {
               int nStatus;
               mCommunications.RequestEvents(0, out nStatus);
               mCommunications.Close();
          }
     }
     mCommunications = null;

}
```

Related Topics

Close

ControllerIPAddress

OnAlive

Open

Kill Method

Visual Basic 6

```
Programs.Kill(pCommunications as Communications, lTaskNumber as Long,
               pStatus as Long)
```

Visual Basic .NET

```
Programs.Kill(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
               lTaskNumber As Integer, ByRef pStatus As Integer)
```

C# .NET

```
Programs.Kill (ActiveV2Lib.Communications pCommunications, System.Int32
               lTaskNumber, out System.Int32 pStatus)
```

Description

This method clears a program execution stack. Invalid task numbers are ignored.

Usage Considerations

KILL cannot be used while the specified program task is executing.

KILL has no effect if the specified task execution stack is empty.

Input Parameters

lTaskNumber	Task number

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
KILL	KILL

Related Topics

Abort

Execute

Prime

Proceed

ZERO

ListBreakpoints Method

Visual Basic 6

```
Programs.ListBreakpoints(pCommunications as Communications, bstrProgramName as String,
                aBreakpoints as ArrayLong, pCount as Long, pStatus as Long)
```

Visual Basic .NET

```
Programs.ListBreakpoints(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                bstrProgramName As String, ByRef aBreakpoints As System.Array,
                ByRef pCount As Integer, ByRef pStatus As Integer)
```

C# .NET

```
Programs.ListBreakpoints(ActiveV2Lib.Communications pCommunications, System.String
                bstrProgramName, out System.Array aBreakpoints, out System.Int32
                pCount, out System.Int32 pStatus)
```

Description

This method lists all breakpoints in a V+ program.

Input Parameters

bstrProgram	Name of V+ program.

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.
aBreakpoints	An array of program line numbers, each of which contains a breakpoint.
pCount	Number of entries in the aBreakpoints array

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
LISTB	LISTB

Related Topics

ClearBreakpoint

SetBreakpoint

Listl Method

Visual Basic 6

```
Status.Listl(pCommunications as Communications, lTask as Long,
               bstrProgram as String, bstrLocationName as String,
               pCount as Long, aLocation as ArraySingle, pStatus as Long)
```

Visual Basic .NET

```
Status.Listl(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
               lTask As Integer, ByVal bstrProgram As String, ByVal bstrLocationName
               As String, ByRef pCount As Integer, ByRef aLocation As System.Array,
               ByRef pStatus As Integer)
```

C# .NET

```
Status.Listl(ActiveV2Lib.Communications pCommunications, System.Int32 lTask,
               System.String bstrProgram, System.String bstrLocationName,
               out System.Int32 pCount, out System.Array aLocation,
               out System.Int32 pStatus)
```

Description

This method returns the value of a location variable.

Usage Considerations

This method is not available with MicroV+.

For array variables, you must explicitly specify the "[]" characters.

The locations will be returned as packs of six values. Therefore, in the aLocation array, the first pack of six values will be the first transform, the second pack of six values will be the second transform, and so on.

Input Parameters

bstrLocationName	Location name, wildcards not recognized
lTask	Task number for context. A value of −1 indicates that no task is specified.
bstrProgram	Program name for context (can be an empty string).

Output Parameters

pCount	Number of coordinates or joints returned (up to 12).
pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.
aLocation	Array of coordinate/joint values.

Example

Visual Basic 6

```
Private Sub getLocation(ByVal strLocationName As String, sngLoc() As Single)
    Dim status As ActiveV2Lib.status
    Dim lngpCount As Long
    Dim ii As Integer
    Dim stat As Long

    Set status = New ActiveV2Lib.status

   ' read location value
    Call status.Listl(comm, -1, "", strLocationName, lngpCount, sngLoc(), stat)

    If lngpCount > 0 Then   ' location exists
        ' for each coordinate
        For ii = 1 To 6
            txtLoc(ii - 1).Text = sngLoc(ii)
        Next ii
        vscZ.Enabled = True ' display jog tools
        vscX.Enabled = True
        hscY.Enabled = True
    Else    ' location doesn't exist or can't be read
        For ii = 1 To 6
```

```
            txtLoc(ii - 1).Text = "?"
        Next ii
        vscZ.Enabled = False ' gray out jog tools
        vscX.Enabled = False
        hscY.Enabled = False
    End If

End Sub
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
LISTL	LISTL

Related Topics

Listp

Listr

Lists

ListsB

Lvariables

Rvariables

Svariables

Listp Method

Visual Basic 6

```
Programs.Listp(pCommunications as Communications, bstrProgramName as String,
               pCount as Long, lStartingLine as Long, aText as ArrayString)
```

Visual Basic .NET

```
Programs.Listp(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
               bstrProgramName As String, ByRef pCount As Integer, ByVal
               lStartingLine As Integer, ByRef aText As System.Array)
```

C# .NET

```
Programs.Listp(ActiveV2Lib.Communications pCommunications, System.String
               bstrProgramName, System.Int32 pCount, System.Int32
               lStartingLine, out System.Array aText)
```

Description

This method returns the text of a program that has been loaded in V+ memory. Requesting a non−existent program will cause an empty array to be returned. All programs can be listed by supplying an empty program name.

Input Parameters

bstrProgramName	Name of program. If empty, all programs are listed.
pCount	Maximum number of lines of text to return.

Output Parameters

pCount	Number of lines of text actually returned.
lStartingLine	First line returned (starting from 1).
aText	Array of text, one for each line.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
LISTP	LISTP

Related Topics

Listl

Listr

Lists

ListsB

Listr Method

Visual Basic 6

```
Status.Listr(pCommunications as Communications, lTask as Long,
               bstrProgram as String, bstrVariableName as String,
               pCount as Long, aVariableName as ArrayString,
               aVariableValue as ArraySingle, pStatus as Long)
```

Visual Basic .NET

```
Status.Listr(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
               lTask As Integer, ByVal bstrProgram As String, ByVal
               bstrVariableName As String, ByRef pCount As Integer, ByRef
               aVariableName As System.Array, ByRef aVariableValue As
               System.Array, ByRef pStatus As Integer)
```

C# .NET

```
Status.Listr(ActiveV2Lib.Communications pCommunications, System.Int32 lTask,
               System.String bstrProgram, System.String bstrVariableName,
               ref System.Int32 pCount, out System.Array aVariableName,
               out System.Array aVariableValue, out System.Int32 pStatus)
```

Description

This method returns values of real V+ variables.

Input Parameters

bstrVariableName	Name of V+ real variable, or empty string to return all V+ real variables.
pCount	Maximum number of variables to return.
lTask	Task number for context. A value of −1 indicates that no task is specified.
bstrProgram	Program name for context (can be an empty string).

Output Parameters

pCount	Actual number of variables returned.
aVariableName	Array of variable names.
aVariableValue	Array of variable values.
pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Examples

Visual Basic 6 Visual Basic .NET C# .NET

Visual Basic 6

```
Private Sub comm_OnReadPosted(ByVal nBoard As Long, ByVal nTask As Long)
    Dim status As ActiveV2Lib.status
    Dim strMsgRName() As String
    Dim sngMsgRValue() As Single
    Dim strMsgSName() As String
    Dim strMsgSContent() As String
    Dim msgBoxResult As VbMsgBoxResult
    Dim strPrompt As String
    Dim msgBoxStyle As VbMsgBoxStyle
    Dim lngpCount As Long
    Dim str As String
    Dim stat As Long

    Set status = New ActiveV2Lib.status

    ' read the real and string arrays ckd.msg[] and $ckd.msg[]
    '  which contain message information
    lngpCount = 2
    Call status.Listr(comm, -1, "", "ckd.msg[]", lngpCount, strMsgRName(), sngMsgRValue(), stat)
    Call status.Lists(comm, -1, "", "$ckd.msg[]", lngpCount, strMsgSName(), strMsgSContent(), stat)

    If lngpCount > 0 Then
        strPrompt = strMsgSContent(1) & vbCrLf & strMsgSContent(2)
        msgBoxStyle = sngMsgRValue(2)
```

```
        ' pop-up message box
        msgBoxResult = MsgBox(strPrompt, msgBoxStyle, "Message from Controller")

        ' send response back to controller
        ' pack response value into 2-byte string
        Call comm.SendStringEx(nBoard, nTask, CStr(msgBoxResult) & vbCrLf)
    End If

End Sub
```

Visual Basic .NET

```
Dim names As System.Array
Dim values As System.Array
Dim numbers As System.Array
Dim status As Integer
Dim i As Integer

Try
    ' Read the status variable.
    mStatus.Listr(mCommunications, 0, "", msCkdStatus, 1, names, values, status)

    ' Cache it locally.
    mnAppStatus = values.GetValue(1)

    ' Update the LED color according to the bit values.
     For i = 0 To 6
         If (mnAppStatus And (2 ^ i)) > 0 Then
             mPanels(i).BackColor = Color.Red
         Else
             mPanels(i).BackColor = Color.Black
         End If
     Next
```

C# .NET

```
Array names;
Array values;
Array numbers;
int   nStatus;
int   nRef;
```

```
try
{
     // Read the status variable.
     nRef = 1;
     mStatus.Listr(mCommunications, 0, "", msCkdStatus, ref nRef, out names, out values, out nStatus);

     // Cache it locally.
     mnAppStatus = (int) ((float) values.GetValue(1));

     // Update the LED color according to the bit values.
     for (int i = 0; i < mnNumPanels; i++)
     {
          mPanels[i].BackColor = (((mnAppStatus & (2 ^ i)) > 0) ? Color.Red : Color.Black);
     }
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
LISTR	LISTR

Related Topics

Listl

Listp

Lists

ListsB

Lvariables

Rvariables

Svariables

Lists Method

Visual Basic 6

```
Status.Lists(pCommunications as Communications, lTask as Long,
                bstrProgram as String, BSTR bstrVariableName as String,
                Count as Long, aVariableName as ArrayString,
                aVariableContent as ArrayString, pStatus as Long)
```

Visual Basic .NET

```
Status.Lists(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                lTask As Integer, ByVal bstrProgram As String, ByVal
                bstrVariableName As String, ByRef pCount As Integer, ByRef
                aVariableName As System.Array, ByRef aVariableContent As
                System.Array, ByRef pStatus As Integer)
```

C# .NET

```
Status.Lists(ActiveV2Lib.Communications pCommunications, System.Int32 lTask,
                System.String bstrProgram, System.String bstrVariableName,
                ref System.Int32 pCount, out System.Array aVariableName,
                out System.Array aVariableContent, out System.Int32 pStatus)
```

Description

This method returns values of string V+ variables.

Usage Considerations

This method is not available for MicroV+.

Input Parameters

bstrVariableName	Name of V+ string variable, or empty string to return all V+ string variables.
pCount	Maximum number of variables to return.

lTask	Task number for context. A value of −1 indicates that no task is specified.
bstrProgram	Program name for context (can be an empty string).

Output Parameters

pCount	Actual number of variables returned.
aVariableName	Array of variable names.
aVariableContent	Array of variable contents.
pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Examples

Visual Basic 6 Visual Basic .NET C# .NET

Visual Basic 6

```
Private Sub comm_OnReadPosted(ByVal nBoard As Long, ByVal nTask As Long)
    Dim status As ActiveV2Lib.status
    Dim strMsgRName() As String
    Dim sngMsgRValue() As Single
    Dim strMsgSName() As String
    Dim strMsgSContent() As String
    Dim msgBoxResult As VbMsgBoxResult
    Dim strPrompt As String
    Dim msgBoxStyle As VbMsgBoxStyle
    Dim lngpCount As Long
    Dim str As String
    Dim stat As Long


    Set status = New ActiveV2Lib.status


    ' read the real and string arrays ckd.msg[] and $ckd.msg[]
    '  which contain message information
    lngpCount = 2
    Call status.Listr(comm, -1, "", "ckd.msg[]", lngpCount, strMsgRName(), sngMsgRValue(), stat)
```

```
    Call status.Lists(comm, -1, "", "$ckd.msg[]", lngpCount, strMsgSName(), strMsgSContent(), stat)

    If lngpCount > 0 Then
        strPrompt = strMsgSContent(1) & vbCrLf & strMsgSContent(2)
        msgBoxStyle = sngMsgRValue(2)


        ' pop-up message box
        msgBoxResult = MsgBox(strPrompt, msgBoxStyle, "Message from Controller")

        ' send response back to controller
        ' pack response value into 2-byte string
        Call comm.SendStringEx(nBoard, nTask, CStr(msgBoxResult) & vbCrLf)
    End If

End Sub
```

Visual Basic .NET

```
Dim Names As System.Array
Dim Content As System.Array
Dim nStatus As Integer

' Display Cookie Demo V+ application software version number.
mStatus.Lists(mCommunications, 0, "", "$ckd.version", 1, Names, Content, nStatus)
VAppVersion.Text = Content(1)
```

C# .NET

```
// Display Cookie Demo V+ application software version number.
nRef = 1;
mStatus.Lists(mCommunications, 0, "", "$ckd.version", ref nRef, out Names, out Content, out nStatus);
VAppVersion.Text = (string) Content.GetValue(Content.GetLowerBound(0));
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
LISTS	N/A

Related Topics

Listl

Listp

Listr

ListsB

Lvariables

Rvariables

Svariables

ListsB Method

Visual Basic 6

```
Status.ListsB(pCommunications as Communications, lTask as long,
               bstrProgram as string, bstrVariableName as string,
               pCount as long, aVariableName as ArrayString,
               aVariableContent as ArrayString, pStatus as Long)
```

Visual Basic .NET

```
Status.ListsB(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
               lTask As Integer, ByVal bstrProgram As String, ByVal
               bstrVariableName As String, ByRef pCount As Integer, ByRef
               aVariableName As System.Array, ByRef aVariableContent As
               System.Array, ByRef pStatus As Integer)
```

C# .NET

```
Status.ListsB(ActiveV2Lib.Communications pCommunications, System.Int32 lTask,
               System.String bstrProgram, System.String bstrVariableName,
               ref System.Int32 pCount, out System.Array aVariableName,
               out System.Array aVariableContent, out System.Int32 pStatus)
```

Description

This method returns the values of V+ string variables as binary values. The difference between this method and Lists is that Lists will escape non−printable characters. For example, the ASCII CR (carriage−return) character will be returned by Lists as the string "^m", whereas ListsB will return it as hexadecimal D.

Input Parameters

bstrVariableName	Name of V+ variable, or empty string to return all V+ string variables.
pCount	Maximum number of variables to return.
lTask	Task number for context. A value of −1 indicates that no task is specified.
bstrProgram	Program name for context (can be an empty string)

Output Parameters

pCount	Actual number of variables returned
aVariableName	Array of variable names
aVariableContent	Array of variable values
pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Related Topics

Listl

Listp

Listr

Lists

Load Method

Visual Basic 6

```
Programs.Load(pCommunications as Communications, bstrFileName as String,
               pStatus as Long, bstrErrorString as String)
```

Visual Basic .NET

```
Programs.Load(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
               bstrFileName As String, ByRef pStatus As Integer, ByRef
               bstrErrorString As String)
```

C# .NET

```
Programs.Load(ActiveV2Lib.Communications pCommunications, System.String
               bstrFileName, out System.Int32 pStatus, out System.String
               bstrErrorString)
```

Description

This method loads programs and variables from a file into V+ memory.

Usage Considerations

NOTE: When used with MicroV+, this method loads all programs and variables stored in NVRAM.

Input Parameters

bstrFileName	Name of file to load (must be an empty string when used with MicroV+)

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.
bstrErrorString	Error string containing name of program causing error.

Examples

Visual Basic 6 Visual Basic .NET C# .NET

Visual Basic 6

```
    ' Loop through query result and load all files
    On Error Resume Next
    With recFiles
        Do While Not .EOF
            Call prog.Load(comm, !Value, stat, strError)
                If (stat <> VE_SUCCESS And stat <> VE_PRGDEF) _
                Or Err.Number <> 0 Then
                    If stat <> VE_SUCCESS Then
                        Call errh.GetErrorString(stat, str)
                    Else
                        Call errh.GetErrorString(Err.Number, str)
                    End If
                    Call MsgBox(str & " " & strError, vbOKOnly, _
                    "Error attempting to Load File " & !Value)
                End If
            .MoveNext
        Loop
    End With
```

Visual Basic .NET

```
    'Load the V+ files.
    Dim nStatus As Integer
    Dim sError As String
    Dim i As Integer

mPrograms.Load(mCommunications, "\\demo\\cookie.v2", nStatus, sError)
    mPrograms.Load(mCommunications, "\\demo\\cookie.lc", nStatus, sError)
```

C# .NET

```
       //Load the V+ files.
       int    nStatus;
       string sError;
```

```
mPrograms.Load(mCommunications, "\\demo\\cookie.v2", out nStatus, out sError);
mPrograms.Load(mCommunications, "\\demo\\cookie.lc", out nStatus, out sError);
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
LOAD	LOAD

Related Topics

CheckProgram

Store

Storem

Storep

Lvariables Method

Visual Basic 6

```
Status.Lvariables(pCommunications as Communications, lTask as Long,
                bstrProgram as String, bstrName as String, pCount as Long,
                aNames as ArrayString)
```

Visual Basic .NET

```
Status.Lvariables(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                lTask As Integer, ByVal bstrProgram As String, ByVal bstrName
                As String, ByRef pCount As Integer, ByRef aNames As System.Array)
```

C# .NET

```
Status.Lvariables(ActiveV2Lib.Communications pCommunications, System.Int32 lTask,
                System.String bstrProgram, System.String bstrName, ref System.Int32
                pCount, out System.Array aNames)
```

Description

This method returns the names of location variables in an array. Only the variable names are returned.

Input Parameters

bstrName	Name (may contain wildcards), or empty string to get the names of all location variables.
pCount	Number of names to return.
lTask	Task number for context. A value of −1 indicates that no task is specified.
bstrProgram	Program name for context (can be an empty string).

Output Parameters

pCount	Actual number of names returned.
aNames	Array of names.

Examples

Visual Basic .NET C# .NET

Visual Basic .NET

```
' Get all the location variables from the controller and update the location data in the list control.
Protected Function RefreshList()

    Dim i As Integer
    Dim nIndex As Integer
    Dim nCount As Integer
    Dim nLocationIndex As Integer
    Dim nStatus As Integer
    Dim Locations As System.Array
    Dim Names As System.Array
    Dim sName As String
    Dim item As ListViewItem

    ' Empty the control list first.
    listLocation.Items.Clear()

    nCount = 10 ' Only get the first 10.

