Minitab®19

Minitab Automation
Contents

1 Getting Started ......................................................................................................................................................... 4
   Introducing Automation in Minitab ............................................................................................................................ 4
   Strategies for Handling Errors in COM overview .................................................................................................. 4

2 Data Types ............................................................................................................................................................... 6
   Minitab Data Types ............................................................................................................................................... 6

3 Data Model ............................................................................................................................................................... 8
   Minitab Automation objects ................................................................................................................................. 8
   Minitab Command Automation Objects ............................................................................................................... 9

4 My Menu ................................................................................................................................................................. 11
   My Menu Overview ........................................................................................................................................... 11

A Minitab Automation Object Reference .................................................................................................................. 22
   Application object ............................................................................................................................................... 22
   ApplicationOptions object ................................................................................................................................. 26
   UserInterface object ......................................................................................................................................... 29
   Project object .................................................................................................................................................... 31

B Worksheet Object Reference .................................................................................................................................. 40
   Worksheets Collection object ............................................................................................................................ 40
   Worksheet object ............................................................................................................................................... 44
   Columns Collection object ................................................................................................................................ 50
   Column object .................................................................................................................................................... 55
   Constants Collection object ................................................................................................................................ 64
   Constant object ................................................................................................................................................... 68
   Matrices Collection object .................................................................................................................................. 73
   Matrix object ....................................................................................................................................................... 77

C Command Object Reference .................................................................................................................................. 83
   Commands Collection object .................................................................................................................................. 83
   Command object .................................................................................................................................................. 86
   Outputs Collection object ...................................................................................................................................... 90
   Output object ..................................................................................................................................................... 92
   OutputDocument object ....................................................................................................................................... 97
   Graph object ...................................................................................................................................................... 99
Introducing Automation in Minitab

The COM automation library contains a set of standard COM (Component Object Model) objects that expose much of Minitab’s internal functionality. You can use this COM library with any COM-compliant language.

Strategies for Handling Errors in COM overview

HRESULT values

COM returns an HRESULT value for all methods in all component interfaces. An HRESULT indicates whether a COM method succeeded or failed. HRESULTs also report any errors in making function calls or interface method calls and identify the facilities associated with the errors, such as RPC, WIN32, or ITF for interface-specific errors. Lastly, system APIs provide a lookup from an HRESULT to a string that describes the error condition.

Using methods that return HRESULTs is fundamental to well-written components and is essential to the debugging process. Microsoft Visual Basic automatically defines each method with an HRESULT as a return. In Microsoft Visual C++, you must explicitly return an HRESULT.

ErrorInfo objects

ErrorInfo objects are often called COM exceptions because they allow an object to pass (or throw) rich error information to its caller, even across apartment boundaries. The value of this generic error object is that it supplements an HRESULT, extending the type of error description, the source of the error, and the interface identifier of the method that originated the error. You can also include pointers to an entry in a Help file.

Automation provides three interfaces to manage the error object:

- Components must implement the ISupportErrorInfo interface to advertise their support for the ErrorInfo object.
- When an error occurs, the component uses the ICreateErrorInfo interface to initialize an error object.
- After the caller inspects the HRESULT and finds that the method call failed, it queries the object to see whether it supports the ErrorInfo object. If it does, the caller uses the IErrorInfo interface to retrieve the error information.

Visual Basic programmers have easy access to the ErrorInfo object, which is exposed through the Err object. You can raise errors with the Err.Raise function and catch errors with the On Error statement. The Visual Basic run-time layer takes care of the mapping for you. If you are using the Visual C++ COM compiler support, you can use the _com_raise_error class to report an error, and the _com_error class to retrieve error information. COM will not propagate traditional C++ exceptions as extended IErrorInfo information.

HRESULT Definitions

The return value of COM functions and methods is an HRESULT. The following table lists the standard HRESULT definitions. To use the return values, you must include winerror.h in your project.
<table>
<thead>
<tr>
<th>HRESULT</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_NOINTERFACE</td>
<td>The <em>QueryInterface</em> function did not recognize the requested interface. The interface is not supported.</td>
</tr>
<tr>
<td>E_NOTIMPL</td>
<td>The function contains no implementation.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>An unspecified failure has occurred.</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>The function failed to allocate necessary memory.</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>Invalid pointer.</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>One or more arguments are invalid.</td>
</tr>
<tr>
<td>E_UNEXPECTED</td>
<td>A catastrophic failure has occurred.</td>
</tr>
<tr>
<td>E_HANDLE</td>
<td>Invalid handle.</td>
</tr>
<tr>
<td>E_ABORT</td>
<td>Operation aborted.</td>
</tr>
</tbody>
</table>

**Note**  The information in this section is from the MSDN Library - January 2001, platform SDK:COM (Component Services).
2 Data Types

Minitab Data Types

MtbAppStatusTypes
Defines the different Mtb Application status types.
0 = ASReady (Minitab is ready to accept commands)
1 = ASBusy (Minitab is busy executing a command)
2 = ASError (The last command executed caused an error)
3 = ASQuit (Quit has been called but the application object is not yet destroyed)

MtbDataTypes
Defines the different data types that are currently supported.
0 = Text
1 = Numeric
2 = DateTime
3 = DataUnassigned

MtbFormulaStatusTypes
Defines the state of the Formula for a Column or Constant object.
0 = FSNone
1 = FSUpToDate
2 = FSOOutOfDate
3 = FSInvalid

MtbGraphFileTypes
Defines the different graph file types.
1 = GFJPEG
2 = GFPNGGrayscale
3 = GFPNGColor
3 = GFPNGHighColor
5 = GFTIFGrayscale
6 = GFTIFColor
7 = GFBMPGrayscale
8 = GFBMPColor
8 = GFBMPHighColor
10 = GFGIF
11 = GFEMF

MtbOutputFileTypes
Defines the different Output File types.
0 = OFPlainText
1 = OFHTML
2 = OFRTF
100 = OFDefault

MtbOutputTypes
Defines the different output types allowed in an Output object.
0 = OTGraph
1 = OTTable
3 = OTTitle
4 = OTMessage
6 = OTFormula

MtbValueOrderTypes
Defines the display ordering associated with a column.
0 = Alphabetical
1 = WorksheetOrder
2 = UserDefined
3 Data Model

Minitab Automation objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application</strong> on page 22</td>
<td>The Application object serves as the root node in the Minitab automation server object hierarchy. The Application object is the only object in the hierarchy that can be created by the client. All the lower objects in the hierarchy are accessed through the Application object or through objects contained within the Application object. The Application object provides an interface to allow the client to set and get application-wide global properties.</td>
</tr>
<tr>
<td><strong>ApplicationOptions</strong> on page 26</td>
<td>Use the ApplicationOptions object to read or set options that pertain to the Application object. Use the Options property of the Application object to access ApplicationOptions. For more information on the Application object, go to Application object on page 22.</td>
</tr>
<tr>
<td><strong>UserInterface</strong> on page 29</td>
<td>The UserInterface object allows control of the Minitab host.</td>
</tr>
<tr>
<td><strong>Project</strong> on page 31</td>
<td>The Project object contains all the information related to an individual project, including the Worksheets collection and the Commands collection.</td>
</tr>
<tr>
<td><strong>Commands collection</strong> on page 83</td>
<td>The Commands collection contains the commands that have been issued to Minitab during the session. See Minitab Command Automation Objects on page 9 for the data model.</td>
</tr>
<tr>
<td><strong>Worksheets collection</strong> on page 40</td>
<td>The Worksheets collection is a set of all the Worksheet objects within a Project object. It supports the standard collection properties and methods.</td>
</tr>
</tbody>
</table>
The Worksheet object contains all the information related to an individual worksheet, including the Columns, Constants, and Matrices collections, which provide access to all the columns, constants, and matrices in the worksheet.