    Try
        'Get the list of location variables.
        mStatus.Lvariables(mCommunications, -1, "", "", nCount, Names)
    Catch Ex As System.Runtime.InteropServices.COMException
        ShowError(Ex)
        Exit Function
    Catch Ex As Exception
        ShowError(Ex)
```

```
        Exit Function
    End Try

    If nCount = 0 Then
        mLocation = Nothing
        Exit Function
    End If

...
```

C# .NET

```
/// Get all the location variables from the controller and update the location data in the list control.
/// </summary>
protected void RefreshList()
{
        int i;
        int nIndex;
        int nCount;
        int nLocationIndex;
        int nStatus;
        Array Locations;
        Array Names;
        string sName;


        // Empty the control list first.
        listLocation.Items.Clear();

        nCount = 10;    // Only request the first 10.
        try
        {
                // Get the list of location variables.
mStatus.Lvariables(mCommunications, -1, "", "", ref nCount, out Names);
        }
        catch (System.Runtime.InteropServices.COMException Ex)
        {
                ShowError(Ex);
                return;
        }
        catch (Exception Ex)
        {
                ShowError(Ex);
```

```
                    return;
                }

        ...
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
LISTL	LISTL

Related Topics

Listr

Listl

Lists

Rvariables

Svariables

MDirectory Method

Visual Basic 6

```
Programs.MDirectory(pCommunications as Communications, bstrModuleName as String,
               pCount as Long, aNames as Array, aAttributes as Array)
```

Visual Basic .NET

```
Programs.MDirectory(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
               bstrModuleName As String, ByRef pCount As Integer, ByRef aNames
               As System.Array, ByRef aAttributes As System.Array)
```

C# .NET

```
Programs.MDirectory(ActiveV2Lib.Communications pCommunications, System.String
               bstrModuleName, ref System.Int32 pCount, out System.Array aNames,
               out System.Array aAttributes)
```

Description

This method lists the modules that are loaded in memory, or alternatively, the programs contained in a specified module. If bstrModuleName is empty, then all loaded modules are listed. If a module name is specified in bstrModuleName, the programs contained in the module are listed.

Usage Considerations

This method is not available for MicroV+.

Input Parameters

bstrModuleName	If "", then list all modules; else, list all programs in that module.
pCount	Number of module/program names to be returned.

Output Parameters

pCount	Actual number of module/program names returned.
aNames	ARRAY of module/program names
aAttributes	ARRAY of program/module attribute bit flags: bit 0 set => is modified bit 1 set => contains error

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
MDIRECTORY	N/A

Related Topics

Directory

Module Method

Visual Basic 6

```
Programs.Module(pCommunications as Communications, bstrModuleName as String,
                bstrProgramName  as String, pStatus as Long)
```

Visual Basic .NET

```
Programs.[Module](ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                bstrModuleName As String, ByVal bstrProgramName As String, ByRef
                pStatus As Integer)
```

C# .NET

```
Programs.Module(ActiveV2Lib.Communications pCommunications, System.String
                bstrModuleName, System.String bstrProgramName, out System.Int32
                pStatus)
```

Description

This method moves the program bstrProgramName into the module bstrModuleName.

Usage Considerations

This method is not available for MicroV+.

For additional information about program modules, refer to the V+ MODULE monitor command documentation.

Input Parameters

bstrModuleName	Module name
bstrProgramName	Program name

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
MODULE	N/A

Related Topics

ModuleFile

ModuleFile Method

Visual Basic 6

```
Programs.ModuleFile(pCommunications as Communications, bstrModuleName as String,
                BSTR bstrFileName as String, pStatus as Long)
```

Visual Basic .NET

```
Programs.ModuleFile(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                bstrModuleName As String, ByVal bstrFileName As String, ByRef
                pStatus As Integer)
```

C# .NET

```
Programs.ModuleFile(ActiveV2Lib.Communications pCommunications, System.String
                bstrModuleName, System.String bstrFileName, out System.Int32
                pStatus)
```

Description

This method associates a file name with a module. This is to extend the normal function of deriving a module file name from the first program in the file.

Usage Considerations

This method is not available for MicroV+.

Input Parameters

bstrModuleName	Name of module
bstrFileName	Name of file

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

NFSMounts Method

Visual Basic 6

```
Files.NFSMounts(pCommunications as Communications, pCount as Long,
                aMountName as ArrayString, aNode as ArrayString,
                aPathName as ArrayString)
```

Visual Basic .NET

```
Files.NFSMounts(ByVal pCommunications As ActiveV2Lib.Communications, ByRef
                pCount As Integer, ByRef aMountName As System.Array, ByRef
                aNode As System.Array, ByRef aPathName As System.Array)
```

C# .NET

```
Files.NFSMounts(ActiveV2Lib.Communications pCommunications, ref System.Int32
                pCount, out System.Array aMountName, out System.Array aNode,
                out System.Array aPathName)
```

Description

This method returns information about the NFS mounts.

Usage Considerations

This method is not available for MicroV+.

Input Parameters

pCount	Maximum number of mounts to return

Output Parameters

pCount	Actual number of mounts returned
aMountName	Array of mount names

aNode	Array of node names
aPathName	Array of path names (as exported by the nodes).

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
NET	N/A

Related Topics

FCopy

FDelete

FDirectory

FRename

Mounting an NFS Drive

NumberOfRobots

Visual Basic 6

```
Status.NumberOfRobots(pCommunications As Communications) As Long
```

Visual Basic .NET

```
Status.NumberOfRobots(ByVal pCommunications As ActiveV2Lib.Communications)
                As Integer
```

C# .NET

```
Status.get_NumberOfRobots(ActiveV2Lib.Communications pCommunications)
```

Description

This read−only property (for C# .NET, this is mapped to a function) returns the number of robots in the V+ system.

Usage Considerations

This property is not available for MicroV+.

For C# .NET, this property is mapped to a function.

Input Parameters

None.

Output Parameters

None.

Related Topics

SelectedRobot

OnAlive Event

Visual Basic 6

```
Communications.OnAlive()
```

Visual Basic .NET

```
Communications.OnAlive()
```

C# .NET

```
Communications.OnAlive()
```

Description

This event is raised periodically when activity is detected on the network while the application is connecting to a V+ controller. The frequency of the event is not specified, but while waiting for a method that could possibly execute for a long time (for example, Files.FDirectory) this event indicates that the connection is alive and active.

Input Parameters

None

Output Parameters

None

Examples

C# .NET

C# .NET

Tell the program you want to catch the event **_IAVAsynchEvents_OnAliveEventHandler** from **mCommunications** by calling the handler **Communications_OnAlive**, as follows:

```
mCommunications.OnAlive += new ActiveV2Lib._IAVAsynchEvents_OnAliveEventHandler(mCommunication_OnAlive);
```

and in your code, define the function:

```
private void mCommunications_OnAlive()

{

}
```

Related Topics

Close

ControllerIPAddress

IsOnline

OnAsynchError

OnProgramOutput

OnProgramStatusChange

OnReadPosted

Open

RequestEvents

OnASynchError Event

Visual Basic 6

```
Communications.OnAsynchError(nBoard as Long, nTask as Long, nErrorCode as Long)
```

Visual Basic .NET

```
Communications.OnAsynchError(ByVal nBoard As Integer, ByVal nTask As Integer, ByVal
                nErrorCode As Integer)
```

C# .NET

```
Communications.OnAsynchError(System.Int32 nBoard, System.Int32 nTask,
                System.Int32 nErrorCode)
```

Description

Event raised when an asynchronous error (e.g. robot error) occurs.

Usage Considerations

This event occurs when a V+ asynchronous error occurs. V+ asynchronous errors are usually robot errors, for instance when an envelope error occurs this event is raised.

Output Parameters

nBoard — Returns the board number (processor number), which is usually 1 to indicate the main CPU, for the device where the error occurred.

nTask — Value returned to identify the source of the error:

Value	Meaning
−1	Error originated from a V+ system task
0 to 31	Error originated from a user program task.

nErrorCode — Returns a standard V+ error code.

Examples

C# .NET

C# .NET

Tell the program you want to catch the event **_IAVAsynchEvents_OnAsynchErrorEventHandler** from **mCommunications** by calling the handler **Communications_OnAsynchError**, as follows:

```
mCommunications.OnReadPosted += new ActiveV2Lib._IAVAsynchEvents_OnAsynchErrorEventHandler(Communications_OnAsynchError);
```

and in your code, define the function:

```
private void mCommunications_OnAsynchError(int nBoard, int nTask, int nErrorCode)

{

}
```

Related Topics

OnAlive

OnProgramOutput

OnProgramStatusChange

OnReadPosted

RequestEvents

V+ Error Messages

OnProgramOutput Event

Visual Basic 6

```
Communications.OnProgramOutput(nBoard as Long, nTask as Long)
```

Visual Basic .NET

```
Communications.OnProgramOutput(ByVal nBoard As Integer, ByVal nTask As Integer)
```

C# .NET

```
Communications.OnProgramOutput (System.Int32 nBoard, System.Int32 nTask)
```

Description

This event is raised when V+ executes a TYPE or WRITE instruction on a terminal or keyboard LUN.

Usage Considerations

The following conditions are necessary:

a) You need to be connected to a controller.

b) You need to request the event from the controller with:

```
ICommunications::RequestEvents (long lRequest, long *pStatus)
```

c) You need to implement the handler for this event and add it to the delegate, as shown in the example code below.

Details

This event signals that the application can now retrieve the V+ program output via the GetStringEx method.

Output Parameters

nBoard	Returns the board number (processor number), which is usually 1 to indicate the main CPU, where the TYPE or WRITE instruction was executed.
nTask	Task number of the V+ program executing the TYPE or WRITE instruction.

Examples

C# .NET

C# .NET

Tell the program you want to catch the event **_IAVAsynchEvents_OnProgramOutputEventHandler** from **mCommunications** by calling the handler **Communications_OnProgramOutput**, as follows:

```
mCommunications.OnProgramOutput +=new ActiveV2Lib._IAVAsynchEvents_OnProgramOutputEventHandler(mCommunications_OnProgramOutput);
```

and in your code, define the function:

```
private void mCommunications_OnProgramOutput(int nBoard, int nTask)
{
     string sMsg; int nCount; int nContinue;
     do
     {
          mCommunications.GetStringEx(nBoard, nTask, "", out sMsg, out nCount, out nContinue);
          sMsg = OutputTextBox.Text + sMsg;
          if (sMsg.Length > 500)
          {
               sMsg = sMsg.Substring(sMsg.IndexOf("\n") + 2);
          }
          OutputTextBox.Text = sMsg;
      }
      while (nContinue != 0);
 }
```

Related Topics

GetStringEx

OnAlive

OnAsynchError

OnProgramStatusChange

OnReadPosted

RequestEvents

TYPE program instruction

WRITE program instruction

OnProgramStatusChange Event

Visual Basic 6

```
Communications.OnProgramStatusChange(nBoard as Long, nTask as Long, nStatus
                as Long)
```

Visual Basic .NET

```
Communications.OnProgramStatusChange(ByVal nBoard As Integer, ByVal nTask
                As Integer, ByVal nStatus As Integer)
```

C# .NET

```
Communications.OnProgramStatusChange(System.Int32 nBoard, System.Int32 nTask,
                System.Int32 nStatus)
```

Description

This event is raised when a V+ program completes or otherwise changes status

Output Parameters

nBoard	Returns the board number (processor number), which is usually 1 to indicate the main CPU, for the device where the program status has changed.
nTask	The V+ task number for the user program is running. Value Range: 0 to 27, if the V+ Extensions license is installed; otherwise, 0 to 6.
nStatus	The resulting V+ program status.

Value	Meaning
3	Program completed
9	Program paused
< 0	Negative numbers indicate a V+ error code.

Examples

C# .NET

C# .NET

Tell the program you want to catch the event **_IAVAsynchEvents_OnProgramStatusChangeEventHandler** from **mCommunications** by calling the handler **Communications_OnProgramStatusChange**, as follows:

```
mCommunications.OnProgramStatusChange +=new ActiveV2Lib._IAVAsynchEvents_OnProgramStatusChangeEventHandler(mCommunications_OnProgramStatusChange);
```

and in your code, define the function:

```
        private void  mCommunications_OnProgramStatusChange(int nBoard, int nTask, int nStatus)

        {

        }
```

Related Topics

OnAlive

OnAsynchError

OnProgramOutput

OnReadPosted

RequestEvents

V+ Error Messages

OnReadPosted Event

Visual Basic 6

```
Communications.OnReadPosted(lBoardNumber as Long, lTaskNumber as Long)
```

Visual Basic .NET

```
Communications.OnReadPosted(ByVal lBoardNumber As Integer, ByVal lTaskNumber
                As Integer)
```

C# .NET

```
Communications.OnReadPosted(System.Int32 lBoardNumber, System.Int32
                lTaskNumber)
```

Description

This event is raised when V+ executes a READ instruction that expects input from the keyboard.

Details

This event signals that ActiveV can now send input to the READ instruction via the SendStringEx method.

Output Parameters

lBoardNumber	Returns the board number (processor number), which is usually 1 to indicate the main CPU, where the READ instruction was executed.
lTaskNumber	Task number of the V+ program that executing the READ instruction.

Examples

C# .NET

C# .NET

Tell the program you want to catch the event **_IAVAsynchEvents_OnReadPostedEventHandler** from **mCommunications** by calling the handler **Communications_OnReadPosted**, as follows:

```
mCommunications.OnReadPosted += new ActiveV2Lib._IAVAsynchEvents_OnReadPostedEventHandler(mCommunications_OnReadPosted);
```

and in your code, define the function:

```
private void mCommunications_OnReadPosted(int nBoard, int nTask)
{
    int nStatus; int nCount; Array Names; Array Values; Array Contents;
    // Read the real and string arrays ckd.msg[] and $ckd.msg[], which
    // contain message information
    nCount = 2;
    mStatus.Listr(mCommunications, -1, "", "ckd.msg[]", ref nCount, out Names, out Values, out nStatus);
    mStatus.Lists(mCommunications, -1, "", "$ckd.msg[]", ref nCount, out Names, out Contents, out nStatus);
}
```

Related Topics

OnAlive

OnAsynchError

OnProgramOutput

OnProgramStatusChange

OnReadPosted

RequestEvents

SendStringEx

READ program instruction

Open Method

Visual Basic 6

```
Communications.Open(bstrDevice as String,  uUnit as Long, bstrName as String,
                uMode as Long, bstrMode as String)
```

Visual Basic .NET

```
Communications.Open(ByVal bstrDevice As String, ByVal uUnit As Integer, ByVal
                bstrName As String, ByVal uMode As Integer, ByVal bstrMode
                As String)
```

C# .NET

```
Communications.Open(System.String bstrDevice, System.Int32 uUnit, System.String
                bstrName, System.Int32 uMode, System.String bstrMode)
```

Description

This method opens a communications channel to a V+ system. It uses the following types of connections:

1. UDP –– Used to connect to V+ systems.
2. SERIAL –– Used to connect to MicroV+ systems.

Input Parameters

bstrDevice	Name of the physical device to use for connection. The recognized names are: 1. UDP 2. COM1 – COM16
uUnit	Unit number of device. (NOTE: Not used on V+ systems at this time.)
bstrName	Name of V+ system to use for connection. This can be an IP address, for instance. See Example for formatting details.
uMode	Various mode bits, if necessary. For serial communications, this is flow control:

	0=none 1=Xon/Xoff 2=RTS/CTS 3=Both
bstrMode	Various mode names, if necessary. For serial communications, this is the serial line settings in format "BBBB,P,D,S" (baud rate, parity, data bits, stop bits) if empty, the default is "9600,N,8,1"

Output Parameters

None.

Examples

Visual Basic 6 Visual Basic .NET C# .NET

Visual Basic 6

```
Set frmOperatorPanel.comm = New ActiveV2Lib.Communications
Set comm = frmOperatorPanel.comm


' open connection to the controller
On Error GoTo ErrorHandler
Call comm.Open("UDP", 0, strHost, 0, "")
frmOperatorPanel.ipAddress = strHost
```

Visual Basic .NET

```
' Establish the connection with the controller.
Private Function MakeConnection()
    Dim status As Long
    Dim errors As New ActiveV2Lib.ErrorHandler

    ' Create the comm object.
    mCommunications = New ActiveV2Lib.Communications

    Try
```

```
            ' Change the mouse cursor to an hourglass because it may take some time to connect.
            Cursor = Cursors.WaitCursor

mCommunications.Open("UDP", 0, AddressListBox.Text, 0, "")

            mCommunications.RequestEvents(1, status)

            If status = 0 Then
                If MessageBox.Show(Me, "This V+ system is already sending events to another Application." + _
                                   vbCrLf + "Do you want to force events to come to this application?", _
                                   "Connect", MessageBoxButtons.YesNo, MessageBoxIcon.Question) = DialogResult.Yes Then
                    mCommunications.RequestEvents(0, status)
                Else
                    mCommunications.Close()
                    Exit Function
                End If
            End If

            DialogResult = DialogResult.OK
            Close()

        Catch Exception As System.Runtime.InteropServices.COMException
            Dim message As String
            errors.GetErrorString(Exception.ErrorCode, message)
            MessageBox.Show(Me, message, "Connection Error", _
                            MessageBoxButtons.OK, MessageBoxIcon.Error)

        Catch Exception As Exception
            MessageBox.Show(Me, Exception.Message, "Connection Error", _
                            MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try

        ' Restore the normal mouse cursor.
        Cursor = Cursors.Default
    End Function
```

C# .NET

```
        /// <summary>
        /// Establish the connection with the controller.
        /// </summary>
        void MakeConnection()
```

```
{
     try
     {
          int status;

mCommunications.Open( "UDP", 0, AddressListBox.Text, 0, "" );

          mCommunications.RequestEvents( 1, out status);

          if (status == 0)
          {
               if (MessageBox.Show(this, "This V+ system is already sending events to another Application." +
                    "\nDo you want to force events to come to this application?",
                    "Connect", MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
               {
                    mCommunications.RequestEvents(0, out status);
               }
               else
               {
                    mCommunications.Close();
                    return;
               }
          }
          DialogResult = DialogResult.OK;
          Close();
     }
     catch (System.Runtime.InteropServices.COMException Ex)
     {
          ActiveV2Lib.ErrorHandler HError = new ActiveV2Lib.ErrorHandler();
          string sMsg;


          HError.GetErrorString(Ex.ErrorCode, out sMsg);
          MessageBox.Show(this, sMsg, "Connection Error");
          mCommunications = null;
     }
     catch (Exception Ex)
     {
          MessageBox.Show( this, Ex.Message, "Connection Error" );
          mCommunications = null;
     }
}
```

Related Topics

Close

ControllerIPAddress

IsOnline

OnAlive

Parameter Method

Visual Basic 6

```
MiscControl.Parameter(pCommunications as Communications, bstrParameterName
               as String, fValue as Single, pStatus as Long)
```

Visual Basic .NET

```
MiscControl.Parameter(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
               bstrParameterName As String, ByVal fValue As Single, ByRef
               pStatus As Integer)
```

C# .NET

```
MiscControl.Parameter(ActiveV2Lib.Communications pCommunications, System.String
               bstrParameterName, System.Single fValue, out System.Int32 pStatus)
```

Description

This method sets a V+ system parameter or V+ parameter arrays (i.e., PARAMETER V.OFFSET[4]). See Examples section below.

Usage Considerations

This method is not available for MicroV+.

Input Parameters

bstrParameterName	Name of parameter to set.
fValue	New value for the parameter.

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Examples

C# .NET – Setting V+ Parameter Example

C# .NET – Setting V+ Parameter–Array Example

C# .NET – Setting V+ Parameter Example

```
Main Method:
------------
....
        // set PARAMETER AVR.LOGSIZE
        parValue = 42;
        cx.parAvrLogsize = parValue;
        Console.WriteLine("Setting Parameter {0} to {1} returned {2}",
                                "AVR.LOGSIZE", parValue.ToString(), cx.SUCCESS);

....

Business Class Method:
----------------------
....
//////// V+ PARAMETERS Get and Set Accessors ////////////////////////////////////////////
        /// <summary>
        /// Property: PARAMETER AVR.LOGSIZE
        /// </summary>
        public float parAvrLogsize
        {
                set     {
                        try {
misc.Parameter(comm, "AVR.LOGSIZE", value, out Success); }
                        catch (System.Runtime.InteropServices.COMException e) {
                                Console.WriteLine("SetParameter - COM Exception: "+e.ToString());}
                }
                get     {
                        this.GetParameters(out parCount, out parNames, out parNumbers, out parValues);
                        for (i = 1; i<= parCount; i++) {
                                if(parNames.GetValue(i).ToString()== "AVR.LOGSIZE") {
                                        parValue = Convert.ToInt32(parValues.GetValue(i));
                                        break;
                                }
```

```
                        }
                        return parValue;
                }
        }
....
```

C# .NET – Setting V+ Parameter–Array Example:

```
....
Main Method:
------------

        // SET PARAMETER V.OFFSET[1] .. V.OFFSET[9] to 220
        for ( i = 1; i <= 9; i++ ) {
                parValue = 220;
                parName = "V.OFFSET[" + i.ToString() +"]";
                cx.SetVisionParameter(parName, parValue);
                Console.WriteLine("Setting Parameter {0} to {1} ", parName, parValue.ToString());
        }
....

Business Class Method:
----------------------

        /// <summary>
        /// Set a V+ VISION PARAMETER Value
        /// (i.e. PARAMETER V.GAIN[4], or PARAMETER DISPLAY.CAMERA = 8)
        /// </summary>
        /// <remarks>All non-array Parameters are available as Properties
        ///  with Get and Set Acessors.</remarks>
        /// <param name="parName"></param>
        /// <param name="parValue"></param>
        public void SetVisionParameter(string parName, float parValue)
        {
                try {
                        if ((parName.StartsWith("V.") == true) | (parName == "DISPLAY.CAMERA")){
 misc.Parameter(comm, parName, parValue, out Success);}
                        else { Success = 0; }
                }
                catch (System.Runtime.InteropServices.COMException e) {
                        Console.WriteLine("SetVisionParameter - COM Exception: "+e.ToString());
                }
        }
```

. . . . .

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
PARAMETER	PARAMETER

Parameter (Status Class)

Visual Basic 6

```
Status.Parameter(pCommunications as Communications, bstrParmName as String,
                pCount as Long, aParmName as ArrayString, aParmNumber as ArrayLong,
                aParmValue as ArraySingle)
```

Visual Basic .NET

```
Status.Parameter(ByVal pCommunications As ActiveV2Lib.Communications, ByVal bstrParmName
                As String, ByRef pCount As Integer, ByRef aParmName As System.Array, ByRef
                aParmNumber As System.Array, ByRef aParmValue As System.Array)
```

C# .NET

```
Status.Parameter(ActiveV2Lib.Communications pCommunications, System.String bstrParmName,
                ref System.Int32 pCount, out System.Array aParmName, out System.Array
                aParmNumber, out System.Array aParmValue)
```

Description

This method returns the values of system parameters. All parameters are treated as array parameters.

Usage Considerations

This method is not available for MicroV+.

Input Parameters

bstrParmName	Name of system parameter. This will normally be an empty string, to mean return all system parameters. It can also contain "V" to return all vision parameters, or "[2]" to return all non−array parameters plus all the 2nd element of array parameters.
pCount	Maximum number of values to return. Note that array parameters will occupy as many array locations as there are parameters by that name.

Output Parameters

pCount	number of array elements actually returned.
aParmName	array of parameter names. Note that the names of array parameters will occur in multiple aParmName indexes.
aParmNumber	parameter number
aParmValue	parameter value

Example

As an example, let's say that the system parameters consist of:

X 10.5
Y 11.5
Z 1: 12.5 2: 13.5 3: 14.5 4: 15.5

Then the output arrays contain the following:

aParmName(1) = X, aParmNumber(1) = 1, aParmValue(1) = 10.5

aParmName(2) = Y, aParmNumber(2) = 1, aParmValue(2) = 11.5

aParmName(3) = Z, aParmNumber(3) = 1, aParmValue(3) = 12.5

aParmName(4) = Z, aParmNumber(4) = 2, aParmValue(4) = 13.5

aParmName(5) = Z, aParmNumber(5) = 3, aParmValue(5) = 14.5

aParmName(6) = Z, aParmNumber(6) = 4, aParmValue(6) = 15.5

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
PARAMETER	PARAMETER

Related Topics

Switch

PCopy Method

Visual Basic 6

```
Programs.PCopy(pCommunications as Communications, bstrSourceName as String,
               bstrDestName as String, pStatus as Long)
```

Visual Basic .NET

```
Programs.PCopy(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
               bstrSourceName As String, ByVal bstrDestName As String, ByRef
               pStatus As Integer)
```

C# .NET

```
Programs.PCopy(ActiveV2Lib.Communications pCommunications, System.String
                bstrSourceName, System.String bstrDestName, ref System.Int32
                pStatus)
```

Description

This method copies one V+ program to another.

Usage Considerations

This method is not available for MicroV+.

Input Parameters

bstrSourceName	old name
bstrDestName	new name

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
COPY	N/A

Related Topics

Delete

Deletep

Rename

Store

Storem

Storep

Prime Method

Visual Basic 6

```
Programs.Prime(pCommunications as Communications, lTaskNumber as Long,
               bstrProgramName as String, pStatus as Long)
```

Visual Basic .NET

```
Programs.Prime(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
               lTaskNumber As Integer, ByVal bstrProgramName As String, ByRef
               pStatus As Integer)
```

C# .NET

```
Programs.Prime(ActiveV2Lib.Communications pCommunications, System.Int32 lTaskNumber,
               System.String bstrProgramName, out System.Int32 pStatus)
```

Description

This method executes the V+ PRIME command to prepare a program for execution (resets the V+ interpreter pointer to the first line of code in the named program) without actually starting execution.

Input Parameters

lTaskNumber	task number
bstrProgramName	program name

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Example

Visual Basic 6

```
' obtain the query result
Set recPrograms = qry.OpenRecordset

' loop through the query result, abort and kill all tasks and prime each
' task with the appropriate program
' then start any autostart program
On Error Resume Next
With recPrograms
    Do While Not .EOF

        ' Abort task
        Call prog.Abort(comm, !task, stat)

        ' Prime task with program name
        Call prog.Prime(comm, !task, !Program, stat)

        ' if autostart task then start it
        If !autostart Then
            Call prog.Proceed(comm, !task, stat)
            If stat <> VE_SUCCESS Or Err.Number <> 0 Then
                If stat <> VE_SUCCESS Then
                    Call errh.GetErrorString(stat, str)
                Else
                    Call errh.GetErrorString(Err.Number, str)
                End If
                Call MsgBox(str, vbOKOnly, "Error attempting to Proceed Program " _
                & !Program & "() in task " & !task)
            End If
        End If
        .MoveNext
    Loop
End With
```

Corresponding V+	Corresponding MicroV+

Keyword	Keyword
PRIME	PRIME

Related Topics

Abort

Execute

Kill

Proceed

ZERO

Proceed Method

Visual Basic 6

```
Programs.Proceed(pCommunications as Communications, lTaskNumber as Long,
                pStatus as Long)
```

Visual Basic .NET

```
Programs.Proceed(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                lTaskNumber As Integer, ByRef pStatus As Integer)
```

C# .NET

```
Programs.Proceed(ActiveV2Lib.Communications pCommunications, System.Int32
                lTaskNumber, out System.Int32 pStatus)
```

Description

This method executes the V+ PROCEED monitor command to resume execution of an application program.

Input Parameters

lTaskNumber	Task number

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Example

Visual Basic 6

```
' obtain the query result
Set recPrograms = qry.OpenRecordset

' loop through the query result, abort and kill all tasks and prime each
```

```
' task with the appropriate program
' then start any autostart program
On Error Resume Next
With recPrograms
    Do While Not .EOF

        ' Abort task
        Call prog.Abort(comm, !task, stat)

        ' Prime task with program name
        Call prog.Prime(comm, !task, !Program, stat)

        ' if autostart task then start it
        If !autostart Then
            Call prog.Proceed(comm, !task, stat)
            If stat <> VE_SUCCESS Or Err.Number <> 0 Then
                If stat <> VE_SUCCESS Then
                    Call errh.GetErrorString(stat, str)
                Else
                    Call errh.GetErrorString(Err.Number, str)
                End If
                Call MsgBox(str, vbOKOnly, "Error attempting to Proceed Program " _
                & !Program & "() in task " & !task)
            End If
        End If
        .MoveNext
    Loop
End With
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
PROCEED	PROCEED

Related Topics

Abort

Execute

Kill

Retry

Prime

ZERO

QueryReadRequest Method

Visual Basic 6

```
Communications.QueryReadRequest(lBoard as Long, lTaskID as Long,
               pStatus as Long)
```

Visual Basic .NET

```
Communications.QueryReadRequest(ByVal lBoard As Integer, ByVal lTaskID As
               Integer, ByRef pStatus As Integer)
```

C# .NET

```
Communications.QueryReadRequest(System.Int32 lBoard, System.Int32 lTaskID,
               out System.Int32 pStatus)
```

Description

This method queries V+ for a pending read request on the specified board and task.

Input Parameters

lBoard
board number (processor number), which is usually 1 to indicate the main CPU
lTaskID
task ID

Output Parameters

pStatus
0 if no read request
non−zero if read request

Details

The QueryReadRequest method queries V+ for a pending read request on the specified board and task. For example, let's say a V+ program executed a READ instruction

to read from the "keyboard" (meaning ActiveV). The PC side should invoke the SendStringEx method to send data to that V+ READ instruction. However, before sending the data, the PC side should check QueryReadRequest, which tells whether the V+ program is really performing a READ. Otherwise, the SendStringEx data might just be discarded (lost) on the V+ side.

Related Topics

GetStringEx

OnReadPosted Event

SendStringEx

Rename Method

Visual Basic 6

```
Programs.Rename(pCommunications as Communications, bstrSourceName as String,
                bstrDestName  as String, pStatus  as Long)
```

Visual Basic .NET

```
Programs.Rename(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                bstrSourceName As String, ByVal bstrDestName As String, ByRef
                pStatus As Integer)
```

C# .NET

```
Programs.Rename(ActiveV2Lib.Communications pCommunications, System.String
                bstrSourceName, System.String bstrDestName, ref System.Int32
                pStatus)
```

Description

Renames a V+ program that is currently in memory.

Usage Considerations

This method is not available for MicroV+.

Input Parameters

bstrSourceName
Old name
bstrDestName
New name

Output Parameters

pStatus
Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
RENAME	RENAME

Related Topics

Delete

Deletep

PCopy

Store

Storem

Storep

RequestEvents Method

Visual Basic 6

```
Communications.RequestEvents(lRequest as Long, pStatus as Long)
```

Visual Basic .NET

```
Communications.RequestEvents(ByVal lRequest As Integer, ByRef pStatus As Integer)
```

C# .NET

```
Communications.RequestEvents(System.Int32 lRequest, out System.Int32 pStatus)
```

Description

This method requests that V+ sends asynchronous output and program output to the ActiveV application. Asynchronous outputs are usually robot errors, for instance, Timeout enabling amplifier, etc. Program output is usually generated by the TYPE instruction.

Input Parameters

lRequest
Indicates whether V+ is to send output to ActiveV.

Value
Meaning
2
V+ always sends asynchronous output to ActiveV.
1
V+ is to send asynchronous output to ActiveV if no other application has requested the output.
0 V+ is to stop sending asynchronous output to the ActiveV application.

Output Parameters

pStatus
Indicates whether V+ is sending output to ActiveV+.

Value
Meaning
1
V+ is sending asynchronous output to the ActiveV application.
0
V+ task is already sending the output to another application.

Examples

Visual Basic 6 Visual Basic .NET C# .NET

Visual Basic 6

```
Private Sub connectV(ByVal strHost As String, pStatus As Long)
    Dim str As String
    Dim comm As ActiveV2Lib.Communications
    Dim stat As Long

    Set frmOperatorPanel.comm = New ActiveV2Lib.Communications
    Set comm = frmOperatorPanel.comm

    ' open connection to the controller
    On Error GoTo ErrorHandler
    Call comm.Open("UDP", 0, strHost, 0, "")
    frmOperatorPanel.ipAddress = strHost

    ' Request V+ to send asynchronous events to this application if
    ' no other task is already receiving them
    Call comm.RequestEvents(1, stat)

    ' If stat returns 0 then V+ is already sending events to another
    ' ActiveV+ application.  Prompt the user to see if they want events
    ' to be serviced by this application
    If stat = 0 Then 'events already requested by someone else
        str = "This V+ system is already sending events to another Application."
        str = str + vbCrLf
        str = str + "Do you want to force events to come to this application?"

        If MsgBox(str, vbOKCancel) = vbOK Then
        ' This call to RequestEvents (with mode=2) will force
```

```
            ' events to come to this application.
                Call comm.RequestEvents(2, stat)
                pStatus = vbOK   ' set our global flag
            Else
                pStatus = vbAbort
            End If
        Else
            pStatus = vbOK ' set our global flag
        End If

    Exit Sub
```

Visual Basic .NET

```
' Establish the connection with the controller.
Private Function MakeConnection()
    Dim status As Long
    Dim errors As New ActiveV2Lib.ErrorHandler

    ' Create the comm object.
    mCommunications = New ActiveV2Lib.Communications

    Try
        ' Change the mouse cursor to an hourglass because it may take some time to connect.
        Cursor = Cursors.WaitCursor

        mCommunications.Open("UDP", 0, AddressListBox.Text, 0, "")

mCommunications.RequestEvents(1, status)

        If status = 0 Then
            If MessageBox.Show(Me, "This V+ system is already sending events to another Application." + _
                               vbCrLf + "Do you want to force events to come to this application?", _
                               "Connect", MessageBoxButtons.YesNo, MessageBoxIcon.Question) = DialogResult.Yes Then
                mCommunications.RequestEvents(0, status)
            Else
                mCommunications.Close()
                Exit Function
            End If
        End If

        DialogResult = DialogResult.OK
```

```
        Close()

    Catch Exception As System.Runtime.InteropServices.COMException
        Dim message As String
        errors.GetErrorString(Exception.ErrorCode, message)
        MessageBox.Show(Me, message, "Connection Error", _
                        MessageBoxButtons.OK, MessageBoxIcon.Error)

    Catch Exception As Exception
        MessageBox.Show(Me, Exception.Message, "Connection Error", _
                        MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Try

    ' Restore the normal mouse cursor.
    Cursor = Cursors.Default
End Function
```

C# .NET

```
/// <summary>
/// Establish the connection with the controller.
/// </summary>
void MakeConnection()
{
     try
     {
          int status;

          mCommunications.Open( "UDP", 0, AddressListBox.Text, 0, "" );

          mCommunications.RequestEvents( 1, out status);

          if (status == 0)
          {
               if (MessageBox.Show(this, "This V+ system is already sending events to another Application." +
                    "\nDo you want to force events to come to this application?",
                    "Connect", MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
               {
                    mCommunications.RequestEvents(0, out status);
               }
               else
               {
```

```
                    mCommunications.Close();
                    return;
                }
            }
            DialogResult = DialogResult.OK;
            Close();
        }
        catch (System.Runtime.InteropServices.COMException Ex)
        {
            ActiveV2Lib.ErrorHandler HError = new ActiveV2Lib.ErrorHandler();
            string sMsg;


            HError.GetErrorString(Ex.ErrorCode, out sMsg);
            MessageBox.Show(this, sMsg, "Connection Error");
            mCommunications = null;
        }
        catch (Exception Ex)
        {
            MessageBox.Show( this, Ex.Message, "Connection Error" );
            mCommunications = null;
        }
    }
```

Related Topics

OnAsynchError

Retry Method

Visual Basic 6

```
Programs.Retry(pCommunications as Communications, lTaskNumber as Long,
               pStatus as Long)
```

Visual Basic .NET

```
Programs.Retry(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
               lTaskNumber As Integer, ByRef pStatus As Integer)
```

C# .NET

```
Programs.Retry(ActiveV2Lib.Communications pCommunications, System.Int32
               lTaskNumber, out System.Int32 pStatus)
```

Description

This method executes a V+ RETRY monitor command to repeat execution of the last interrupted program instruction and continue execution of the program.

Input Parameters

lTaskNumber
Task number

Output Parameters

pStatusReturns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
RETRY	RETRY

Related Topics

Abort

Execute

Kill

Prime

ZERO

Rvariables Method

Visual Basic 6

```
Status.Rvariables(pCommunications as Communications, lTask as Long,
                bstrProgram as String, bstrName as String, pCount as Long,
                aNames as ArrayString)
```

Visual Basic .NET

```
Status.Rvariables(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                lTask As Integer, ByVal bstrProgram As String, ByVal bstrName
                As String, ByRef pCount As Integer, ByRef aNames As System.Array)
```

C# .NET

```
Status.Rvariables(ActiveV2Lib.Communications pCommunications, System.Int32
                lTask, System.String bstrProgram, System.String bstrName,
                ref System.Int32 pCount, out System.Array aNames)
```

Description

This method returns the names of real variables. Only the variable names are returned.

Usage Considerations

This method is not available for MicroV+.

Input Parameters

bstrName
Name (may contain wildcards), or empty string to get the names of all real variables.
pCount
Number of names to return.
lTask
Task number for context. A value of −1 indicates that no task is specified.
bstrProgram
Program name for context (can be an empty string).

Output Parameters

pCount
Actual number of names returned.
aNames
Array of names.

Examples

C# .NET

C# .NET

```
int   nCount = 10;      // Only request the first 10 real variables names.
Array Names;            // Storages for the variables names.
try
{
     // Get the list of location variables.
     mStatus.Rvariables(mCommunications, -1, "", "", ref nCount, out Names);
}
catch (System.Runtime.InteropServices.COMException Ex)
{
     ShowError(Ex);
     return;
}
catch (Exception Ex)
{
     ShowError(Ex);
     return;
}
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
LISTR	LISTR

Related Topics

Listr

Listl

Lists

Lvariables

Rvariables

Svariables

SelectedRobot

Visual Basic 6

```
Status.SelectedRobot(pCommunications As Communications) As Long
```

Visual Basic .NET

```
Status.SelectedRobot(ByVal pCommunications As ActiveV2Lib.Communications) As Integer
```

C# .NET

```
Status.set_SelectedRobot(ActiveV2Lib.Communications pCommunications, System.Int32 pVal)

System.Int32 Status.get_SelectedRobot(ActiveV2Lib.Communications pCommunications)
```

Description

This read/write property (for C# .NET, this is mapped to a function, see below for details) reflects the currently selected robot.