The Columns collection is a set of all the Column objects within a Worksheet object. It supports the standard collection properties and methods.

The Column object contains all the information related to an individual column. The data type on page 6 for each Column object can be Text, Numeric, DateTime, or DataUnassigned.

The Constants collection is a set of all the Constant objects within a Worksheet object. It supports the standard collection properties and methods.

The Constant object contains all the information related to an individual constant. The Constant object can contain numeric or text values.

The Matrices collection is a set of all the Matrix objects within a Worksheet object. It supports the standard collection properties and methods.

The Matrix object contains all the information related to an individual matrix. The Matrix object can contain only numeric data values.

### Minitab Command Automation Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commands collection</td>
<td>The Commands collection contains the commands that have been issued to Minitab during the session.</td>
</tr>
<tr>
<td>OutputDocument</td>
<td>An OutputDocument object contains all output generated by a single Command object or by all commands in the Commands collection.</td>
</tr>
<tr>
<td>Command</td>
<td>Command objects are created when you execute a Minitab command either programmatically or directly in Minitab.</td>
</tr>
</tbody>
</table>
### Object Description

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outputs collection on page 90</td>
<td>The Outputs collection for each Command object contains all the output generated by that command.</td>
</tr>
<tr>
<td>Output on page 92</td>
<td>Each Output object contains one component of the output from a Minitab Command object.</td>
</tr>
<tr>
<td>Graph on page 99</td>
<td>Each Graph object contains a single graph generated by a Minitab Command object.</td>
</tr>
</tbody>
</table>
4 My Menu

My Menu Overview

By creating specialized dynamic link libraries (DLLs) and placing them in the AddIns folder of your Minitab directory, you can add customized menus to Minitab that allow you to do the following:

• Run a Minitab macro from the menu.
• Display a customized dialog box for running a Minitab macro.
• Launch customized interfaces to corporate databases, or company-created macros.
• Create custom Minitab procedures using Minitab’s new COM objects.
• Launch a separate executable from Minitab.

Your custom menus will appear at the right end of the menu bar. Minitab’s new customizable menus and toolbars allow you to move any of the items from your custom menus to any menu or toolbar you would like.

My Menu Addin DLLs

To specify the layout, items, and actions for a custom menu, you need to create a DLL and place it in the AddIns folder of your Minitab directory.

Each Addin DLL must include the following 9 methods:

**public string GetName on page 13()**
This method returns the friendly name of your Addin. Both the name and the description of the Addin are stored in the registry.

**public string GetDescription on page 13()**
This method returns the description of your Addin.

**public void GetMenuItems on page 13(ref string sMainMenu, ref Array saMenuItems, ref int iFlags)**
This method returns the text for the main menu and each menu item. You can return ”|” to create a menu separator in your menu items. You must instantiate the menu items array to fit your number of items.

**public void OnConnect on page 13(int iHwnd, object pApp, ref int iFlags)**
This method is called as Minitab is initializing your Addin. The “iHwnd” parameter is the handle to the main Minitab window. The “pApp” parameter is a reference to the “Minitab Automation object.” You can hold onto either of these for use in your Addin. “iFlags” is used to tell Minitab if your Addin has dynamic menus (i.e., should be reloaded each time Minitab starts up). Set iFlags to 1 for dynamic menus and 0 for static.

**public void OnDisconnect on page 13()**
This method is called as Minitab is closing your Addin.

**public string OnDispatchCommand on page 13(int iMenu)**
This method is called whenever a user selects one of your menu items. The “iMenu” variable should be equivalent to the menu item index set in “GetMenuItems”.

**public void OnNotify on page 13(AddinNotifyType eAddinNotifyType)**
This method is called when Minitab notifies your Addin that something has changed. Use the "eAddinNotifyType" to figure out what changed.
public bool QueryCustomCommand on page 13(string sCommand)
This method is called when Minitab asks your Addin if it supports a custom command. The argument "sCommand" is the name of the custom command. Return "true" if you support the command.

public void ExecuteCustomCommand on page 13(string sCommand, ref Array saArgs)
This method is called when Minitab asks your Addin to execute a custom command. The argument "sCommand" is the name of the command, and "saArgs" is an array of arguments.

Create a Custom Menu

To create a custom menu, refer to the following high-level steps.
1. Create a C# Class Library on page 12.
2. Add COM References on page 12.

Before you start

- Refer to the example DLL, along with all supporting files for the C# project used to create it, which are located in the MyMenu folder of your Minitab directory.
- Refer to portions of the code from the main module of this project, which are displayed in My Menu - C# Example on page 13.
- Be sure that the add-in implements the add-in interface.
- Add the Minitab Addin Interface to the project references.
- If you want to use the Minitab objects, add a reference to the Minitab Type Library.
- Add the following four attributes to your class:
  - [ComVisible(true)]
  - [Guid("40B9FD8-21FB-4E92-AD9B-7DE6358B675B")]
  - [ClassInterface(ClassInterfaceType.None)]
  - [ProgId("MyMenu.AddIn")]

Note When creating multiple menu items, be sure to create a unique GUID for each item.

- Add this attribute to your class:
  - [DllExport("DllRegisterServer", CallingConvention.StdCall)]

Create a C# Class Library
1. Open Microsoft Visual Studio.
2. Choose File > New > Project.
3. Create C# class library, then click OK.
4. From the Solutions Explorer, right-click the solution and choose Configuration Manager.
5. In the Configuration Manager dialog box, under Active solution platform, choose <New...>.
6. In the New Solution Platform dialog box, choose x86 as the new platform, click OK, and then Close.

Add COM References
1. In Microsoft Visual Studio, choose Tools > NuGet Package Manager > Package Manager Console.
2. At the Package Manager prompt, type `Install-Package UnmanagedExports`, then press Enter. (For the 64-bit version of Visual Studio, type `Install-Package UnmanagedExports-Version 1.2.3-Beta`.) Under References, you now have a reference to `RGiesecke.DllExport`.

3. Under the project, right-click References and choose Add Reference.

4. In the Reference Manager, click COM, select the following libraries, then click OK.
   - Minitab Addin Interface
   - Mtb Type Library
   Under References, you now have references to `MinitabAddinTLB` and `Mtb`.

Build and Test Your Solution

1. In Visual Studio, open the Class1.cs file in your solution and add the following directive:
   ```csharp
   using RGiesecke.DllExport;
   ```

2. Choose Build > Build Solution.

3. Create a compiled resource file.
   a. Open a Visual Studio Developer Command prompt as an administrator.
   b. At the prompt, change the directory to the object (OBJ) folder where the MyMenu.DLL file is located.
   c. Type the following commands and press Enter:
      ```
      TlbExp.exe /win32 MyMenu.dll
      echo 1 typelib MyMenu.tlb > MyMenu.rc
      rc.exe MyMenu.rc
      ```

4. In Visual Studio, point to the resource file you created in the previous step.
   a. Right-click the project in your solution and choose Properties.
   c. Click the browse button.
   d. Browse to the compiled resource (.res) file and click OK.

5. Choose Build > Rebuild Solution.

6. Test the solution.
   a. Copy the MyMenu.DLL file from the binary (BIN) folder to the English\Addins folder of your Minitab installation directory, for example, C:\Program Files (x86)\Minitab\Minitab 19.
   b. Run Minitab as an administrator.
   c. In Minitab, choose View > Customize.
   d. On the Menu tab, under Application Frame Menus, click Reset.

My Menu - C# Example

```csharp
using System;
using System.Collections;
```
using System.Diagnostics;
using System.Reflection;
using System.Runtime.InteropServices;
using System.Text;
using System.Windows.Forms;
using Microsoft.Win32;
using MinitabAddinTLB;
using Mtb;
using Application = Mtb.Application;

namespace MyMenu
{
    [ComVisible(true)]
    [Guid("40B99FD8-21FB-4E92-AD9B-7DE6358B675B")]
    [ClassInterface(ClassInterfaceType.None)]
    [ProgId("MyMenu.AddIn")]
    public class AddIn : IMinitabAddin
    {
        internal static Application gMtbApp;

        [DllExport("DllRegisterServer", CallingConvention.StdCall)]
        public static int DllRegisterServer()
        {
            try
            {
                SetUpCLSID(Registry.ClassesRoot);
                SetUpCLSID(Registry.LocalMachine.OpenSubKey("SOFTWARE", true).OpenSubKey("SOFTWARE\Classes", true));
            }
            catch (Exception)
            {
                // Probably didn't have permissions to modify the registry
            }

            return 0;
        }

        private static void SetUpCLSID(RegistryKey root)
        {
            Type type = typeof(AddIn);
            string guid = type.GUID.ToString("B");
            string runtimeVersion = Environment.Version.ToString();
            string codeBase = Assembly.GetExecutingAssembly().CodeBase;

            RegistryKey typeRoot = root.CreateSubKey(type.FullName);
            typeRoot.SetValue("", type.FullName);
            typeRoot.CreateSubKey("CLSID").SetValue("", guid);

            RegistryKey clsidGuid = root.OpenSubKey("CLSID", true).CreateSubKey(guid);
            clsidGuid.SetValue("", type.FullName);

            clsidGuid.CreateSubKey("Implemented
Categories").CreateSubKey("{62C8FE65-4EBB-45e7-B440-6E39B2CDBF29}");
            RegistryKey server = clsidGuid.CreateSubKey("InprocServer32");
            server.SetValue("", "mscoree.dll");
            server.SetValue("ThreadingModel", "Both");
            server.SetValue("Class", type.FullName);
            server.SetValue("Assembly", type.Assembly.FullName);
            server.SetValue("RuntimeVersion", runtimeVersion);
            server.SetValue("CodeBase", codeBase);
        }
    }
}
RegistryKey serverVersion = server.CreateSubKey("1.0.0.0");
serverVersion.SetValue("Class", type.FullName);
serverVersion.SetValue("Assembly", type.Assembly.FullName);
serverVersion.SetValue("RuntimeVersion", runtimeVersion);
serverVersion.SetValue("CodeBase", codeBase);

clsidGuid.CreateSubKey("ProdId").SetValue("", type.FullName);
}

public void OnConnect(IntPtr iHwnd, object pApp, ref int iFlags)
{
    // This method is called as Minitab is initializing your add-in.
    // The "iHwnd" parameter is the handle to the main Minitab window.
    // The "pApp" parameter is a reference to the "Minitab Automation object."
    // You can hold onto either of these for use in your add-in.
    // "iFlags" is used to tell Minitab if your add-in has dynamic menus
    // (i.e. should be reloaded each time
    // Minitab starts up). Set iFlags to 1 for dynamic menus and 0 for static.
    gMtbApp = pApp as Application;
    // Static menus:
    iFlags = 0;
    return;
}

public void OnDisconnect()
{
    // This method is called as Minitab is closing your add-in.
    GC.Collect();
    GC.WaitForPendingFinalizers();
    GC.Collect();
    GC.WaitForPendingFinalizers();
    try
    {
        Marshal.ReleaseComObject(gMtbApp);
        gMtbApp = null;
    }
    catch
    {
    }
    return;
}

public string GetName()
{
    // This method returns the friendly name of your add-in:
    // Both the name and the description of the add-in are stored in the registry.
    return "Example C# Minitab Add-In";
}

public string GetDescription()
{
    // This method returns the description of your add-in:
    return "An example Minitab add-in written in C# using the "My Menu"
    functionality.";
}

public void GetMenuItems(ref string sMainMenu, ref Array saMenuItems, ref int iFlags)
{
    // This method returns the text for the main menu and each menu item.
    // You can return "|
    return;
}
sMainMenu = "&My Menu";  // This string is the name of the menu.
saMenuItems = new string[5];  // The strings in this array are the names of the
// items on the aforementioned menu.
    saMenuItems.SetValue("Describe &column(s)…", 0);
saMenuItems.SetValue("Rename active &worksheet…", 1);
saMenuItems.SetValue("|", 2);
saMenuItems.SetValue("&DOS window", 3);
saMenuItems.SetValue("&Geometric Mean and Mean Absolute Difference…", 4);

    // Flags is not currently used:
iFlags = 0;
    return;
}

public string OnDispatchCommand(int iMenu)
{
    // This method is called whenever a user selects one of your menu items.
    // The iMenu variable should be equivalent to the menu item index set in
    // "GetMenuItems."
    string command = string.Empty;
    DialogResult dialogResult = new DialogResult();
    switch (iMenu)
    {
    case 0:
        // Describe column(s):
        FormDescribe formDescribe = new FormDescribe(ref gMtbApp);
        // Fill up list box in dialog with numeric columns in worksheet:
        formDescribe.checkedListBoxOfColumns.ClearSelected();
        int lColumnCount = gMtbApp.ActiveProject.ActiveWorksheet.Columns.Count;
        for (int i = 1; i <= lColumnCount; i += 1)
        {
            // Select only the numeric columns:
            if (gMtbApp.ActiveProject.ActiveWorksheet.Columns.Item(i).DataType == MtbDataTypes.Numeric)
            {
                formDescribe.checkedListBoxOfColumns.Items.Add(gMtbApp.ActiveProject.ActiveWorksheet.Columns.Item(i).SynthesizedName);
            }
        }
        // Show the dialog:
        dialogResult = formDescribe.ShowDialog();
        if (dialogResult == DialogResult.OK)
        {
            StringBuilder cmnd = new StringBuilder("Describe ");
            bool bPrev = false;
            for (int i = 0; i < formDescribe.checkedListBoxOfColumns.CheckedItems.Count; i += 1)
            {
                if (bPrev)
                {
                    cmnd.Append(" ");
                }
                cmnd.Append(formDescribe.checkedListBoxOfColumns.CheckedItems[i].ToString());
            }
        }
        break;
    case 1:
        // Rename active worksheet:
        break;
    case 2:
        // Select active worksheet:
        break;
    case 3:
        // Show DOS window:
        break;
    case 4:
        // Show geometric mean and mean absolute difference:
        break;
    }
    return;
}
bPrev = true;