Usage Considerations

This property is implemented differently in C# .NET versus Visual Basic (or Visual Basic .NET), as follows:

- For C# .NET applications, the function is implemented twice: once for writing the value, and once for reading the value. Therefore, in a C# .NET application, you would see two different functions, as shown in the C#.NET syntax above.
- For Visual Basic applications, that detail is "hidden" from the user and only one function is used. For example, in Visual Basic, your program may contain the following lines:

```
dim X as long

X = SelectedRobot 'Save current robot

SelectedRobot = 2 'Now select robot 2
```

Input Parameters

None.

Output Parameters

None.

Related Topics

NumberOfRobots

SendStringEx Method

Visual Basic 6

```
Communications.SendStringEx(nBoard as Long, nTask as Long, bstrString as String)
```

Visual Basic .NET

```
Communications.SendStringEx(ByVal nBoard As Integer, ByVal nTask As Integer, ByVal
               bstrString As String)
```

C# .NET

```
Communications.SendStringEx(System.Int32 nBoard, System.Int32 nTask, System.String
               bstrString)
```

Description

This method sends a string to V+ for input to a program running on the specified board/task. The V+ program will be executing a READ instruction. This method will usually be called from the OnReadPosted event handler, and/or when QueryReadRequest indicates that the specified board/task expects input.

Input Parameters

nBoard	board number (processor number), which is usually 1 to indicate the main CPU, to which to send (currently ignored)
nTask	task to which to send (currently ignored)
bstrString	string to send

Output Parameters

None.

Examples

Visual Basic 6 Visual Basic .NET C# .NET

Visual Basic 6

```
Private Sub comm_OnReadPosted(ByVal nBoard As Long, ByVal nTask As Long)
    Dim status As ActiveV2Lib.status
    Dim strMsgRName() As String
    Dim sngMsgRValue() As Single
    Dim strMsgSName() As String
    Dim strMsgSContent() As String
    Dim msgBoxResult As VbMsgBoxResult
    Dim strPrompt As String
    Dim msgBoxStyle As VbMsgBoxStyle
    Dim lngpCount As Long
    Dim str As String
    Dim stat As Long

    Set status = New ActiveV2Lib.status

    ' read the real and string arrays ckd.msg[] and $ckd.msg[]
    '  which contain message information
    lngpCount = 2
    Call status.Listr(comm, -1, "", "ckd.msg[]", lngpCount, strMsgRName(), sngMsgRValue(), stat)
    Call status.Lists(comm, -1, "", "$ckd.msg[]", lngpCount, strMsgSName(), strMsgSContent(), stat)

    If lngpCount > 0 Then
        strPrompt = strMsgSContent(1) & vbCrLf & strMsgSContent(2)
        msgBoxStyle = sngMsgRValue(2)

        ' pop-up message box
        msgBoxResult = MsgBox(strPrompt, msgBoxStyle, "Message from Controller")

        ' send response back to controller
        ' pack response value into 2-byte string
        Call comm.SendStringEx(nBoard, nTask, CStr(msgBoxResult) & vbCrLf)
    End If

End Sub
```

Visual Basic .NET

```
' Event-handler called by ActiveV when a V+ program requests some
' input from the user. Display the relevant information in
' a message box and return the response to V+.
Private Sub CommReadPosted(ByVal nBoard As Integer, ByVal nTask As Integer) Handles mCommunications.OnReadPosted

    Dim nStatus As Integer
    Dim nCount As Integer
    Dim Names As System.Array
    Dim Values As System.Array
    Dim Contents As System.Array

    Dim sPrompt As String
    Dim nStyleMask As Integer
    Dim nMsgBoxResult As Integer

    Dim eButton As MessageBoxButtons
    Dim eIcon As MessageBoxIcon
    Dim eDefButton As MessageBoxDefaultButton
    Dim eDialogResult As DialogResult

    ' Read the real and string arrays ckd.msg[] and $ckd.msg[], which
    ' contain message information.
    nCount = 2
    mStatus.Listr(mCommunications, -1, "", "ckd.msg[]", nCount, Names, Values, nStatus)
    mStatus.Lists(mCommunications, -1, "", "$ckd.msg[]", nCount, Names, Contents, nStatus)

    If nCount > 0 Then
        sPrompt = Contents(1) & vbCrLf & Contents(2)

        nStyleMask = Values(2)

        ' Select the default button.
        Select Case (nStyleMask And 768)
            Case 256 ' DefaultButton2
                eDefButton = MessageBoxDefaultButton.Button2
            Case 512 ' DefaultButton3
                eDefButton = MessageBoxDefaultButton.Button3
            Case Else ' DefaultButton1
                eDefButton = MessageBoxDefaultButton.Button1
        End Select
```

```
' Convert the icon.
Select Case (nStyleMask And 160)
    Case 16 ' Critical
        eIcon = MessageBoxIcon.Error
    Case 32 ' Question
        eIcon = MessageBoxIcon.Question
    Case 48 ' Exclamation
        eIcon = MessageBoxIcon.Exclamation
    Case 64 ' Information
        eIcon = MessageBoxIcon.Information
    Case Else ' None
        eIcon = MessageBoxIcon.None
End Select

' Convert the buttons.
Select Case (nStyleMask And 8)
    Case 1 ' OKCancel
        eButton = MessageBoxButtons.OKCancel
    Case 2 ' AbortRetryIgnore
        eButton = MessageBoxButtons.AbortRetryIgnore
    Case 3 ' YesNoCancel
        eButton = MessageBoxButtons.YesNoCancel
    Case 4 ' YesNo
        eButton = MessageBoxButtons.YesNo
    Case 5 ' RetryCancel
        eButton = MessageBoxButtons.RetryCancel
    Case Else ' OKOnly
        eButton = MessageBoxButtons.OK
End Select

' Pop-up message box.
eDialogResult = MessageBox.Show(Me, sPrompt, "Message from Controller", eButton, eIcon, eDefButton)

' Convert the result enum to the proper value for the controller.
Select Case eDialogResult
    Case DialogResult.Abort
        nMsgBoxResult = 3  ' Abort
    Case DialogResult.Cancel
        nMsgBoxResult = 2  ' Cancel
    Case DialogResult.Ignore
        nMsgBoxResult = 5  ' Ignore
    Case DialogResult.No
        nMsgBoxResult = 7  ' No
    Case DialogResult.Retry
```

```
                nMsgBoxResult = 4  ' retry
            Case DialogResult.Yes
                nMsgBoxResult = 6  ' Yes
            Case DialogResult.OK
            Case Else
                nMsgBoxResult = 1  ' OK
        End Select

        ' Send response back to the controller.
        ' Pack response value into a 2-byte string.
        mCommunications.SendStringEx(nBoard, nTask, CStr(nMsgBoxResult) & vbCrLf)
    End If
End Sub
```

C# .NET

```
    /// <summary>
    /// Called by the Communication component. This handler will display a dialog box
    /// to request input from the user.
    /// </summary>
    /// <param name="nBoard">Board Number</param>
    /// <param name="nTask">Task Number</param>
    private void mCommunications_OnReadPosted(int nBoard, int nTask)
    {
         int nStatus;
         int nCount;
         Array Names;
         Array Values;
         Array Contents;

         // Read the real and string arrays ckd.msg[] and $ckd.msg[], which
         // contain message information.
         nCount = 2;
         mStatus.Listr(mCommunications, -1, "", "ckd.msg[]", ref nCount, out Names, out Values, out nStatus);
         mStatus.Lists(mCommunications, -1, "", "$ckd.msg[]", ref nCount, out Names, out Contents, out nStatus);

         if (nCount > 0)
         {
              string sPrompt;
              int nStyleMask;
              int nMsgBoxResult;
              MessageBoxButtons          eButton;
              MessageBoxIcon               eIcon;
```

```
MessageBoxDefaultButton eDefButton;
DialogResult               eDialogResult;

sPrompt = Contents.GetValue(Contents.GetLowerBound(0)) + "\n" +
     Contents.GetValue(Contents.GetLowerBound(0) + 1);


nStyleMask = (int)((float) Values.GetValue(Values.GetLowerBound(0) + 1));


// Select the default button.
switch (nStyleMask & 0x0300)
{
     case 256:          // DefaultButton2
          eDefButton = MessageBoxDefaultButton.Button2;
          break;
     case 512:          // DefaultButton3
          eDefButton = MessageBoxDefaultButton.Button3;
          break;
     case 0:
     default:           // DefaultButton1
          eDefButton = MessageBoxDefaultButton.Button1;
          break;
}

// Convert the icon.
switch (nStyleMask & 0x00A0)
{
     case 16:           // Critical
          eIcon = MessageBoxIcon.Error;
          break;
     case 32:           // Question
          eIcon = MessageBoxIcon.Question;
          break;
     case 48:           // Exclamation
          eIcon = MessageBoxIcon.Exclamation;
          break;
     case 64:           // Information
          eIcon = MessageBoxIcon.Information;
          break;
     case 0:
     default:           // None
          eIcon = MessageBoxIcon.None;
```

```
          break;
}

// Convert the buttons.
switch (nStyleMask & 0x0008)
{
     case 1:          // OKCancel
          eButton = MessageBoxButtons.OKCancel;
          break;
     case 2:          // AbortRetryIgnore
          eButton = MessageBoxButtons.AbortRetryIgnore;
          break;
     case 3:          // YesNoCancel
          eButton = MessageBoxButtons.YesNoCancel;
          break;
     case 4:          // YesNo
          eButton = MessageBoxButtons.YesNo;
          break;
     case 5:          // RetryCancel
          eButton = MessageBoxButtons.RetryCancel;
          break;
     case 0:
     default:          // OKOnly
          eButton = MessageBoxButtons.OK;
          break;
}

// Pop-up message box.
eDialogResult = MessageBox.Show(this, sPrompt, "Message from Controller", eButton, eIcon, eDefButton);

// Convert the result enum to the proper value for the controller
     switch(eDialogResult)
{
     case DialogResult.Abort:
          nMsgBoxResult = 3;          // Abort
          break;
     case DialogResult.Cancel:
          nMsgBoxResult = 2;          // Cancel
          break;
     case DialogResult.Ignore:
          nMsgBoxResult = 5;          // Ignore
          break;
     case DialogResult.No:
          nMsgBoxResult = 7;          // No
```

```
                    break;
               case DialogResult.Retry:
                    nMsgBoxResult = 4;          // retry
                    break;
               case DialogResult.Yes:
                    nMsgBoxResult = 6;          // Yes
                    break;
                  case DialogResult.OK:
               default:
                    nMsgBoxResult = 1;          // OK
                    break;
          }

          // Send response back to the controller.
          // Pack response value into a 2-byte string.
          mCommunications.SendStringEx(nBoard, nTask, nMsgBoxResult.ToString() + "\n");
     }
}
```

Related Topics

GetStringEx

OnReadPosted Event

QueryReadRequest

SetBreakpoint Method

Visual Basic 6

```
Programs.SetBreakpoint(pCommunications as Communications, bstrProgram as String,
                nLine as Long, pStatus as Long)
```

Visual Basic .NET

```
Programs.SetBreakpoint(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                strProgram As String, ByVal nLine As Integer, ByRef pStatus
                As Integer)
```

C# .NET

```
Programs.SetBreakpoint(ActiveV2Lib.Communications pCommunications, System.String
                strProgram, System.Int32 nLine, out System.Int32 pStatus)
```

Description

This method sets a breakpoint in a V+ program.

Input Parameters

bstrProgram	Name of V+ program
nLine	Line number for the breakpoint (must be positive)

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
BPT	BPT

Related Topics

ClearBreakpoint

ListBreakpoints

SetL

Visual Basic 6

```
MiscControl.SetL(pCommunications as Communications, bstrVariable as String,
                fX as Single, fY as Single, fZ as Single, fYaw as Single,
                fPitch as Single, fRoll as Single, pStatus as Long)
```

Visual Basic .NET

```
MiscControl.SetL(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                bstrVariable As String, ByVal fX As Single, ByVal fY As Single,
                ByVal fZ As Single, ByVal fYaw As Single, ByVal fPitch As Single,
                ByVal fRoll As Single, ByRef pStatus As Integer)
```

C# .NET

```
MiscControl.SetL(ActiveV2Lib.Communications pCommunications, System.String
                bstrVariable, System.Single fX, System.Single fY, System.Single fZ,
                System.Single fYaw, System.Single fPitch, System.Single fRoll,
                out System.Int32 pStatus)
```

Description

This method sets the value of a location V+ variable.

CAUTION: Extreme care must be taken while setting the value of an array element. If the array element did not previously exist, V+ will allocate it dynamically. This dynamic allocation may cause a V+ crash due to memory corruption, if a currently executing V+ task simultaneously accesses the same array. If the variable is an array element, it is safest to use this method only to modify existing variables.

Usage Considerations

This method is not available for MicroV+.

Input Parameters

bstrVariable	name of location variable
fX, fY, fZ, fYaw, fPitch, fRoll	components of the locations variable

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Examples

Visual Basic 6 Visual Basic .NET C# .NET

Visual Basic 6

```
'--------------------------------------------------------------------
' Subroutine that writes new location value to the controller
'--------------------------------------------------------------------
Private Sub setLocation(ByVal strLocationName As String, ByRef sngLoc() As Single)
    Dim mctrl As Activev2lib.MiscControl
    Dim stat As Long
    Dim str As String

    Set mctrl = New Activev2lib.MiscControl

    ' download new location to controller
    Call mctrl.SetL(comm, strLocationName, sngLoc(1), sngLoc(2), sngLoc(3), sngLoc(4), sngLoc(5), sngLoc(6), stat)

    If stat <> VE_SUCCESS Or Err.Number <> 0 Then
        If stat <> VE_SUCCESS Then
            Call errh.GetErrorString(stat, str)
        Else
            Call errh.GetErrorString(Err.Number, str)
        End If
        Call MsgBox(str, vbOKOnly, "Error attempting to Download Location " & strLocationName)
    End If

End Sub
```

Visual Basic .NET

```
'Set the location to the controller.
Private Function SetLocation(ByVal location As LocationDef)

    Dim nStatus As Integer

    Try
        ' Set the specific location.
        mMisc.SetL(mCommunications, location.sName, location.fX, location.fY, location.fZ, _
location.fYaw, location.fPitch, location.fRoll, nStatus)
        CheckError(nStatus)
    Catch Ex As System.Runtime.InteropServices.COMException
        ShowError(Ex)
    Catch Ex As Exception
        ShowError(Ex)
    End Try
End Function
```

C# .NET

```
    /// <summary>
    /// Set the location to the controller.
    /// </summary>
    /// <param name="location">Location structure with all the information to be set.</param>
    private void SetLocation(LocationDef location)
    {
         int nStatus;

         try
         {
              // Set the specific location.
              mMisc.SetL(mCommunications, location.sName, location.fX, location.fY, location.fZ,
                   location.fYaw, location.fPitch, location.fRoll, out nStatus);
              CheckError(nStatus);
         }
         catch (System.Runtime.InteropServices.COMException Ex)
         {
              ShowError(Ex);
         }
         catch (Exception Ex)
         {
              ShowError(Ex);
```

```
        }
    }
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
SET	SET

Related Topics

SetPP

SetR

SetS

SetPP Method

Visual Basic 6

```
MiscControl.SetPP(pCommunications as Communications, bstrVariable as String,
                lCount as Long, asValues as ArraySingle, pStatus as Long)
```

Visual Basic .NET

```
MiscControl.SetPP(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                bstrVariable As String, ByVal lCount As Integer, ByRef asValues
                As System.Array, ByRef pStatus As Integer)
```

C# .NET

```
MiscControl.SetPP(ActiveV2Lib.Communications pCommunications, System.String
                bstrVariable, System.Int32 lCount, ref System.Array asValues,
                out System.Int32 pStatus)
```

Description

This method sets the value of a precision point variable.

NOTE: For more information about Precision Points, see Locations and Precision Points.

Input Parameters

bstrVariable	Variable name. **NOTE:** Precision Point variable names must begin with the # character (for example, *#location_1*). If the string specified for this input does not begin with the # character, it will be added automatically.
ICount	Number of array elements to use from asValues.
asValues	Array of values to use in precision point variable.

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Examples

C# .NET

C# .NET

```
Main Class:
-----------

.....

////////////Write several Test PRECISION-POINTs to the AdeptController ///////////
     length = 5;     // number of precision points
     jts = 4;     // number of joints
     // Initialize 'ppName' Array of type String.
     ppName = new string[length];
     // Initialize 'ppValue' System.Array of type Float.
     Array ppValue = Array.CreateInstance( typeof(float), length*jts);
     for (i = 0; i < length; i++)
     {
          // Create one-dimensional ppName Array of type String.
          ppName[i] = "test.pp["+(i+1).ToString()+"]";
     }
     // Create 'ppValue' System.Array of type Float.
     for (j = ppValue.GetLowerBound(0); j <= ppValue.GetUpperBound(0); j++ )
     {
          ppValue.SetValue( (42*j), j);
     }

     ////////////////////////////////
     cx.SetVPlusPPoint(ppName, ppValue);
     ////////////////////////////////
```

```
.......

Business Class:
---------------

.......

     /// <summary>
     /// Set PrecisionPoint variable(s) in the AdeptController's RAM.
     /// </summary>
     /// <param name="ppName"></param>
     /// <param name="ppValue"></param>
     public void SetVPlusPPoint(string[] ppName, System.Array ppValue)
     {
          try {
               IEnumerator e = ppValue.GetEnumerator();


               int jts = ppValue.GetLength( ppValue.Rank-1 ) / ppName.Length;
               for (i= 0; i < ppName.Length; i++) {
                    string name = ppName[i];
                    Console.Write("Write PrecisionPoint {0}", name);
                    int j = 0;
                    while ( e.MoveNext() && (j <= jts)) {
                         if ( j < jts) {
                              Console.Write( "\t{0}", e.Current );
                              j++;
                         }
                         if ( j == (jts)) {
                              Console.WriteLine();
                              misc.SetPP(this.comm, name, jts, ref ppValue, out Success);
                              break;
                         }
                    }
               }
          }
          catch (System.Runtime.InteropServices.COMException e)
          {
               Console.WriteLine("SetVPlusPrecisionPoint - COM Exception: "+e.Message+e.ToString());
          }
     }

.......
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
#SET.POINT	N/A

Related Topics

SetL

SetR

SetS

SetR

Visual Basic 6

```
MiscControl.SetR(pCommunications as Communications, bstrVariable as String,
                 dValue as Double, pStatus as Long)
```

Visual Basic .NET

```
MiscControl.SetR(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                bstrVariable As String, ByVal dValue As Double, ByRef pStatus
                As Integer)
```

C# .NET

```
MiscControl.SetR(ActiveV2Lib.Communications pCommunications, System.String
                bstrVariable, System.Double dValue, out System.Int32 pStatus)
```

Description

This method sets the value of a real V+ variable.