if (formDescribe.chkMean.Checked)
{
    cmnd.Append("; Mean");
}
if (formDescribe.chkVariance.Checked)
{
    cmnd.Append("; Variance");
}
if (formDescribe.chkSum.Checked)
{
    cmnd.Append("; Sums");
}
if (formDescribe.chkNnonmissing.Checked)
{
    cmnd.Append("; N");
}
if (formDescribe.chkHistogram.Checked)
{
    cmnd.Append("; GHist");
}
if (formDescribe.chkBoxplot.Checked)
{
    cmnd.Append("; GBoxplot");
}

command = cmnd.ToString();
formDescribe.Close();
break;

// Rename active worksheet:
FormRename formRename = new FormRename(ref gMtbApp);
string sCurrent = gMtbApp.ActiveProject.ActiveWorksheet.Name;
formRename.textBoxCurrent.Enabled = true;
formRename.textBoxCurrent.Text = sCurrent;
formRename.textBoxCurrent.Enabled = false;

// Show the dialog:
dialogResult = formRename.ShowDialog();
if (dialogResult == DialogResult.OK)
{
    gMtbApp.ActiveProject.ActiveWorksheet.Name = formRename.textBoxNew.Text;
}
formRename.Close();
break;

// Open a DOS Window:
string[] fileNamePossibilities = { "cmd.exe", "command.com" };
Process process;
ProcessStartInfo processStartInfo;
foreach (string fileNamePossibility in fileNamePossibilities)
{
    process = new Process();
    processStartInfo = new ProcessStartInfo();
    processStartInfo.UseShellExecute = true;
    processStartInfo.FileName = fileNamePossibility;
    process.StartInfo = processStartInfo;
}
try {
    process.Start();
    break;
} catch (Exception e) {
    MessageBox.Show(e.Message, "My Menu");
    MessageBox.Show("Cannot locate DOS executable or otherwise start
a command prompt...", "My Menu");
    continue;
}
break;
case 4:
    // "Geometric Mean" and "Mean Absolute Difference" (stored in the
worksheet):
    FormGeoMean formGeoMean = new FormGeoMean(ref gMtbApp);
    // Fill up list box in dialog with numeric columns in worksheet:
    lColumnCount = gMtbApp.ActiveProject.ActiveWorksheet.Columns.Count;
    Hashtable hashtableOfNumericColumns = new Hashtable();
    for (int i = 1; i <= lColumnCount; i += 1) {
        if (gMtbApp.ActiveProject.ActiveWorksheet.Columns.Item(i).DataType
== MtbDataTypes.Numeric) {
            string sSynthesizedColumnName =
gMtbApp.ActiveProject.ActiveWorksheet.Columns.Item(i).SynthesizedName;
            string sColumnName =
gMtbApp.ActiveProject.ActiveWorksheet.Columns.Item(i).Name;
            // Add column name (if it exists):
            if (sColumnName != sSynthesizedColumnName) {
                sSynthesizedColumnName += string.Concat(" ", sColumnName);
            }
            formGeoMean.comboBox.Items.Add(sSynthesizedColumnName);
            hashtableOfNumericColumns.Add(sSynthesizedColumnName,
gMtbApp.ActiveProject.ActiveWorksheet.Columns.Item(i));
        }
    }
    // Show the dialog:
    dialogResult = formGeoMean.ShowDialog();
    if (dialogResult == DialogResult.OK) {
        // Get data from the column and pass it to the function to do
        calculations:
        object selectedItem = formGeoMean.comboBox.SelectedItem;
        Column mtbDataColumn =
        (Column)hashtableOfNumericColumns[selectedItem];
        // "FindGeoMean" takes an array of doubles and returns the geometric
        mean.
        // "bSuccess" indicates if the calculations were completed.
        Array daData = (Array)mtbDataColumn.GetData();
        bool bSuccess;
        object dGeoMean = FindGeoMean(ref daData, out bSuccess);
        if (bSuccess) {
            // Find the "Mean Absolute Difference":
            object dMAD = FindMAD(ref daData);
            // Store both values in the first available column:
Column mtbStorageColumn =
gMtbApp.ActiveProject.ActiveWorksheet.Columns.Add();
    mtbStorageColumn.SetData(ref dGeoMean, 1, 1);
    mtbStorageColumn.SetData(ref dMAD, 2, 1);
    mtbStorageColumn.Name = "MyResults";
}
else
{
    // An error occurred:
    gMtbApp.ActiveProject.ExecuteCommand("NOTE ** Error ** Cannot compute statistics...");
    formGeoMean.Close();
}
break;
default:
    break;
}
return command;
}

public void OnNotify(AddinNotifyType eAddinNotifyType)
{
    // This method is called when Minitab notifies your add-in
    // that something has changed.
    // Use the "eAddinNotifyType" parameter to figure out what changed.
    // Minitab currently fires no events, so this method is not called.
    return;
}

public bool QueryCustomCommand(string sCommand)
{
    // This method is called when Minitab asks your Addin if it supports a custom
    // command.
    // The argument "sCommand" is the name of the custom command.
    // Return "true" if you support the command.
    return sCommand.ToUpper() == "EXPLORER" || sCommand.ToUpper() == "CLEAR";
}

public void ExecuteCustomCommand(string sCommand, ref Array saArgs)
{
    // This method is called when Minitab asks your add-in to execute a custom
    // command.
    // The argument "sCommand" is the name of the command, and "saArgs" is an array
    // of arguments.
    if (sCommand.ToUpper() == "EXPLORER")
    {
        // Open Windows Explorer:
        Process process = new Process();
        ProcessStartInfo processStartInfo = new ProcessStartInfo();
        processStartInfo.UseShellExecute = true;
        processStartInfo.FileName = "explorer.exe";
        process.StartInfo = processStartInfo;
        try
        {
            process.Start();
        }
        catch (Exception e)
        {
            MessageBox.Show(e.Message, "My Menu");
            MessageBox.Show("Apparently, Windows Explorer could not be started...",
                "My Menu");
        }
    }
}
else if (sCommand.ToUpper() == "CLEAR")
{
    // Clear indicated columns:
    int lColumnCount = gMtbApp.ActiveProject.ActiveWorksheet.Columns.Count;
    int saArgsCardinality = saArgs.GetLength(saArgs.Rank - 1);
    IEnumerator myEnumerator = saArgs.GetEnumerator();
    while (myEnumerator.MoveNext())
    {
        for (int i = 1; i <= lColumnCount; i++)
        {
            int myEnumeratorCurrent = 0;
            int.TryParse(myEnumerator.Current.ToString(), out myEnumeratorCurrent);
            if (gMtbApp.ActiveProject.ActiveWorksheet.Columns.Item(i).Number == myEnumeratorCurrent)
            {
                gMtbApp.ActiveProject.ActiveWorksheet.Columns.Item(i).Clear();
            }
        }
    }
}

public double FindGeoMean(ref Array saData, out bool bSuccess)
{
    // Find geometric mean:
    double dSum = 0.0;
    int iCount = 0;
    bSuccess = true;
    foreach (double dValue in saData)
    {
        if (dValue <= 0)
        {
            bSuccess = false;
            MessageBox.Show("All values must be strictly positive!", "My Menu");
            break;
        }
        dSum += Math.Log(dValue);
        iCount += 1;
    }
    return Math.Exp(dSum / iCount);
}

public double FindMAD(ref Array daData)
{
    // Find M(ean) Absolute D(ifference):
    double dSum = 0.0;
    int iCount = 0;
    foreach (double dValue in daData)
    {
        dSum += dValue;
        iCount += 1;
    }
    double dMAD = 0.0;
    double dMean = dSum / iCount;
foreach (double dValue in daData)
{
    dMAD += Math.Abs(dValue - dMean);
}

dMAD /= iCount;
return dMAD;
}
A Minitab Automation Object Reference

Application object

The Application object serves as the root node in the Minitab automation server object hierarchy. The Application object is the only object in the hierarchy that can be created by the client. All the lower objects in the hierarchy are accessed through the Application object or through objects contained within the Application object. The Application object provides an interface to allow the client to set and get application-wide global properties.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveProject</td>
<td>Returns the currently active project.</td>
</tr>
<tr>
<td>AppPath</td>
<td>The path to the currently running Minitab executable.</td>
</tr>
<tr>
<td>Handle</td>
<td>The handle to the main Minitab window.</td>
</tr>
<tr>
<td>LastError</td>
<td>Holds the last error that occurred during execution of a command.</td>
</tr>
<tr>
<td>Options</td>
<td>Returns the ApplicationOptions object.</td>
</tr>
<tr>
<td>Status</td>
<td>Indicates the current status of the Minitab application.</td>
</tr>
<tr>
<td>UserInterface</td>
<td>Returns the UserInterface object of the application.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Help</td>
<td>Use to launch this Help file.</td>
</tr>
<tr>
<td>New</td>
<td>Use to create a new project and make it the active project.</td>
</tr>
<tr>
<td>Open</td>
<td>Use to open an existing project file and make it the active project.</td>
</tr>
<tr>
<td>Quit</td>
<td>Use to close and delete all objects in the Application object hierarchy, including the Application object.</td>
</tr>
</tbody>
</table>

Example

Create a Minitab Application object (mtbApp) and make it visible to the user. Then display a message box with the values of the Status, LastError, AppPath, and Handle properties, as well as the DefaultFilePath property of the ApplicationOptions object. Finally, change the comment for the active project via the ActiveProject property.

```
Mtb.Application mtbApp = new Mtb.Application();
mtbApp.UserInterface.Visible = true;
MessageBox.Show("Status = " + mtbApp.Status + "\r\n" + "LastError = " + mtbApp.LastError + "\r\n" +
```
"Default File Path = " + mtbApp.Options.DefaultFilePath + "
"Application Path = " + mtbApp.AppPath + "
"Window Handle = " + mtbApp.Handle);

mtbApp.ActiveProject.Comment = "New Minitab Project."

Application property - ActiveProject

Description
Returns the currently active project.

Type
Project

Range
N/A

Access
Read-only

Application property - AppPath

Description
The path to the currently running Minitab executable.

Type
String

Range
Valid string

Access
Read-only

Application property - Handle

Description
The handle to the main Minitab window.

Type
Long

Range
Any valid long integer

Access
Read-only
Application property - LastError

**Description**
Holds the last error that occurred during execution of a command.

**Type**
String

**Range**
Valid string

**Access**
Read-only

After executing an asynchronous command, the `Status` property should be checked to see when the command completes. If the `Status` property indicates an error occurred then use the `LastError` property to retrieve the error message. For more information on the `Status` property, go to Application property - Status on page 24.

Application property - Options

**Description**
Returns the `ApplicationOptions` object.

**Type**
`ApplicationOptions` on page 26

**Range**
N/A

**Access**
Read-only

Application property - Status

**Description**
Indicates the current status of the Minitab application.

**Type**
`MtbAppStatusTypes` on page 6

**Range**
Any `MtbAppStatusTypes` constant

**Access**
Read-only

After executing an asynchronous command, the `Status` property should be checked to see when the command completes. If the `Status` property indicates an error occurred then use the `LastError` property to retrieve the error message. For more information on the `LastError` property, go to Application property - LastError on page 24.
Application property - UserInterface

Description
Returns the UserInterface object of the application.

Type
UserInterface on page 29

Range
N/A

Access
Read-only

Application method - Help
Use to launch this Help file.

Syntax
Help()

Returns
HRESULT

Example
Call the online Help file.
Mtb.Application mtbApp = new Mtb.Application();
mtbApp.Help();

Application method - New
Use to create a new project and make it the active project.

Syntax
New()

Returns
HRESULT

Example
Create a new project and make it the active project.
Mtb.Application mtbApp = new Mtb.Application();
mtbApp.New();
Application method - Open

Use to open an existing project file and make it the active project.

Syntax

Open(Filename as String)

Arguments

Filename

Required. The path and name of the project file to be opened. If a path is not specified, the DefaultFilePath is used. For more information on the DefaultFilePath, go to ApplicationOptions property - DefaultFilePath on page 28.

Returns

HRESULT

Example

Open an existing project file and make it the active project.
Mtb.Application mtbApp = new Mtb.Application();
mtbApp.Open("C:\\MyProject.mpj");

Application method - Quit

Use to close and delete all objects in the Application object hierarchy, including the Application object.

Syntax

Quit()

Returns

HRESULT

Example

Delete all the objects in the Application Object hierarchy, including the mtbApp.
Mtb.Application mtbApp = new Mtb.Application();
mtbApp.Quit();

ApplicationOptions object

Use the ApplicationOptions object to read or set options that pertain to the Application object. Use the Options property of the Application object to access ApplicationOptions. For more information on the Application object, go to Application object on page 22.
## Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ClientMissingValueDateTime</strong> on page 27</td>
<td>Date/time to use to represent missing date/time values when receiving date/time values from a client or giving date/time values to a client.</td>
</tr>
<tr>
<td><strong>ClientMissingValueNumeric</strong> on page 28</td>
<td>Number to use to represent missing numeric values when receiving numeric data from a client or giving numeric data to a client.</td>
</tr>
<tr>
<td><strong>DefaultFilePath</strong> on page 28</td>
<td>Default file path used by the application for opening/saving files.</td>
</tr>
<tr>
<td><strong>DefaultOutputFileType</strong> on page 29</td>
<td>Returns or sets the default type of output document file that will be produced.</td>
</tr>
</tbody>
</table>

### Example

Create a Minitab Application object (mtbApp), then use the ApplicationOptions object to access and display the DefaultFilePath in a message box. For more information on the ApplicationOptions object, go to ApplicationOptions object on page 26.

```csharp
Mtb.Application mtbApp = new Mtb.Application();
MessageBox.Show("The current default file path is " + mtbAppOpt.DefaultFilePath);
```

### ApplicationOptions property - ClientMissingValueDateTime

**Description**

Date/time to use to represent missing date/time values when receiving date/time values from a client or giving date/time values to a client.