CAUTION: Extreme care must be taken while setting the value of an array element. If the array element did not previously exist, V+ will allocate it dynamically. This dynamic allocation may cause a V+ crash due to memory corruption, if a currently executing V+ task simultaneously accesses the same array. If the variable is an array element, it is safest to use this method only to modify existing variables.

Input Parameters

bstrVariable
Name of real variable
dValue
New value of the variable

Output Parameters

pStatus
Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Examples

Visual Basic 6 Visual Basic .NET C# .NET

Visual Basic 6

```
'---------------------------------------------------------------------
' Pause button causes pause request flag to be set
'---------------------------------------------------------------------
Private Sub cmdPause_Click()
    Dim mctrl As Activev2lib.MiscControl
    Dim stat As Long
    Dim str As String

    Set mctrl = New Activev2lib.MiscControl

    ' set or clear pause flag depending on current pause state
    On Error Resume Next

    If appStatus And STATUS_BX_PAUSED Then
        Call mctrl.SetR(comm, "ckd.req.pause", 0, stat) ' clear pause flag
    Else
        Call mctrl.SetR(comm, "ckd.req.pause", -1, stat) ' set pause flag
    End If

    If stat <> VE_SUCCESS Or Err.Number <> 0 Then
        If stat <> VE_SUCCESS Then
            Call errh.GetErrorString(stat, str)
        Else
            Call errh.GetErrorString(Err.Number, str)
        End If
        Call MsgBox(str, vbOKOnly, "Error attempting to Pause")
    End If

End Sub
```

Visual Basic .NET

```
' Event-handler called by the system when the user clicks the
' Pause button. Toggle the value of the variable that tells
' the V+ application to pause or run.
```

```
Private Sub PauseButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles PauseButton.Click

    Dim nStatus As Integer

    Try
        If mnAppStatus And mlStatusBXPaused Then
            mMisc.SetR(mCommunications, msCkdReqPause, 0, nStatus)
        Else
            mMisc.SetR(mCommunications, msCkdReqPause, -1, nStatus)
        End If

        CheckError(nStatus)
    Catch Ex As System.Runtime.InteropServices.COMException
        ShowError(Ex)
    Catch Ex As Exception
        ShowError(Ex)
    End Try
End Sub
```

C# .NET

```
    /// <summary>
    /// Called when the user clicks the Pause button.
    /// Pause or resume the program on the controller.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void PauseButton_Click(object sender, System.EventArgs e)
    {
         int nStatus;

         try
         {
              if ((mnAppStatus & mlStatusBXPaused) == mlStatusBXPaused)
              {
                   // if pause, resume the program
                   mMisc.SetR(mCommunications, msCkdReqPause, 0, out nStatus);
              }
              else
              {
                   // if running, pause the program
                   mMisc.SetR(mCommunications, msCkdReqPause, -1, out nStatus);
              }
```

```
            CheckError(nStatus);
        }
        catch (System.Runtime.InteropServices.COMException Ex)
        {
            ShowError(Ex);
        }
        catch (Exception Ex)
        {
            ShowError(Ex);
        }
    }
```

Related Topics

SetL

SetPP

SetS

SetS

Visual Basic 6

```
IMiscControl.SetS(pCommunications as Communications, bstrVariable as String,
                bstrValue as String, pStatus as Long)
```

Visual Basic .NET

```
IMiscControl.SetS(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                bstrVariable As String, ByVal bstrValue As String, ByRef pStatus
                As Integer)
```

C# .NET

```
IMiscControl.SetS(ActiveV2Lib.Communications pCommunications, System.String
                bstrVariable, System.String bstrValue, out System.Int32 pStatus)
```

Description

This method sets the value of a V+ string variable.

CAUTION: Extreme care must be taken while setting the value of an array element. If the array element did not previously exist, V+ will allocate it dynamically. This dynamic allocation may cause a V+ crash due to memory corruption, if a currently executing V+ task simultaneously accesses the same array. If the variable is an array element, it is safest to use this method only to modify existing variables.

Usage Considerations

This method is not available for MicroV+.

Input Parameters

bstrVariable
Name of string variable
bstrValue
New value of the variable

Output Parameters

pStatus
Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Examples

C# .NET

C# .NET

```
Main Class:
-----------

....

/////// Write several Test STRINGs to the AdeptController ////////////////////

        length =5;
        sName = new string[length];
        sValue = new string[length];
        for (i = 0; i < length; i++)
        {
                sName[i] = "test["+i.ToString()+"]";
                sValue[i]= "Hello Travis on "+cx.IP.ToString()+"! This is an ActiveV+ Test.";
        }
        ////////////////////////////////
        cx.SetVPlusString(sName, sValue);
        ////////////////////////////////


....

Business Class:
---------------

....

        /// <summary>
```

```
        /// Set STRING variable(s) in the AdeptController's RAM.
        /// </summary>
        /// <param name="listStringNames:   Names of STRING Variables to write"></param>
        /// <param name="listStringValues:  Contents of the STRING Variables"></param>
        public void SetVPlusString(string[] sName, string[] sValue)
        {
                try
                {
                        for (i = 0; i < sName.Length; i++) {
                                Console.WriteLine("WriteString  \'{0}\': {1}",sName[i], sValue[i]);
misc.SetS(this.comm, sName[i], sValue[i], out Success);
                        }
                }
                catch (System.Runtime.InteropServices.COMException e) {
                        Console.WriteLine("SetVPlusString - COM Exception: "+e.Message);
                }
        }

....
```

Related Topics

SetL

SetPP

SetR

SetStackSize Method

Visual Basic 6

```
Programs.SetStackSize(pCommunications as Communications, lTaskNumber as Long,
                      fStackSize as Single, pStatus as Long)
```

Visual Basic .NET

```
Programs.SetStackSize(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                lTaskNumber As Integer, ByVal fStackSize As Single, ByRef pStatus
                As Integer)
```

C# .NET

```
Programs.SetStackSize(ActiveV2Lib.Communications pCommunications, System.Int32
                lTaskNumber, System.Single fStackSize, out System.Int32 pStatus)
```

Description

This method sets the stack size for the specified task.

Usage Considerations

This method is not available with MicroV+.

Input Parameters

lTaskNumber
Task number
fStackSize
Stack size in kilo−bytes

Output Parameters

pStatus
Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
STACK	STACK

Related Topics

Abort

Kill

Signal8 Method

Visual Basic 6

```
MiscControl.Signal8(pCommunications as Communications, lFirstSignal as Long,
                aSignal as ArrayLongl)
```

Visual Basic .NET

```
MiscControl.Signal8(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                lFirstSignal As Integer, ByRef aSignal As System.Array)
```

C# .NET

```
MiscControl.Signal8(ActiveV2Lib.Communications pCommunications, System.Int32
                lFirstSignal, ref System.Array aSignal)
```

Description

This method sets 8 digital I/O signals at once. The signals can be either on or off. No errors are returned, even for non−existent signals.

Input Parameters

lFirstSignal
First signal number to modify. This number will be rounded down to the closest multiple of 8 and be added to 1. For instance, specifying 10 will cause signals 9 through 16 to be modified.
aSignal
Array of longs specifying the desired state of the corresponding signal. A value of 0 turns the signal off and a value of 1 turns the signal on. The first index is 1.

Output Parameters

None.

Examples

Visual Basic .NET C# .NET

Visual Basic .NET

```
' Will get the address from the UI and set the value to the controller.
Protected Function SetValue()

    Dim nAddress As Integer

    Try
        nAddress = cmbAddress.Text

        Select Case SelectionToEnum(cmbByte.SelectedIndex)
                ...
            Case ComboByte._8Bits
                Dim j As Integer
                Dim i As Integer

                ' Set 8 bits at the same time.

                ' Declare an array with the appropriate size AND the lower bound set to 1.
                Dim nSize() As Integer = {8}
                Dim nLowerBound() As Integer = {1}
                Dim Value As System.Array = Array.CreateInstance(GetType(Integer), nSize, nLowerBound)

                ' Set the values first by reading the panel.
                j = 0
                For i = Value.GetLowerBound(0) To Value.GetUpperBound(0)
                    If mBits(j).BackColor.ToArgb = Color.Red.ToArgb Then
                        Value.SetValue(1, i)
                    Else
                        Value.SetValue(0, i)
                    End If
                    j = j + 1
                Next

                ' Set the value to the controller.
                mMisc.Signal8(mCommunications, nAddress, Value)
        End select
    Catch Ex As System.Runtime.InteropServices.COMException
        ShowError(Ex)
    Catch Ex As Exception
```

```
            ShowError(Ex)
        End Try
    End Function
```

C# .NET

```
Main Class:
-----------

// Set eight consecutive digital outputs on the AdeptController //////////

// Initialize and create 'dios' System.Array of type int32.
int[] nSize = {8};
int[] nLowerBound = {1};
Array dios = Array.CreateInstance(typeof(System.Int32), nSize, nLowerBound);

//'0' turns the signal OFF and '1' turns the signal ON
dios.SetValue( 1, 1);    //LSB ON
dios.SetValue( 0, 2);
dios.SetValue( 1, 3);
dios.SetValue( 0, 4);
dios.SetValue( 0, 5);
dios.SetValue( 1, 6);
dios.SetValue( 0, 7);    //output 7 OFF
dios.SetValue( 1, 8);    //output 8 ON

//////////////////////////
cx.Set8Outputs(0097,dios);
//////////////////////////

....

Business Class:
---------------

/// <summary>
/// This routine sets eight consecutive digital I/O signals at once.
/// </summary>
/// <param name="firstDio - first of the eight consecutive outputs to set"></param>
/// <param name="ioStatus - int32 array specifying the desired states of the corresponding eight signals.
/// A value of 0 turns the signal off and a value of 1 turns the signal on. The first index is 1."></param>
public void Set8Outputs(int firstDio, System.Array ioStatus)
```

```
{
        try {
                Console.WriteLine("Set Outputs {0} to {1} as: {2} {3} {4} {5} {6} {7} {8} {9}", firstDio, (firstDio+7),
                                        ioStatus.GetValue(8), ioStatus.GetValue(7), ioStatus.GetValue(6), ioStatus.GetValue(5),
                                        ioStatus.GetValue(4), ioStatus.GetValue(3), ioStatus.GetValue(2), ioStatus.GetValue(1));
misc.Signal8(comm, firstDio, ref ioStatus);
        }
        catch (System.Runtime.InteropServices.COMException e) {
                Console.WriteLine("Set8Outputs - COM Exception: "+e.Message+e.ToString());
        }
}
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
BITS	BITS

Related Topics

IO

SignalOff

SignalOn

SignalOff Method

Visual Basic 6

```
MiscControl.SignalOff(pCommunications as Communications, lSignalNumber as Long)
```

Visual Basic .NET

```
MiscControl.SignalOff(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
               lSignalNumber As Integer)
```

C# .NET

```
MiscControl.SignalOff(ActiveV2Lib.Communications pCommunications, System.Int32
               lSignalNumber)
```

Description

This method turns a DIO signal off. All errors, including illegal signal numbers, are ignored.

Input Parameters

lSignalNumber
Signal number

Output Parameters

None.

Examples

Visual Basic 6 Visual Basic .NET C# .NET

Visual Basic 6

```
'---------------------------------------------------------------------
' Toggle button checks current state of IO and sets to opposite state
'---------------------------------------------------------------------
Private Sub cmdToggle_Click()
```

```
    Dim st As Activev2lib.status
    Dim sigState As Long
    Dim stat As Long
    Dim valid As Long
    Dim sigNum As Long
    Dim str As String
    Dim temp As Long

    ' instantiate MiscControl object
    Set mctrl = New Activev2lib.MiscControl

    ' Error handler in case what is in the text box is not a number
    On Error GoTo ErrorHandler

    ' Extract signal number displayed in text box
    sigNum = Int(txtIONumber.Text)

    ' action depends on signal state
    Select Case iostate(sigNum)
      Case 0    ' turn signal on
        stat = mctrl.SignalOn(comm, sigNum)

      Case 1   ' turn signal off
        stat = mctrl.SignalOff(comm, sigNum)

      Case Else ' Error condition - throw error message
        Call errh.GetErrorString(stat, str)
        Call MsgBox(str, vbOKOnly, "Error attempting to toggle IO")
    End Select

    Exit Sub

ErrorHandler:
    Call MsgBox("Invalid signal number entered", vbOKOnly, "Error attempting to toggle IO")
End Sub
```

Visual Basic .NET

```
' Get the address from the UI and set the value to the controller.
Protected Function SetValue()

    Dim nAddress As Integer
```

```
    Try
        nAddress = cmbAddress.Text

        Select Case SelectionToEnum(cmbByte.SelectedIndex)
            Case ComboByte._1Bit
                If mBits(0).BackColor.ToArgb = Color.Red.ToArgb Then
                    ' Set the bit
                    mMisc.SignalOn(mCommunications, nAddress)

                Else
                    ' Reset the bit.
                    mMisc.SignalOff(mCommunications, nAddress)
                End If
            Case ComboByte._8Bits
          ...
     End select
    Catch Ex As System.Runtime.InteropServices.COMException
        ShowError(Ex)
    Catch Ex As Exception
        ShowError(Ex)
    End Try
End Function
```

C# .NET

```
    /// <summary>
    /// Get the address from the UI and set the value to the controller.
    /// </summary>
    protected void SetValue()
    {
         int nAddress;


         try
         {
              nAddress = int.Parse(cmbAddress.Text);

              switch(SelectionToEnum(cmbByte.SelectedIndex))
              {
                   case ComboByte._1Bit:

                        if (mBits[0].BackColor == Color.Red)
                        {
```

```
                    // Set the bit
                    mMisc.SignalOn(mCommunications, nAddress);
               }
               else
               {
                    // Reset the bit
                    mMisc.SignalOff(mCommunications, nAddress);
               }

               break;
          case ComboByte._8Bits:

               ...

               break;
     }
  }
  catch (System.Runtime.InteropServices.COMException Ex)
  {
       ShowError(Ex);
  }
  catch (Exception Ex)
  {
       ShowError(Ex);
  }
}
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
SIGNAL	SIGNAL

Related Topics

IO

Signal8

SignalOn

SignalOn Method

Visual Basic 6

```
MiscControl.SignalOn(pCommunications as Communications, lSignalNumber as Long)
```

Visual Basic .NET

```
MiscControl.SignalOn(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
               lSignalNumber As Integer)
```

C# .NET

```
MiscControl.SignalOn(ActiveV2Lib.Communications pCommunications, System.Int32
               lSignalNumber)
```

Description

This method turns a DIO signal on. All errors, including illegal signal numbers are ignored.

Input Parameters

lSignalNumber
signal number

Output Parameters

None.

Examples

Visual Basic 6 Visual Basic .NET C# .NET

Visual Basic 6

```
'---------------------------------------------------------------------
' Toggle button checks current state of IO and sets to opposite state
'---------------------------------------------------------------------
Private Sub cmdToggle_Click()
```

```
        Dim st As Activev2lib.status
        Dim sigState As Long
        Dim stat As Long
        Dim valid As Long
        Dim sigNum As Long
        Dim str As String
        Dim temp As Long

        ' instantiate MiscControl object
        Set mctrl = New Activev2lib.MiscControl

        ' Error handler in case what is in the text box is not a number
        On Error GoTo ErrorHandler

        ' Extract signal number displayed in text box
        sigNum = Int(txtIONumber.Text)

        ' action depends on signal state
        Select Case iostate(sigNum)
          Case 0    ' turn signal on
            stat = mctrl.SignalOn(comm, sigNum)

          Case 1   ' turn signal off
            stat = mctrl.SignalOff(comm, sigNum)

          Case Else ' Error condition - throw error message
            Call errh.GetErrorString(stat, str)
            Call MsgBox(str, vbOKOnly, "Error attempting to toggle IO")
        End Select

        Exit Sub

    ErrorHandler:
        Call MsgBox("Invalid signal number entered", vbOKOnly, "Error attempting to toggle IO")
    End Sub
```

Visual Basic .NET

```
' Get the address from the UI and set the value to the controller.
Protected Function SetValue()

    Dim nAddress As Integer
```

```
    Try
        nAddress = cmbAddress.Text

        Select Case SelectionToEnum(cmbByte.SelectedIndex)
            Case ComboByte._1Bit
                If mBits(0).BackColor.ToArgb = Color.Red.ToArgb Then
                    ' Set the bit
                    mMisc.SignalOn(mCommunications, nAddress)

                Else
                    ' Reset the bit
                    mMisc.SignalOff(mCommunications, nAddress)
                End If
            Case ComboByte._8Bits
          ...
     End select
    Catch Ex As System.Runtime.InteropServices.COMException
        ShowError(Ex)
    Catch Ex As Exception
        ShowError(Ex)
    End Try
End Function
```

C# .NET

```
    /// <summary>
    /// Get the address from the UI and set the value to the controller.
    /// </summary>
    protected void SetValue()
    {
         int nAddress;


         try
         {
              nAddress = int.Parse(cmbAddress.Text);

              switch(SelectionToEnum(cmbByte.SelectedIndex))
              {
                   case ComboByte._1Bit:

                        if (mBits[0].BackColor == Color.Red)
                        {
```

```
                    // Set the bit
                    mMisc.SignalOn(mCommunications, nAddress);
               }
               else
               {
                    // Reset the bit
                    mMisc.SignalOff(mCommunications, nAddress);
               }

               break;
          case ComboByte._8Bits:

               ...

               break;
     }
  }
  catch (System.Runtime.InteropServices.COMException Ex)
  {
       ShowError(Ex);
  }
  catch (Exception Ex)
  {
       ShowError(Ex);
  }
}
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
SIGNAL	SIGNAL

Related Topics

IO

Signal8

SignalOff

Speed Method

Visual Basic 6

```
MiscControl.Speed(pCommunications as Communications, fSpeed as float, pStatus as Long)
```

Visual Basic .NET

```
MiscControl.Speed(ByVal pCommunications As ActiveV2Lib.Communications, ByVal fSpeed As
                Single, ByRef pStatus As Integer)
```

C# .NET

```
MiscControl.Speed(ActiveV2Lib.Communications pCommunications, System.Single fSpeed,
                out System.Int32 pStatus)
```

Description

This method sets the robot speed.

Input Parameters

fSpeed
Robot speed as a percentage of the maximum speed

Output Parameters

pStatus
Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Examples

Visual Basic 6 Visual Basic .NET C# .NET

Visual Basic 6

```
'-----------------------------------------------------------------------
' Set robot monitor speed
'-----------------------------------------------------------------------
```

```
    Private Sub hscMonitorSpeed_Change()
        Dim mctrl As Activev2lib.MiscControl
        Dim stat As Long
        Dim str As String

        Set mctrl = New Activev2lib.MiscControl

        On Error Resume Next
        Call mctrl.Speed(comm, hscMonitorSpeed.Value, stat)
        If stat <> VE_SUCCESS Or Err.Number <> 0 Then
            If stat <> VE_SUCCESS Then
                Call errh.GetErrorString(stat, str)
            Else
                Call errh.GetErrorString(Err.Number, str)
            End If
            Call MsgBox(str, vbOKOnly, "Error attempting to set Monitor Speed")
        End If

    End Sub
```

Visual Basic .NET

```
' Event-handler called by the system when the user changes the
' speed trackbar value. Update monitor speed.
Private Sub SpeedTrackBar_Scroll(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SpeedTrackBar.Scroll

    Dim nStatus As Integer

    Try
        mMisc.Speed(mCommunications, SpeedTrackBar.Value, nStatus)
        CheckError(nStatus)
    Catch Ex As System.Runtime.InteropServices.COMException
        ShowError(Ex)
    Catch Ex As Exception
        ShowError(Ex)
    End Try

End Sub
```

C# .NET

```
/// <summary>
/// Called when the user change the speed trackbar value.
/// Update the monitor speed.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void SpeedTrackBar_Scroll(object sender, System.EventArgs e)
{
     int nStatus;

     try
     {
          mMisc.Speed(mCommunications, SpeedTrackBar.Value, out nStatus);

          CheckError(nStatus);
     }
     catch (System.Runtime.InteropServices.COMException Ex)
     {
          ShowError(Ex);
     }
     catch (Exception Ex)
     {
          ShowError(Ex);
     }
}
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
SPEED	SPEED

SStep Method

Visual Basic 6

```
Programs.SStep(pCommunications as Communications, lTaskNumber as Long, pStatus
               as Long)
```

Visual Basic .NET

```
Programs.SStep(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
               lTaskNumber As Integer, ByRef pStatus As Integer)
```

C# .NET

```
Programs.SStep(ActiveV2Lib.Communications pCommunications, System.Int32
               lTaskNumber, out System.Int32 pStatus)
```

Description

This method executes the next line of the specific V+ task. If the next line happens to be a subroutine, this method steps across the subroutine (as opposed to stepping into the subroutine).