**Type**

Date

**Range**

Valid COM DATE value

**Default**

12/31/9999

**Access**

Read/Write

This property does not affect Minitab’s convention that uses "*" to represent missing values; therefore, in the Minitab worksheet missing values will always appear as "*".

### Example

Set the **ClientMissingValueDateTime** to June 1, 2018 for a Minitab Application object, then display it in a message box.

```csharp
Mtb.Application mtbApp = new Mtb.Application();

DateTime missingDate = new DateTime(2018, 6, 1);
mtbAppOpt.ClientMissingValueDateTime = missingDate;

MessageBox.Show("The ClientMissingValueDateTime is " + mtbAppOpt.ClientMissingValueDateTime);
```
ApplicationOptions property - ClientMissingValueNumeric

**Description**
Number to use to represent missing numeric values when receiving numeric data from a client or giving numeric data to a client.

**Type**
Double

**Range**
Valid double precision value

**Default**
1.23456E30

**Access**
Read/Write

This property does not affect Minitab's convention that uses "*" to represent missing values; therefore, in the Minitab worksheet missing numeric values will always appear as "*".

**Example**
Set the ClientMissingValueNumeric to 0.001 for a Minitab Application object, then display it in a message box.

```csharp
Mtb.Application mtbApp = new Mtb.Application();
mtbAppOpt.ClientMissingValueNumeric = 0.001;
MessageBox.Show("The ClientMissingValueNumeric is " + mtbAppOpt.ClientMissingValueNumeric);
```

ApplicationOptions property - DefaultFilePath

**Description**
Default file path used by the application for opening/saving files.

**Type**
String

**Range**
Any valid path

**Access**
Read/Write

The default DefaultFilePath is the directory where the task scheduler will schedule the task to execute.

**Example**
Create a Minitab Application object (mtbApp), then use the ApplicationOptions object to access and display the DefaultFilePath in a message box. For more information on the ApplicationOptions object, go to ApplicationOptions object on page 26.

```csharp
Mtb.Application mtbApp = new Mtb.Application();
MessageBox.Show("The current default file path is " + mtbAppOpt.DefaultFilePath);
```
ApplicationOptions property - DefaultOutputFileType

**Description**
Returns or sets the default type of output document file that will be produced.

**Type**
-MtbOutputFileTypes on page 7

**Range**
Any MtbOutputFileTypes constant

**Access**
Read/Write
The default is HTML.

**Example**
Create a Minitab Application object (mtbApp) and then display the DefaultOutputFileType in a messagebox.

Mtb.Application mtbApp = new Mtb.Application();
MessageBox.Show("The current default output file type is " +
  mtbAppOpt.DefaultOutputFileType);

UserInterface object

The UserInterface object allows control of the Minitab host.

**Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DisplayAlerts</td>
<td>Controls whether Minitab displays alerts and messages while a client script is running.</td>
</tr>
<tr>
<td>Interactive</td>
<td>Controls whether the user is able to issue commands or alter data directly in Minitab if it is visible.</td>
</tr>
<tr>
<td>UserControl</td>
<td>Controls whether Minitab quits when the last client object is released.</td>
</tr>
<tr>
<td>Visible</td>
<td>Controls whether Minitab server is visible.</td>
</tr>
</tbody>
</table>

**Example**
Create a Minitab Application object (mtbApp), make it visible to the user, and set the Interactive, UserControl, and DisplayAlerts properties of the UserInterface object to True.

Mtb.Application mtbApp = new Mtb.Application();
Mtb.UserInterface mtbUI = mtbApp.UserInterface;

mtbUI.Visible = true;
mtbUI.Interactive = true;
mtbUI.UserControl = true;
mtbUI.DisplayAlerts = true;
UserInterface property - DisplayAlerts

**Description**
Controls whether Minitab displays alerts and messages while a client script is running.

**Type**
Boolean

**Range**
True/False

**Access**
Read/Write

It is good practice to set `DisplayAlerts` back to True when a script finishes. Minitab does not do this automatically.

UserInterface property - Interactive

**Description**
Controls whether the user is able to issue commands or alter data directly in Minitab if it is visible.

**Type**
Boolean

**Range**
True/False

**Access**
Read/Write

The default is True.

UserInterface property - UserControl

**Description**
Controls whether Minitab quits when the last client object is released.

**Type**
Boolean

**Range**
True/False

**Access**
Read/Write

The default is False if the application was started programmatically. Minitab must be visible for `UserControl` to be True.

UserInterface property - Visible

**Description**
Controls whether Minitab server is visible.
Type
  Boolean

Range
  True/False

Access
  Read/Write

The default is True when Minitab is started by the user, False when started programatically.

Project object

The Project object contains all the information related to an individual project, including the Worksheets collection and the Commands collection.

For more information on the Worksheets collection, go to Worksheets Collection object on page 40. For more information on the Commands collection, go to Commands Collection object on page 83.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveWorksheet on page 32</td>
<td>Returns or sets the active worksheet for the project.</td>
</tr>
<tr>
<td>Commands on page 33</td>
<td>Returns the Commands collection for the project.</td>
</tr>
<tr>
<td>Comment on page 33</td>
<td>Comment for the project.</td>
</tr>
<tr>
<td>Creator on page 33</td>
<td>Creator of the project.</td>
</tr>
<tr>
<td>Date on page 34</td>
<td>Date of the project.</td>
</tr>
<tr>
<td>FullName on page 34</td>
<td>Full name of the Project object disk file, including the path name and/or drive name, set when a project is opened or saved.</td>
</tr>
<tr>
<td>Name on page 35</td>
<td>Name of the project and its disk file.</td>
</tr>
<tr>
<td>Path on page 35</td>
<td>Path of the Project object disk file, set when a project is opened or saved</td>
</tr>
<tr>
<td>Worksheets on page 36</td>
<td>Returns the Worksheets collection of the project.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CancelCommand on page 36</td>
<td>Use to cancel the execution of a user-issued or COM-issued command.</td>
</tr>
<tr>
<td>Delete on page 36</td>
<td>Use to delete the Project object and the underlying Worksheets and Commands collections.</td>
</tr>
<tr>
<td>ExecuteCommand on page 37</td>
<td>Use to run a Minitab session command and create a command object.</td>
</tr>
<tr>
<td>ExecuteCommandAsync on page 37</td>
<td>Use to run a Minitab session command asynchronously and create a command object.</td>
</tr>
<tr>
<td>Save on page 38</td>
<td>Use to save the project to FullName.</td>
</tr>
<tr>
<td>SaveAs on page 38</td>
<td>Use to save a copy of the project.</td>
</tr>
</tbody>
</table>
Example
Create a Minitab Application object, execute Minitab commands both synchronously and asynchronously, and then attempt to cancel the asynchronous command. Also use the Comment, Creator, and Date properties as well as the Save and SaveAs methods.

```csharp
Mtb.Application mtbApp = new Mtb.Application();
Mtb.Project mtbProj = mtbApp.ActiveProject;
Mtb.MtbAppStatusTypes status;

mtbProj.ExecuteCommand("rand 30 c1");
mtbProj.ExecuteCommandAsync("rand 100000 c2-c100");

status = mtbApp.Status;
if (status == Mtb.MtbAppStatusTypes.ASBusy)
{
    mtbProj.CancelCommand();
}

mtbApp.UserInterface.Visible = true;

Save the project as a Release 19 project called MyProject. For Minitab 19 and higher, 19 is the earliest valid argument.
mtbProj.SaveAs("C:\MyProject",true,19);

Add creator, date, and comment information.
mtbProj.Creator = "Me";
mtbProj.Date = DateTime.Now.ToShortDateString();
mtbProj.Comment = "This is my project.";

Display creator, date, and comment information.
MessageBox.Show("This project created by " + mtbProj.Creator + 
    " on " + mtbProj.Date + "\r\nComment: " + mtbProj.Comment);

Save the project again.
mtbProj.Save();
```