Usage Considerations

During execution, this method steps across subroutine calls as opposed to stepping into them. To step into subroutines, use the XStep method. To execute a specific line in a program, use XStep2.

Input Parameters

lTaskNumber	task number

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
SSTEP	SSTEP

Related Topics

XStep

XStep2

StackContents Method

Visual Basic 6

```
Status.StackContents(pCommunications as Communications, lTaskNumber as Long,
                pCount as Long, pState as Long, aCurrentProg as ArrayString,
                aCurrentStep as ArrayLong, pCurrentStack as Single, pMaxStack
                as Single, pStackLimit as Single, pStatus as Long)
```

Visual Basic .NET

```
Status.StackContents(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                lTaskNumber As Integer, ByRef pCount As Integer, ByRef pState As
                Integer, ByRef aCurrentProg As System.Array, ByRef aCurrentStep
                As System.Array, ByRef pCurrentStack As Single, ByRef pMaxStack
                As Single, ByRef pStackLimit As Single, ByRef pStatus As Integer)
```

C# .NET

```
Status.StackContents(ActiveV2Lib.Communications pCommunications, System.Int32
                lTaskNumber, ref System.Int32 pCount, out System.Int32 pState,
                out System.Array aCurrentProg, out System.Array aCurrentStep,
                out System.Single pCurrentStack, out System.Single pMaxStack,
                out System.Single pStackLimit, out System.Int32 pStatus)
```

Description

This method returns the contents and settings of the stack for a specified task.

Input Parameters

lTaskNumber
Task number.
pCount
Number of programs on stack to be returned.

Output Parameters

pCount
Number of programs on stack actually returned.
pState
0 if task is idle
1 if task is active
aCurrentProg
Array of program names on stack.
aCurrentStep
Array of program steps corresponding to aCurrentProg.
pCurrentStack
Size of current stack.
pMaxStack
Maximum used size of stack.
pStackLimit
Maximum space allocated for the stack.
pStatus
Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
STACK	N/A

Related Topics

SetStackSize

Status

Visual Basic 6

```
Status.Status(pCommunications as Communications, pCount as Long,
              aTask as ArrayLong, aState as ArrayLong,
              aMainProg as ArrayString, aCurrentProg as ArrayString,
              aCurrentStep as ArrayLong, aStack as ArraySingle)
```

Visual Basic .NET

```
Status.Status(ByVal pCommunications As ActiveV2Lib.Communications, ByRef
               pCount As Integer, ByRef aTask As System.Array, ByRef aState
               As System.Array, ByRef aMainProg As System.Array, ByRef
               aCurrentProg As System.Array, ByRef aCurrentStep As
               System.Array, ByRef aStack As System.Array)
```

C# .NET

```
Status.Status(ActiveV2Lib.Communications pCommunications, ref System.Int32
               pCount, out System.Array aTask, out System.Array aState,
               out System.Array aMainProg, out System.Array aCurrentProg,
               out System.Array aCurrentStep, out System.Array aStack)
```

Description

This method gets the system task/program execution status. See the links to the corresponding V+ or MicroV+ keywords for more information.

Input Parameters

pCount	maximum number of array elements to return

Output Parameters

pCount	actual number of array elements
aTask	array of task numbers
aState	

	array of task states: 0 means not running 1 means running
aMainProg	array of names of main program
aCurrentProg	array of names of current program
aCurrentStep	array of current step
aStack	array of stack size

Examples

Visual Basic 6

Visual Basic 6

```
' read program status
lngpCount = 25
Call status.status(comm, lngpCount, lngTask(), lngState(), strMainProg(), _
strCurrProg(), lngCurrStep(), sngStackSize())
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
STATUS	STATUS

Store Method

Visual Basic 6

```
Programs.Store(pCommunications as Communications, bstrProgName as String,
               bstrFileName as String, pStatus as Long
```

Visual Basic .NET

```
Programs.Store(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
               bstrProgName As String, ByVal bstrFileName As String, ByRef
               pStatus As Integer)
```

C# .NET

```
Programs.Store(ActiveV2Lib.Communications pCommunications, System.String
               bstrProgName, System.String bstrFileName, ref System.Int32
               pStatus)
```

Description

Saves programs and variables currently in memory to disk using the V+ STORE monitor command. See the links to the corresponding V+ or MicroV+ keywords for more information.

NOTE: On a MicroV+ system, this routine stores ALL programs and variables in the NVRAM.

Input Parameters

bstrProgName
Name of program name to save. For MicroV+, this must be an empty string.
bstrFileName
Name of disk file. For MicroV+, this must be an empty string.

Output Parameters

pStatus
Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
STORE	STORE

Related Topics

Load

Storem

Storep

Storem Method

Visual Basic 6

```
Programs.Storem(pCommunications as Communications, bstrModuleName as String,
                bstrFileName  as String, storeReals as Long, storeLocations
                as Long, storeStrings as Long, pStatus  as Long)
```

Visual Basic .NET

```
Programs.Storem(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                bstrModuleName As String, ByVal bstrFileName As String, ByVal
                storeReals As Integer, ByVal storeLocations As Integer, ByVal
                storeStrings As Integer, ByRef pStatus As Integer)
```

C# .NET

```
Programs.Storem(ActiveV2Lib.Communications pCommunications, System.String
                bstrModuleName, System.String bstrFileName, System.Int32
                storeReals, System.Int32 storeLocations, System.Int32
                storeStrings, out System.Int32 pStatus)
```

Description

This method stores a program module onto disk.

Usage Considerations

This method is not available for MicroV+.

Input Parameters

bstrModuleName
Module name
bstrFileName
The name of the file where the programs will be stored. This string consists of an optional physical device, an optional disk unit, an optional directory path, a file name, and an optional file extension.
storeReals

storeLocations
storeStrings
These fields determine whether variables of the specified type referenced directly by the module are to be stored. For each type of variable to be stored, enter a value:
Value
Description
1
Specify 1 to store the variables.
0
Specify 0 if you do not want to store the variables.

Output Parameters

pStatus
Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
STOREM	STOREM

Related Topics

Load

Store

Storep

Storep Method

Visual Basic 6

```
Programs.Storep(pCommunications as Communications, bstrProgName as String,
                bstrFileName  as String, pStatus as Long)
```

Visual Basic .NET

```
Programs.Storep(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                bstrProgName As String, ByVal bstrFileName As String, ByRef
                pStatus As Integer)
```

C# .NET

```
Programs.Storep(ActiveV2Lib.Communications pCommunications, System.String
                bstrProgName, System.String bstrFileName, ref System.Int32
                pStatus)
```

Description

This method saves a program to disk using the STOREP monitor command.

Usage Considerations

This method is not available for MicroV+.

Input Parameters

bstrProgName
Name of program name to save.
bstrFileName
Name of disk file.

Output Parameters

pStatus
Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
STOREP	N/A

Related Topics

Load

Store

Storem

Svariables Method

Visual Basic 6

```
Status.Svariables(pCommunications as Communications, lTask as Long,
                bstrProgram as String, bstrName as String, pCount as Long,
                aNames as ArrayString)
```

Visual Basic .NET

```
Status.Svariables(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                lTask As Integer, ByVal bstrProgram As String, ByVal bstrName
                As String, ByRef pCount As Integer, ByRef aNames As System.Array)
```

C# .NET

```
Status.Svariables(ActiveV2Lib.Communications pCommunications, System.Int32
                lTask, System.String bstrProgram, System.String bstrName,
                ref System.Int32 pCount, out System.Array aNames)
```

Description

This method returns the names of string variables. Only the variable names are returned.

Usage Considerations

This method is not available for MicroV+.

Input Parameters

bstrName
Name (may contain wildcards). An empty string returns the names of all string variables.
pCount
Number of names to return.
lTask
Task number for context. A value of −1 indicates that no task is specified.
bstrProgram
Program name for context (can be an empty string).

Output Parameters

pCount
Actual number of names returned.
aNames
Array of names.

Examples

C# .NET

C# .NET

```
int   nCount = 10;      // Only request the first 10 string variables names.
Array Names;            // Storages for the variables names.
try
{
     // Get the list of location variables.
     mStatus.Svariables(mCommunications, -1, "", "", ref nCount, out Names);
}
catch (System.Runtime.InteropServices.COMException Ex)
{
     ShowError(Ex);
     return;
}
catch (Exception Ex)
{
     ShowError(Ex);
     return;
}
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
LISTS	N/A

Related Topics

Listr

Listl

Lists

Lvariables

Rvariables

Switch Method

Visual Basic 6

```
Status.Switch(pCommunications as Communications, bstrSwitchName as String,
               pCount as Long, aSwitchName as ArrayString, aSwitchNumber
           , aSwitchState as ArrayLong)
```

Visual Basic .NET

```
Status.Switch(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
               bstrSwitchName As String, ByRef pCount As Integer, ByRef
               aSwitchName As System.Array, ByRef aSwitchNumber As
               System.Array, ByRef aSwitchState As System.Array)
```

C# .NET

```
Status.Switch(ActiveV2Lib.Communications pCommunications, System.String
               bstrSwitchName, ref System.Int32 pCount, out System.Array
               aSwitchName, out System.Array aSwitchNumber, out System.Array
               aSwitchState)
```

Description

This method returns the state of system switches. All switches are treated as array switches.

As an example, let's say that the system switches consist of:

X On

Y Off

Z 1: On 2: On 3: Off 4: Off

Then the output arrays contain the following:

rsaSwitchName(1) = X, rsaSwitchNumber(1) = 1, rsaSwitchState(1) = 1

rsaSwitchName(2) = Y, rsaSwitchNumber(2) = 1, rsaSwitchState(2) = 0

rsaSwitchName(3) = Z, rsaSwitchNumber(3) = 1, rsaSwitchState(3) = 1

rsaSwitchName(4) = Z, rsaSwitchNumber(4) = 2, rsaSwitchState(4) = 1

rsaSwitchName(5) = Z, rsaSwitchNumber(5) = 3, rsaSwitchState(5) = 0

rsaSwitchName(6) = Z, rsaSwitchNumber(6) = 4, rsaSwitchState(6) = 0

NOTE: Not available with MicroV+.

Input Parameters

bstrSwitchName
The name of the system switch. This will normally be an empty string, which means to return all system switches. It can also contain "V" to return all vision switches, or "[2]" to return all non−array switches plus all second elements of the array.
pCount
The number of array elements to return. Note that array switches will occupy as many array locations as there are switches by that name.

Output Parameters

pCount
Number of array elements actually returned
aSwitchName
Array of switch names. Note that the names of array switches will occur in multiple aSwitchName indexes.
aSwitchNumber
Switch number.
aSwitchState
Switch state.

Examples

Visual Basic 6 Visual Basic .NET C# .NET

Visual Basic 6

```
'---------------------------------------------------------------------
' Enable Power button toggles robot power
'---------------------------------------------------------------------
Private Sub cmdEnablePower_Click()
    Dim mctrl As Activev2lib.MiscControl
    Dim status As Activev2lib.status
    Dim stat As Long
    Dim str As String
    Dim strName() As String
    Dim lngNumber() As Long
    Dim lngState() As Long
    Dim lngCount As Long

    Set mctrl = New Activev2lib.MiscControl
    Set status = New Activev2lib.status

    On Error Resume Next

    ' Get Current Power status
    lngCount = 1
    Call status.Switch(comm, "POWER", lngCount, strName(), lngNumber(), _
    lngState())

    ' Enable or diable switch depending on existing state
    If lngState(1) Then 'disable
        Call mctrl.SwitchOff(comm, "POWER", stat)
        cmdEnablePower.Caption = "Enable Power"
    Else    'enable
        Call mctrl.SwitchOn(comm, "POWER", stat)
        cmdEnablePower.Caption = "Disable Power"
    End If

    If stat <> VE_SUCCESS Or Err.Number <> 0 Then
        If stat <> VE_SUCCESS Then
            Call errh.GetErrorString(stat, str)
        Else
            Call errh.GetErrorString(Err.Number, str)
        End If
        Call MsgBox(str, vbOKOnly, "Error attempting to enable/disable power")
    End If
```

```
    End Sub
```

Visual Basic .NET

```
' Called every time the timer is triggered.
' Set the state of the menu item based on the status read from the controller.
Private Sub Timer_Tick(ByVal sender As System.Object, ByVal e As System.Timers.ElapsedEventArgs) Handles Timer.Elapsed

    ' If we are not connected (or lost the connection) disable most
    ' of the interface.
    If (mCommunications Is Nothing) Or (mCommunications.IsOnline = 0) Then
        DisableAll()
        Exit Sub
    End If

    Dim names As System.Array
    Dim values As System.Array
    Dim numbers As System.Array
    Dim status As Integer
    Dim i As Integer

    Try

     ...

        ' Read the current status of High Power, so we can update the
        ' UI elements accordingly.
        mStatus.Switch(mCommunications, msSwitchPower, 1, names, numbers, values)

        If values.GetValue(1) = 0 Then
            PowerButton.Text = "Enable Power"
            mbPowerOn = False
            StartButton.Enabled = False
            PauseButton.Enabled = False
            StopButton.Enabled = False
            CalibrateButton.Enabled = False
        Else
            PowerButton.Text = "Disable Power"
            mbPowerOn = True
            CalibrateButton.Enabled = True

            ' Update other UI elements, depending on status bits.
```

```
            If mnAppStatus And mlStatusBXRun Then
                StartButton.Enabled = False
                StopButton.Enabled = True
                PauseButton.Enabled = True
                If mnAppStatus And mlStatusBXPaused Then
                    PauseButton.Text = "Continue"
                Else
                    PauseButton.Text = "Pause"
                End If
            Else
                StartButton.Enabled = True
                PauseButton.Enabled = False
                StopButton.Enabled = False
            End If
        End If
    Catch Ex As System.Runtime.InteropServices.COMException
        ShowError(Ex)
    Catch Ex As Exception
        ShowError(Ex)
    End Try
End Sub
```

C# .NET

```
/// <summary>
/// Called whenever the timer is triggered.
/// Set the state of the menu item based on the status read from the controller.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void Timer_Elapsed(object sender, System.Timers.ElapsedEventArgs e)
{
     // Ensure we have a communication establish before running this code.
     if (mCommunications == null ||
          mCommunications.IsOnline == 0)
     {
          DisableAll();
          return;
     }


     Array names;
     Array values;
```

```
Array numbers;
int   nStatus;
int   nRef;

try
{

     ...

     // Read the current status of High Power so we can update
     // UI elements accordingly.
     nRef = 1;
     mStatus.Switch(mCommunications, msSwitchPower, ref nRef, out names, out numbers, out values);

     if (((int) values.GetValue(1)) == 0L)
     {
          PowerButton.Text = "Enable Power";
          mbPowerOn = false;
          StartButton.Enabled = false;
          PauseButton.Enabled = false;
          StopButton.Enabled = false;
          CalibrateButton.Enabled = false;
     }
     else
     {
          PowerButton.Text = "Disable Power";
          mbPowerOn = true;
          CalibrateButton.Enabled = true;

          // Update other UI elements, depending on status bits.
          if ((mnAppStatus & mlStatusBXRun) != 0)
          {
               StartButton.Enabled = false;
               StopButton.Enabled = true;
               PauseButton.Enabled = true;
               if ((mnAppStatus & mlStatusBXPaused) != 0)
               {
                    PauseButton.Text = "Continue";
               }
               else
               {
                    PauseButton.Text = "Pause";
               }
```

```
                }
                else
                {
                     StartButton.Enabled = true;
                     PauseButton.Enabled = false;
                     StopButton.Enabled = false;
                }
           }
      }
      catch (System.Runtime.InteropServices.COMException Ex)
      {
           ShowError(Ex);
      }
      catch (Exception Ex)
      {
           ShowError(Ex);
      }
}
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
SWITCH	SWITCH

Related Topics

Parameter

SwitchOff Method

Visual Basic 6

```
MiscControl.SwitchOff(pCommunications as Communications, bstrSwitchName as
               String, pStatus as Long)
```

Visual Basic .NET

```
MiscControl.SwitchOff(ByVal pCommunications As ActiveV2Lib.Communications,
               ByVal bstrSwitchName As String, ByRef pStatus As Integer)
```

C# .NET

```
MiscControl.SwitchOff(ActiveV2Lib.Communications pCommunications,
               System.String bstrSwitchName, out System.Int32 pStatus)
```

Description

This method disables a system switch. No errors are returned.

Input Parameters

bstrSwitchName
Switch name

Output Parameters

pStatus
Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Examples

Visual Basic 6 Visual Basic .NET C# .NET

Visual Basic 6

```
'---------------------------------------------------------------------
' Enable Power button toggles robot power
'---------------------------------------------------------------------
Private Sub cmdEnablePower_Click()
    Dim mctrl As Activev2lib.MiscControl
    Dim status As Activev2lib.status
    Dim stat As Long
    Dim str As String
    Dim strName() As String
    Dim lngNumber() As Long
    Dim lngState() As Long
    Dim lngCount As Long

    Set mctrl = New Activev2lib.MiscControl
    Set status = New Activev2lib.status

    On Error Resume Next

    ' Get Current Power status
    lngCount = 1
    Call status.Switch(comm, "POWER", lngCount, strName(), lngNumber(), _
    lngState())

    ' Enable or diable switch depending on existing state
    If lngState(1) Then 'disable
        Call mctrl.SwitchOff(comm, "POWER", stat)
        cmdEnablePower.Caption = "Enable Power"
    Else    'enable
        Call mctrl.SwitchOn(comm, "POWER", stat)
        cmdEnablePower.Caption = "Disable Power"
    End If

    If stat <> VE_SUCCESS Or Err.Number <> 0 Then
        If stat <> VE_SUCCESS Then
            Call errh.GetErrorString(stat, str)
        Else
            Call errh.GetErrorString(Err.Number, str)
        End If
        Call MsgBox(str, vbOKOnly, "Error attempting to enable/disable power")
    End If
```

```
    End Sub
```

Visual Basic .NET

```
' Event-handler called by the system when the user clicks the
' Power button. Toggle High Power on/off.
Private Sub PowerButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles PowerButton.Click

    Dim nStatus As Integer

    ' This function may take some time to execute. Change the curser to an hourglass
    ' to let the user know it is processing.
    Cursor = Cursors.WaitCursor

    Try
        If mbPowerOn Then
            mMisc.SwitchOff(mCommunications, msSwitchPower, nStatus)
        Else
            mMisc.SwitchOn(mCommunications, msSwitchPower, nStatus)
        End If

        CheckError(nStatus)
    Catch Ex As System.Runtime.InteropServices.COMException
        ShowError(Ex)
    Catch Ex As Exception
        ShowError(Ex)
    End Try

    ' Return to the normal cursor.
    Cursor = Cursors.Default

End Sub
```

C# .NET

```
    /// <summary>
    /// Called when the user clicks the Power button.
    /// Toggle High Power on/off.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void PowerButton_Click(object sender, System.EventArgs e)
```

```
{
     int nStatus;

     // This function may take some time to execute. Change the curser to an hourglass
     // to let the user know it is processing.
     Cursor = Cursors.WaitCursor;

     try
     {
          if (mbPowerOn)
          {
               mMisc.SwitchOff(mCommunications, msSwitchPower, out nStatus);
          }
          else
          {
               mMisc.SwitchOn(mCommunications, msSwitchPower, out nStatus);
          }

          CheckError(nStatus);
     }
     catch (System.Runtime.InteropServices.COMException Ex)
     {
          ShowError(Ex);
     }
     catch (Exception Ex)
     {
          ShowError(Ex);
     }

     // Return to the normal cursor.
     Cursor = Cursors.Default;
}
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
DISABLE	DISABLE

Related Topics

SwitchOn

SwitchOn Method

Visual Basic 6

```
MiscControl.SwitchOn(pCommunications as Communications, bstrSwitchName as String,
               pStatus as Long)
```

Visual Basic .NET

```
MiscControl.SwitchOn(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
               bstrSwitchName As String, ByRef pStatus As Integer)
```

C# .NET

```
MiscControl.SwitchOn(ActiveV2Lib.Communications pCommunications, System.String
               bstrSwitchName, out System.Int32 pStatus)
```

Description

This method enables a system switch. No errors are returned.

Input Parameters

bstrSwitchName
switch name

Output Parameters

pStatus
Returns 1 (VE_SUCCESS) if the operation is successful, or a V+ error code if the operation fails.

Examples

Visual Basic 6 Visual Basic .NET C# .NET

Visual Basic 6

```
'---------------------------------------------------------------------
' Enable Power button toggles robot power
'---------------------------------------------------------------------
Private Sub cmdEnablePower_Click()
    Dim mctrl As Activev2lib.MiscControl
    Dim status As Activev2lib.status
    Dim stat As Long
    Dim str As String
    Dim strName() As String
    Dim lngNumber() As Long
    Dim lngState() As Long
    Dim lngCount As Long

    Set mctrl = New Activev2lib.MiscControl
    Set status = New Activev2lib.status

    On Error Resume Next

    ' Get Current Power status
    lngCount = 1
    Call status.Switch(comm, "POWER", lngCount, strName(), lngNumber(), _
    lngState())

    ' Enable or diable switch depending on existing state
    If lngState(1) Then 'disable
        Call mctrl.SwitchOff(comm, "POWER", stat)
        cmdEnablePower.Caption = "Enable Power"
    Else    'enable
        Call mctrl.SwitchOn(comm, "POWER", stat)
        cmdEnablePower.Caption = "Disable Power"
    End If

    If stat <> VE_SUCCESS Or Err.Number <> 0 Then
        If stat <> VE_SUCCESS Then
            Call errh.GetErrorString(stat, str)
        Else
            Call errh.GetErrorString(Err.Number, str)
        End If
        Call MsgBox(str, vbOKOnly, "Error attempting to enable/disable power")
    End If
```

```
    End Sub
```

Visual Basic .NET

```
' Event-handler called by the system when the user clicks the
' Power button. Toggle High Power on/off.
Private Sub PowerButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles PowerButton.Click

    Dim nStatus As Integer

    ' This function may take some time to execute. Change the curser to an hourglass
 ' to let the user know it is processing.
    Cursor = Cursors.WaitCursor

    Try
        If mbPowerOn Then
            mMisc.SwitchOff(mCommunications, msSwitchPower, nStatus)
        Else
            mMisc.SwitchOn(mCommunications, msSwitchPower, nStatus)
        End If

        CheckError(nStatus)
    Catch Ex As System.Runtime.InteropServices.COMException
        ShowError(Ex)
    Catch Ex As Exception
        ShowError(Ex)
    End Try

    ' Return to the normal cursor.
    Cursor = Cursors.Default

End Sub
```

C# .NET

```
    /// <summary>
    /// Called when the user clicks the Power button.
    /// Toggle High Power on/off.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void PowerButton_Click(object sender, System.EventArgs e)
```

```
{
     int nStatus;

     // This function may take some time to execute. Change the curser to an hourglass
     // to let the user know it is processing.
     Cursor = Cursors.WaitCursor;

     try
     {
          if (mbPowerOn)
          {
               mMisc.SwitchOff(mCommunications, msSwitchPower, out nStatus);
          }
          else
          {
               mMisc.SwitchOn(mCommunications, msSwitchPower, out nStatus);
          }

          CheckError(nStatus);
     }
     catch (System.Runtime.InteropServices.COMException Ex)
     {
          ShowError(Ex);
     }
     catch (Exception Ex)
     {
          ShowError(Ex);
     }

     // Return to the normal cursor.
     Cursor = Cursors.Default;
}
```

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
ENABLE	ENABLE

Related Topics

SwitchOff

Where Method

Visual Basic 6

```
Status.Where(pCommunications as Communications, lRobotNumber as Long,
             pNumberOfJoints as Long, aWorldCoord as ArraySingle,
             aJointCoord as ArraySingle)
```

Visual Basic .NET

```
Status.Where(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                lRobotNumber As Integer, ByRef pNumberOfJoints As Integer,
                ByRef aWorldCoord As System.Array, ByRef aJointCoord As
                System.Array)
```

C# .NET

```
Status.Where(ActiveV2Lib.Communications pCommunications, System.Int32
                lRobotNumber, out System.Int32 pNumberOfJoints, out
                System.Array aWorldCoord, out System.Array aJointCoord)
```

Description

This method returns the robot position.

Input Parameters

lRobotNumber
Robot number. If the robot number is invalid, all output parameters will be set to zero.

Output Parameters

pNumberOfJoints
Number of joints.
aWorldCoord
World coordinates as seven numbers, representing x, y, z, y, p, r, hand
aJointCoord
Joints angles.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
WHERE	WHERE

Related Topics

Here

XStep Method

Visual Basic 6

```
Programs.XStep(pCommunications as Communications, lTaskNumber as Long,
               pStatus as Long)
```

Visual Basic .NET

```
Programs.XStep(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
               lTaskNumber As Integer, ByRef pStatus As Integer)
```

C# .NET

```
Programs.XStep(ActiveV2Lib.Communications pCommunications, System.Int32
               lTaskNumber, out System.Int32 pStatus)
```

Description

Executes the next step in a specified V+ task.

Usage Considerations

During execution, this method steps into a subroutine, as opposed to stepping across it. To step across subroutines, use the SStep method. To execute a specific line in a program, use XStep2.

Input Parameters

lTaskNumber	Task number

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the step executes successfully, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
XSTEP	XSTEP

Related Topics

XStep2

SStep

XStep2 Method

Visual Basic 6

```
Programs.XStep2(pCommunications as Communications, lTaskNumber as Long,
                lStepNumber as Long, pStatus as Long)
```

Visual Basic .NET

```
Programs.XStep2(ByVal pCommunications As ActiveV2Lib.Communications, ByVal
                lTaskNumber As Integer, ByVal lStepNumber As Integer, ByRef
                pStatus As Integer)
```

C# .NET

```
Programs.XStep2(ActiveV2Lib.Communications pCommunications, System.Int32
                lTaskNumber, System.Int32 lStepNumber, out System.Int32
                pStatus)
```

Description

Execute a specific program line in a V+ task.

Usage Considerations

If you want to step through an entire program, you can use the XStep method which steps through the program including subroutines. Or, you can use the SStep method which steps through the program, but across subroutines.

Input Parameters

lTaskNumber	Task number
lStepNumber	Step number, 0 for default

Output Parameters

pStatus	Returns 1 (VE_SUCCESS) if the step executes successfully, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword
XSTEP	XSTEP

Related Topics

XStep

SStep

ZERO Method

Visual Basic 6

```
Programs.ZERO(pCommunications as Communications, pStatus as Long)
```

Visual Basic .NET

```
Programs.ZERO(ByVal pCommunications As ActiveV2Lib.Communications, ByRef
              pStatus As Integer)
```

C# .NET

```
Programs.ZERO(ActiveV2Lib.Communications pCommunications, out System.Int32
              pStatus)
```

Description

This method clears V+ memory. All V+ programs and variables are removed.

Usage Considerations

This method cannot be used while a program task is executing.

Input Parameters

None.

Output Parameters

pStatus
Returns VE_SUCCESS if the operation is successful, or a V+ error code if the operation fails.

Corresponding V+ Keyword	Corresponding MicroV+ Keyword

ZERO	ZERO

Related Topics

Abort

Execute

Kill

Prime

Proceed

ZERO

Programming Example Overview

This documentation includes a programming example called "Cookie Demo" that is included with this documentation. Cookie Demo is an example of a multiple document interface (MDI). The example application is a simple pick−and−place operation using a V+ background monitoring task (ckd.monitor) and two runtime tasks: one that simulates a conveyor (ckd.conveyor) and one that controls the robot (ckd.robot).

Features and Operation

The example provides sample usage of the following ACTIVEV2.DLL library functionality:

- Reading/setting of switches/parameters
- Reading/setting global real and string variables
- Priming/Executing/Proceeding/Aborting/Killing V+ programs
- Reading Program status
- Reading Monitor status

The example includes the following core design features:

- A local database is used to configure the application (Visual Basic 6 version only)
- A software loader, which automatically loads all the required V+ files on startup
- A program scheduler, which is responsible for executing and aborting programs.
- A "watchdog" that monitors the status of programs that should continuously run (background tasks)
- Once the application has been started, V+ does not rely on the continuous operation of the Cookie Demo program.

Connect Dialog

When the Cookie Demo program is started, the Connect dialog opens and displays the default controller IP address. This is the address specified as IP_ADDRESS in tblSettings (Visual Basic 6 version); or msIPAddress (Visual Basic .NET and C# versions). The displayed IP address can be changed so that the program can connect to a different Adept controller.

Operator Interface Dialog

Once the application successfully connects to the Adept controller the Operator Interface dialog is displayed. This is the main dialog of the MDI and is divided into the following areas of functionality:

- Status: displays various status items that correspond to either the status of critical hardware parameters and switches, or software flags (e.g., Paused). This display is generated by reading a global status word from the V+ part of the cookie demo program (see ckd.monitor() in cookie.v2).

NOTE: Some of these flags have been defined but not implemented (such as "teach" and "home").

- Program Execution: Three buttons labeled Start, Pause/Continue and Stop. Start causes all runtime programs defined in tblPrograms to be started (executed). Pause/Continue forces the global ckd.req.pause flag to be set, which pauses all runtime programs (ckd.conveyor() and ckd.robot() in COOKIE.V2) until *Continue* is pressed (resetting the ckd.req.pause flag). Stop causes all runtime programs to be aborted.
- Robot Control: Two buttons allow power to be enabled/disabled and the robot to be calibrated. A slider bar sets the robot Monitor Speed.
- Program Output: This text area displays output from the V+ programs.

Cell IO Maintenance Dialog (Visual Basic 6 version)
Digital IO Panel (Visual Basic .NET and C# versions)

The Cell IO Maintenance or Digital IO Panel dialogs display a scrolling list of IO points (defined in tblIO in the Visual Basic 6 version only). Clicking on any IO point will select that IO point for status display and manual manipulation (toggle).

Jog Locations Dialog (Visual Basic 6 version)
Location Panel (Visual Basic .NET and C# versions)

The Jog Location or Location Panel dialogs display a pick list of locations (specified as EDIT_LOCATIONs in tblSettings in the Visual Basic 6 version only). A location can be selected from the list and then jogged in X, Y, and Z using the appropriate slide bars. The modified location is uploaded to the controller after each jog event (change in slider value). The location displayed is read back from the controller.

About Dialog

The About dialog displays information about the Cookie Demo program and the V+ system on the Adept controller (if connected).

Obtaining the Source Code

There are three versions of source code for the Cookie Demo program: Visual Basic 6, Visual Basic .NET, and C#. Each module header has associated documentation for the module. The source code for all three versions is bundled into one ZIP file, which can be obtained from the Adept Download Center. After accessing the Download Center, request Download ID: 7001.

Visual Basic 6

The Visual Basic 6 project file name is "CookieDemo.vbp", which can be opened and edited with MS Visual Basic 6.0 or later. Each module header has associated documentation for the module.

Visual Basic .NET

The Visual Basic .NET project file name is "CookieDemo.vbproj", which can be opened and edited with MS Visual Studio .NET 2003 or later. Each module header has associated documentation for the module.

C# .NET

The C# .NET project file name is "CookieDemo.csproj", which can be opened and edited with MS Visual Studio .NET 2003 or later. Each module header has associated documentation for the module.

Installation Instructions

NOTE: Before proceeding, be sure that your system (PC and Adept equipment) meets the requirements described in the System Requirements topic.

1. The CookieDemo.ZIP file contains the source code and installation files for the programming examples. Download the file as described in the section Obtaining the Source Code.
2. Unzip the CookieDemo.ZIP file to access the required components. The program and associated files will be installed by default into the following directories:

Programming Language	UnZIPed Folder Name	Project File
Visual Basic 6	\CookieDemo\VB6	CookieDemo.vbp
Visual Basic .NET	\CookieDemo\VB.NET	CookieDemo.vbproj
C# .NET	\CookieDemo\CSharp	CookieDemo.csproj
V+	\CookieDemo	COOKIE.LC COOKIE.V2

3. Create a directory named "DEMO" (no quotes) on your Adept controller and then copy the V+ files COOKIE.V2 and COOKIE.LC to that directory.
4. For the Visual Basic 6 version only, define the environment variable ADEPT_APP_DB to describe the full path and file name of the PC−side, MS−jet database file. For the default installation, this will be:

ADEPT_APP_DB = \CookieDemo\VB6\CookieDemo.mdb

Settings in COOKIEDEMO.MDB (Visual Basic 6 version only)

For the Visual Basic 6 version, the database CookieDemo.mdb is used to store the following information:

Defaults Settings (see tblSettings):

Field Name	Description
parameter	name of parameter (see below)
value	value for this parameter

Parameter Name	Description
IP_ADDRESS	Default IP address for connection
LOAD_FILE	V+ File(s) automatically loaded
EDIT_LOCATION	Name(s) of editable locations

Program–Task Scheduling (see tblPrograms):

Stores program name as Long with the associated task in which to run this program. The following behavioral flags are also used:

Field Name	Description
task	V+ task number
program	V+ program name
autostart	If TRUE, this task will be automatically started when the CookieDemo program starts
background	If TRUE, this task is expected to run continuously in the background

IO Tag List (see tblIO)

Stores information about cell IO that needs to be accessed from the PC−side.

Field Name	Description
TagName	Text tag name for this IO
IONumber	V+ IO number for this IO
Description	Description text for this IO
ManualAccess	If TRUE, the state of this IO point can be manually toggled using the IO panel

Developer's Notes

For most applications, you will want to store data on the PC−side. More...

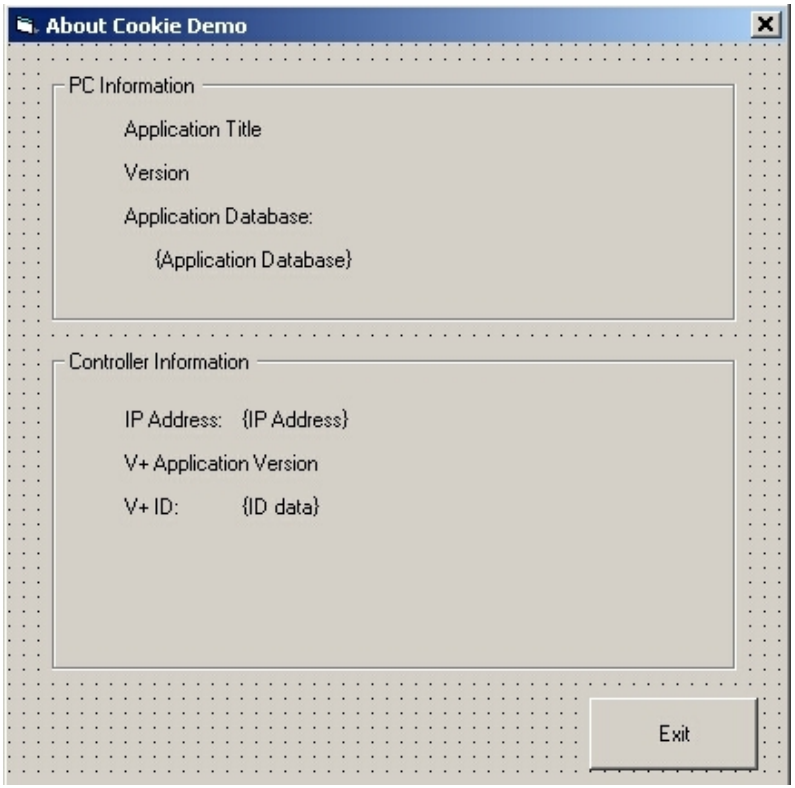
The Visual Basic 6 version of the Cookie Demo program uses the Microsoft Jet database engine and Data Access Objects (DAO). This is Microsoft's simple interface for databases. Note that the exact structure of your database (the "schema") will depend on your application.

Structured Query Language (SQL) is a standard language for querying databases. The Cookie Demo program uses some basic SQL to retrieve information from the DAO. See http://www.sql.org for more information.