Project property - ActiveWorksheet

**Description**
Returns or sets the active worksheet for the project.

**Type**
Worksheet on page 44

**Range**
N/A

**Access**
Read/Write

Example
Create a Minitab Application object, then rename the project's active worksheet and display a message with the new name. For more information on the Application object, go to Application object on page 22.

```csharp
Mtb.Application mtbApp;
Mtb.Worksheet mtbSheet;

mtbApp = new Mtb.Application();
mtbSheet = mtbApp.ActiveProject.ActiveWorksheet;
```
mtbSheet.Name = "My Worksheet";
MessageBox.Show("The active worksheet is called " + 
mtbApp.ActiveProject.ActiveWorksheet.Name);

Project property - Commands

**Description**
Returns the Commands collection for the project. For more information on the Commands collection, go to
Commands Collection object on page 83.

**Type**
Commands collection

**Range**
N/A

**Access**
Read-only

**Example**
Retrieve the Commands collection from the Project object and display a message with the number of commands
in the collection.

Mtb.Application mtbApp = new Mtb.Application();
Mtb.Project mtbProj = mtbApp.ActiveProject;
Mtb.Commands mtbCommands = mtbProj.Comands;

MessageBox.Show("The number of commands that has been run is: " + 
mtbCommands.Count.ToString());

Project property - Comment

**Description**
Comment for the project.

**Type**
String

**Range**
Valid string

**Access**
Read/Write
Default is blank.

Project property - Creator

**Description**
Creator of the project.

**Type**
String
**Range**
Valid string

**Access**
Read/Write
Default is blank.

**Project property - Date**

**Description**
Date of the project.

**Type**
String

**Range**
Valid string

**Access**
Read/Write
Default is blank

**Project property - FullName**

**Description**
Full name of the Project object disk file, including the path name and/or drive name, set when a project is opened or saved.

**Type**
String

**Range**
Valid file name, including the path name and/or drive name

**Access**
Read-only

**Example**
Display the FullName property (path and name) of the project in a message box.

```csharp
Mtb.Application mtbApp;
Mtb.Project mtbProj;

mtbApp = new Mtb.Application();
mtbProj = mtbApp.ActiveProject;

mtbProj.Name = "My Project";
MessageBox.Show("The FullName is " + mtbProj.FullName);
```
Project property - Name

**Description**
Name of the project and its disk file. It also is the name of the file if the project is saved to disk. Setting the `Name` property automatically updates the file name portion of the `FullName` property.

**Type**
String

**Range**
Any valid file name

**Access**
Read/Write

Do not include the path when setting the `Name` property.

**Example**
Retrieve the active project, name it "My Project," and print the name in a message box.
```csharp
Mtb.Application mtbApp;
Mtb.Project mtbProj;

mtbApp = new Mtb.Application();
mtbProj = mtbApp.ActiveProject;

mtbProj.Name = "My Project";
MessageBox.Show("The project is " + mtbProj.Name);
```

Project property - Path

**Description**
Path of the Project object disk file, set when a project is opened or saved

**Type**
String

**Range**
Valid path name. It may include the drive name.

**Access**
Read-only

**Example**
Display the `Path` property of the project in a message box.
```csharp
Mtb.Application mtbApp;
Mtb.Project mtbProj;

mtbApp = new Mtb.Application();
mtbProj = mtbApp.ActiveProject;

MessageBox.Show("The path is " + mtbProj.Path);
```
Project property - Worksheets

**Description**

Returns the Worksheets collection of the project. For more information on the Worksheets collection, go to *Worksheets Collection object* on page 40.

**Type**

Worksheets collection

**Range**

N/A

**Access**

Read-only

**Example**

Retrieve the Worksheets collection from the Project object and display a message with the number of worksheets in the collection.

```csharp
Mtb.Application mtbApp;
Mtb.Project mtbProj;
Mtb.Worksheets mtbSheets;

mtbApp = new Mtb.Application();
mtbProj = mtbApp.ActiveProject;
mtbSheets = mtbProj.Worksheets;

MessageBox.Show("There are this many worksheets in the project: " + mtbSheets.Count);
```

---

Project method - CancelCommand

Use to cancel the execution of a user-issued or COM-issued command.

**Syntax**

```csharp
CancelCommand()
```

**Returns**

Boolean

**Remarks**

Returns True if the command was cancelled, False if no command was executing.

---

Project method - Delete

Use to delete the Project object and the underlying Worksheets and Commands collections. Delete also sets the active project for the application to NULL.

For more information on the Worksheets collection, go to *Worksheets Collection object* on page 40. For more information on the Commands collection, go to *Commands Collection object* on page 83.
Syntax
Delete()

Returns
HRESULT

Example
Delete the Project object and the underlying Worksheets and Commands collections.
Mtb.Application mtbApp;
Mtb.Project mtbProj;
mtbApp = new Mtb.Application();
mtbProj = mtbApp.ActiveProject;
mtbProj.Delete();

Project method - ExecuteCommand
Use to run a Minitab session command and create a command object.

Syntax
ExecuteCommand(Command as String, WorksheetObj as Worksheet)

Arguments

Command
Required. One or more session commands to execute in Minitab. Multiple commands and subcommands may be included in the same command. Subcommands must be separated from commands and from each other by semicolons. Each command must end with a period, with the exception of LET. LET commands must be separated from other commands by a new line rather than a period.

WorksheetObj
Optional. The worksheet to use when executing the command. The specified worksheet becomes the ActiveWorksheet. If none specified, then the current ActiveWorksheet is used.