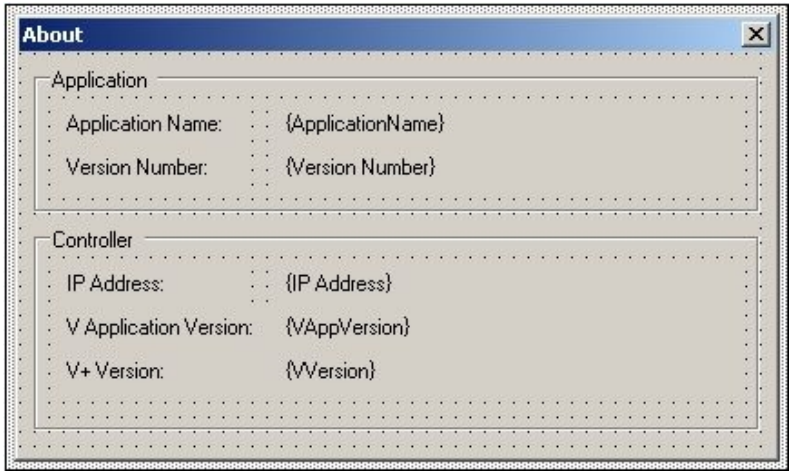
About Dialog

The files About.frm (VB6), About.vb (VB.NET) and About.cs (C#.NET) contain the code used by the About Cooke Demo dialog of the Cookie Demo application.

Visual Basic 6



Visual Basic .NET and C# .NET



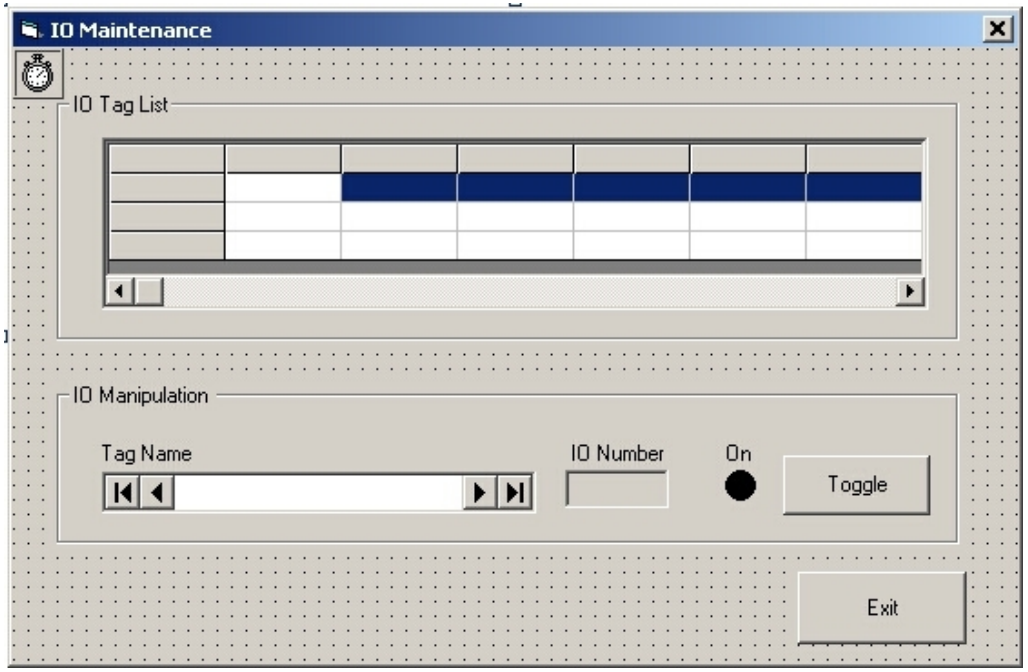
The following functionality is provided on this page:

- display of PC application information
- display of V+ system information from the Adept controller

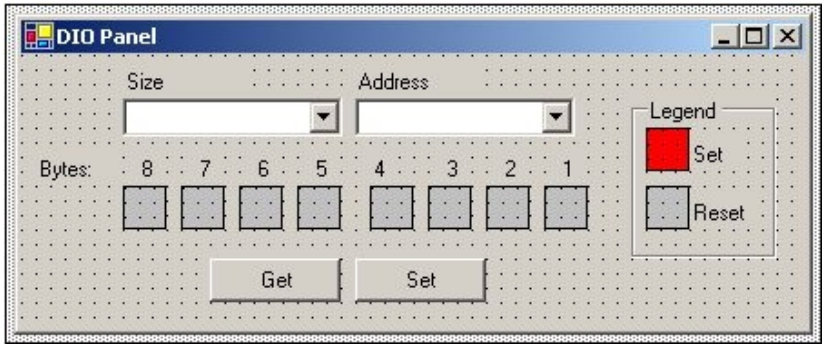
Cell IO Maintenance and DIO Panel Dialogs

The file CellIO.frm (VB6) contains the code used by the IO Maintenance page of the Cookie Demo application; the files DIOPanel.vb (VB.NET) and DIOPanel.cs (C#.NET) contain the code used by the DIO Panel page of the Cookie Demo application.

Visual Basic 6



Visual Basic .NET and C# .NET



The following functionality is provided on this page (VB6 version only):

1. Cell IO display: An MSFlexGrid control is used to display the raw contents of the tblIO database.
2. IO State monitoring: a Timer Control is used to periodically scan all IO and mirror IO data into the global arrays ioValid() and ioState(). In general this approach is recommended as it is more efficient to read IO in blocks than to perform a separate read for each IO point displayed. The Timer Control is also used to change the color of the ON/OFF IO state for the currently selected IO point.
3. IO State Manipulation: If *ManualAccess* is set to "yes" in tblIO then the state of the IO can be changed using the Toggle Button

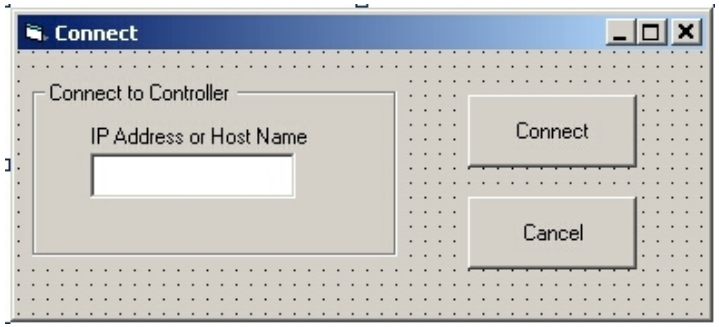
This programming example provides sample usage of the following ACTIVEV2lib functionality:

- Reading/setting of IO

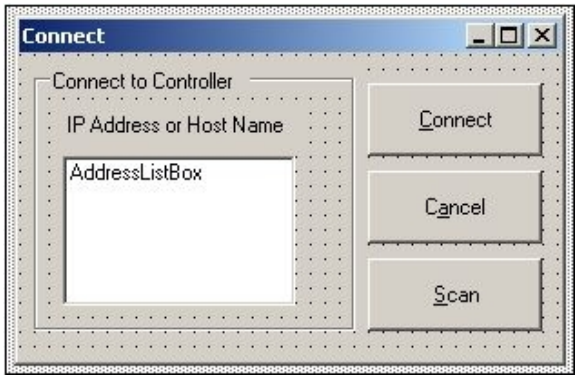
Connect Dialog

The files Connect.frm (VB6), Connect.vb (VB.NET) and Connect.cs (C#.NET) contain the code used by the Connect dialog of the Cookie Demo application. This form is "Modal" which means that the calling form should wait until this form is unloaded (exited) before proceeding.

Visual Basic 6



Visual Basic .NET and C# .NET



The following functionality is provided on this page:

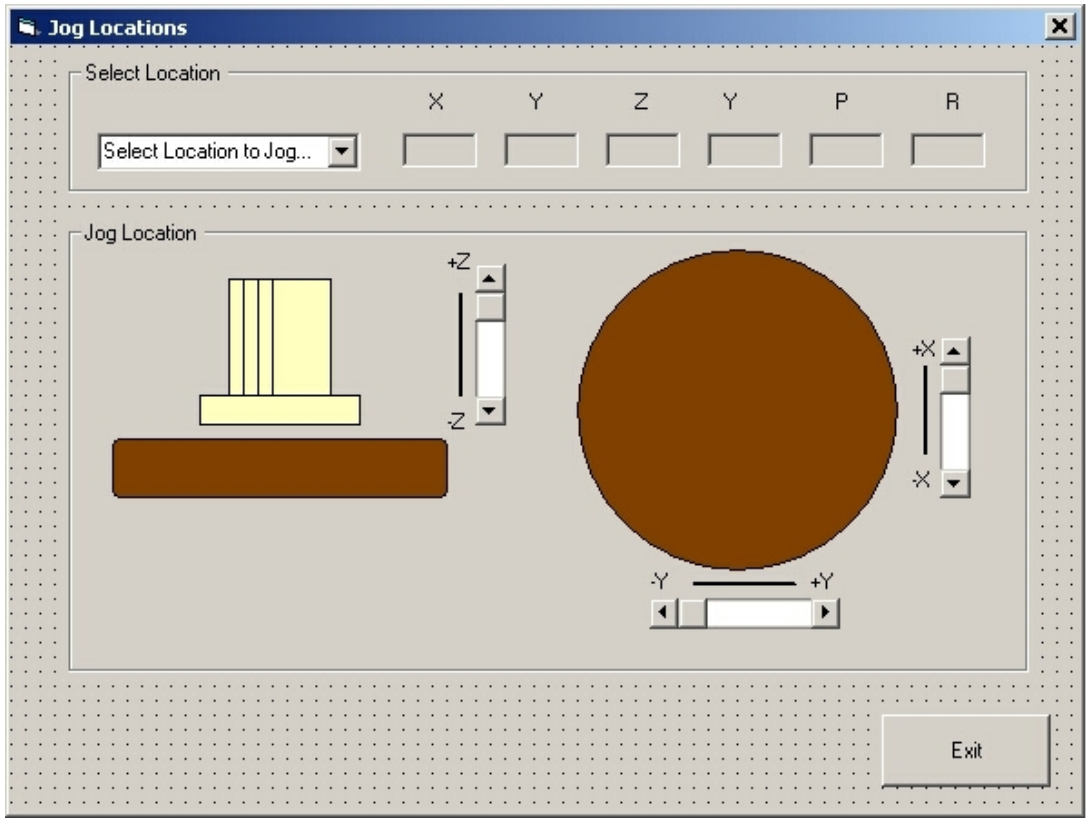
- Display of an IP address or host name
- Connect button – initiates the connection to the controller

- Cancel button – exits this form
- Scan button – scans for available IP address (VB.NET and C#.NET versions only)

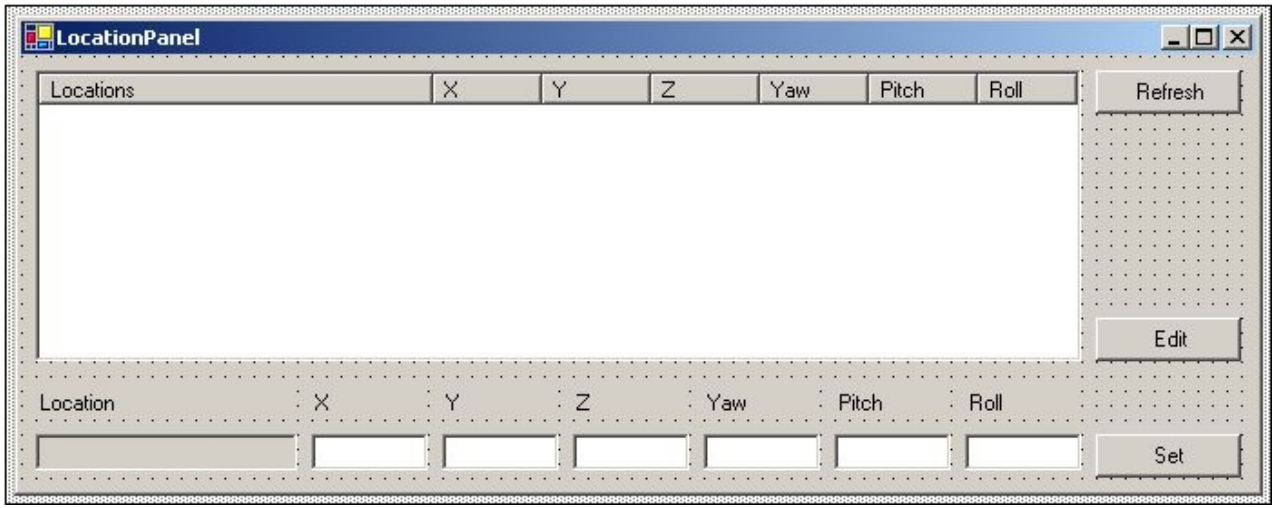
Jog Locations and Locations Panel Dialogs

The file JogPanel.frm (VB6) contains the code used by the Jog Panel of the Cookie Demo application; the files LocationPanel.vb (VB.NET) and LocationPanel.cs (C#.NET) contain the code used by the Location Panel of the Cookie Demo application.

Visual Basic 6



Visual Basic .NET and C# .NET



The following functionality is provided on this page (VB6 version only):

1. On Startup: When this page is loaded all editable location names specified in tblSettings (as parameter "EDIT_LOCATION") are loaded into a ComboBox Control for display and selection.
2. Location selection: A location is selected using the ComboBox. The global array currentLoc() is loaded with the current (X,Y,Z,Y,P,R) read from the controller for this location.
3. Jogging a Location: Horizontal and Vertical Slide Bar controls are used as jog controls. Each time a change is made a new (X,Y,Z,Y,P,R) is sent to the controller for this location and then read back to update the display. This ensures that the actual location value is always displayed.

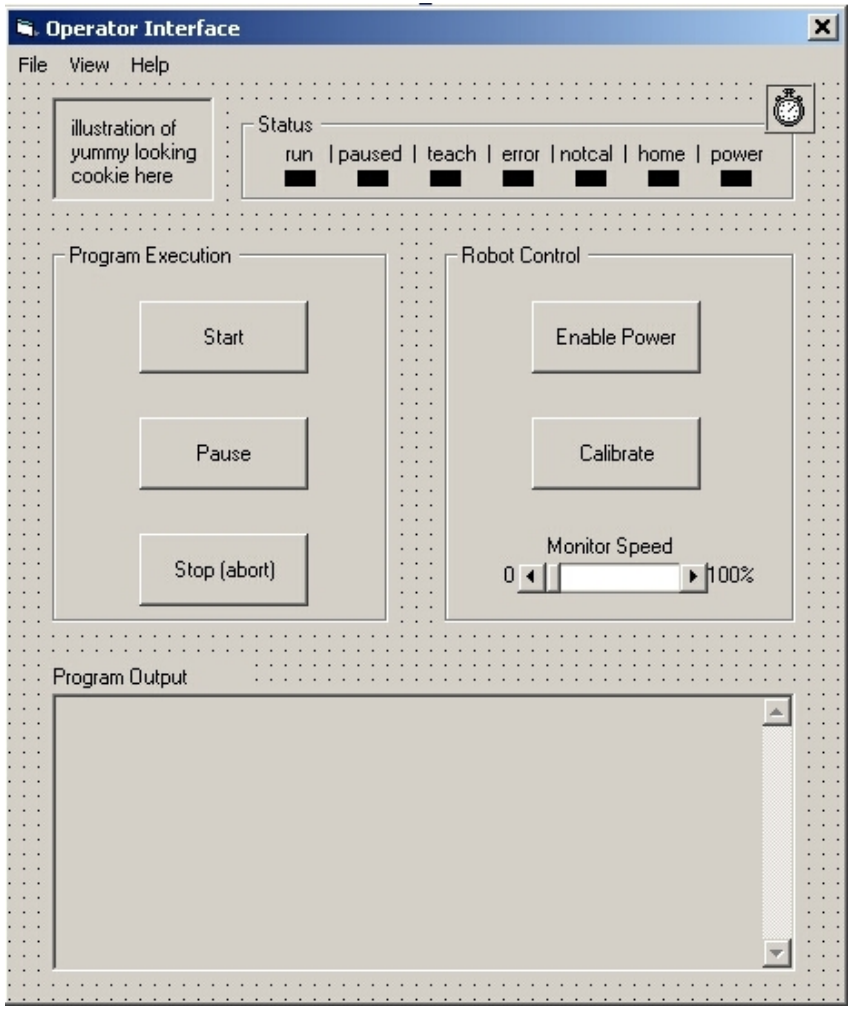
This programming example provides sample usage of the following ActiveV2lib functionality:

- Reading/setting of robot locations

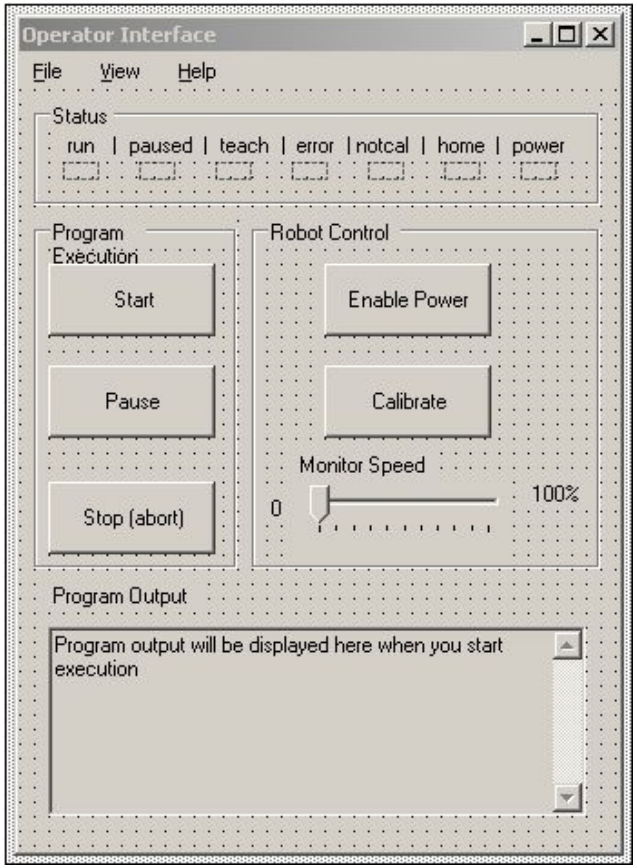
Operator Interface Dialog

The file OperatorPanel.frm (VB6), OperatorPanel.vb (VB.NET) and OperatorPanel.cs (C#.NET) contain the code used by the Operator Panel page of the Cookie Demo application.

Visual Basic 6



Visual Basic .NET and C# .NET



The following functionality is provided in this page:

1. When first started: Connects automatically to the V+ controller with the IP address specified by IP_ADDRESS in the connect panel.
2. Program Start/Stop: All programs that are not marked "background" are started (i.e., proceeded) when the Start Button is pressed, and stopped (i.e., aborted) when the Stop Button is pressed.
3. Program Pause: The Pause Button sets the V+ flag ckd.req.pause, which is used by the V+ software to determine when to pause (soft−stop).
4. Robot Power/Calibrate: Separate buttons enable/disable robot power and calibrate/home the robot
5. Status and background task monitoring: A Visual Basic timer control is used to monitor the status of the application by reading the variable ckd.status. Each bit of ckd.status has a different meaning (see Globals.bas) and is displayed as an array of indicators.