Remarks
The command is executed synchronously, meaning this interface will not return until the command has completed executing, giving direct feedback as to the completion status of the command, success or failure.

Project method - ExecuteCommandAsync
Use to run a Minitab session command asynchronously and create a command object.

Syntax
ExecuteCommandAsync(Command as String, WorksheetObj as Worksheet)
Arguments

Command

Required. One or more session commands to execute in Minitab. Multiple commands and subcommands may be included in the same command. Subcommands must be separated from commands and from each other by semicolons. Each command must end with a period, with the exception of LET. LET commands must be separated from other commands by a new line rather than a period.

WorksheetObj

Optional. The worksheet to use when executing the command. The specified worksheet becomes the ActiveWorksheet. If none specified, then the current ActiveWorksheet is used.

Returns

HRESULT

Remarks

The command is submitted for execution asynchronously, meaning this interface will return before the command is executed. Use the Application object's Status property to see if the command completed successfully. If an error occurred, use the application's LastError property to retrieve the error message.

For more information on the Application object, go to Application object on page 22. For more information on the Status property, go to Application property - Status on page 24. For more information on the LastError property, go to Application property - LastError on page 24.

Project method - Save

Use to save the project to FullName.

For more information on FullName, go to Project property - FullName on page 34.

Syntax

Save() 

Returns

HRESULT

Project method - SaveAs

Use to save a copy of the project.

Syntax

SaveAs(Filename as String, Replace as Boolean, Version as Long)
Arguments

**Filename**
Optional. Path and file name to use when saving the project. If a path is not specified, then the `DefaultFilePath` is used. If a file name is not specified then the `Name` property is used.

**Replace**
Optional. If `True`, an existing file with the same name will be overwritten. The default is `True`.

**Version**
Optional. The Minitab version number to save the project as. If not specified, the current version number is used.

For more information on the `DefaultFilePath`, go to ApplicationOptions property - DefaultFilePath on page 28.  
For more information on the `Name` property, go to Project property - Name on page 35.

Returns

HRESULT
Worksheets Collection object

The Worksheets collection is a set of all the Worksheet objects within a Project object. It supports the standard collection properties and methods.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count on page 40</td>
<td>Number of Worksheet objects within the Worksheets collection.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add on page 41</td>
<td>Use to add Count Worksheet objects to the Worksheets collection.</td>
</tr>
<tr>
<td>Delete on page 41</td>
<td>Use to remove all Worksheet objects from the Worksheets collection.</td>
</tr>
<tr>
<td>Item on page 42</td>
<td>Use to return a Worksheet object within the Worksheets collection.</td>
</tr>
<tr>
<td>Open on page 43</td>
<td>Use to open an existing Minitab worksheet disk file, read it into a Worksheet object, and add the Worksheet object to the end of the Worksheets collection.</td>
</tr>
<tr>
<td>Remove on page 43</td>
<td>Use to delete a Worksheet object and remove it from the Worksheets collection.</td>
</tr>
</tbody>
</table>

Example

Retrieve the Worksheets collection from the Project, add two worksheets to it and name the first one "First Year," open an existing Minitab worksheet ("Market"), then remove the second worksheet from the Worksheets collection:

```csharp
Mtb.Worksheets mtbSheets;
mtbSheets = mtbProj.Worksheets;

mtbSheets.Add(2).Name = "First Year";
mtbSheets.Open("Market");
mtbSheets.Remove(2);
```

Note This example assumes that the MtbProject object was previously initialized to a valid Project object as demonstrated in the Project object example.

Worksheets Collection property - Count

**Description**

Number of Worksheet objects within the Worksheets collection.

**Type**

Long
Range
0 - number of Worksheet objects within the Worksheets collection

Access
Read-only

Example
Retrieve the Worksheets collection from a previously initialized Project object, then display the number of Worksheet objects in the Worksheets collection in a message box.

tbSheets = mtbProj.Worksheets;
MessageBox.Show(tbSheets.Count.ToString());

Worksheets Collection method - Add
Use to add Count Worksheet objects to the Worksheets collection.

Syntax
Add(Quantity Long String)

Arguments

Quantity
Optional. Number of worksheets to add. The default is 1.

Returns
Worksheet

Remarks
The first worksheet added is returned.

Examples
Retrieve the Worksheets collection, add two worksheets to it, naming the first one "Growth."

tbSheets = mtbProj.Worksheets;
tbSheets.Add(2).Name = "Growth";

Add one worksheet to the Worksheets collection.

tbSheets.Add();

Worksheets Collection method - Delete
Use to remove all Worksheet objects from the Worksheets collection.

For more information on the Worksheet object, go to Worksheet object on page 44.
Syntax
Delete()

Returns
HRESULT

Remarks
To remove a single worksheet, use Remove or the Worksheet object method, Delete.

For more information on the Remove method for the Worksheets collection, go to Worksheets Collection method - Remove on page 43. For more information on the Delete method for a worksheet object, go to Worksheet method - Delete on page 49.

Example
Delete the Worksheets collection, including all its worksheets:
mtbSheets.Delete();

Worksheets Collection method - Item
Use to return a Worksheet object within the Worksheets collection.

Syntax
Item(Index as Variant)

Arguments
Index
Required. The index of the worksheet as an integer (Long) from 1 - the number of worksheets in the collection, or the name (String) of the worksheet. For more information on the name property of the worksheet, go to Worksheet property - Name on page 48.

Returns
Worksheet

Examples
Retrieve the second worksheet in the Worksheets collection, name the worksheet "First Year," then print the name in a message box.
mtbSheet = mtbSheets.Item(2);
mtbSheet.Name = "First Year";
MessageBox.Show( "The second worksheet is: " + mtbSheet.Name);

Retrieve the worksheet called "Second Year" and print the name in a message box.
mtbSheet = mtbSheets.Item( "Second Year" );
MessageBox.Show( "The current worksheet is: " + mtbSheet.Name );
Worksheets Collection method - Open

Use to open an existing Minitab worksheet disk file, read it into a Worksheet object, and add the Worksheet object to the end of the Worksheets collection.

Syntax

Open(Filename as String)

Arguments

Filename
Optional. The path and name of the worksheet file to be opened. If a path is not specified, the DefaultFilePath is used. For more information on the DefaultFilePath, go to ApplicationOptions property - DefaultFilePath on page 28.

Returns

HRESULT

Remarks

When you open a worksheet file, the Name, Path, and FullName properties of the Worksheet object are automatically updated. If you don't specify an extension,.MWX is automatically added.

Example

Open the Minitab worksheet "Market.mtw", retrieve the first worksheet, then print the name in a message box.

mtbSheets.Open("C:\\sheets\\Market.mtw");
mtbSheet = mtbSheets.Item(1);
MessageBox.Show("Worksheet name: " + mtbSheet.Name);

Worksheets Collection method - Remove

Use to delete a Worksheet object and remove it from the Worksheets collection.

Syntax

Remove(Index as Variant)

Arguments

Index
Required. The index of the worksheet as an integer (Long) from 1 - the number of worksheets in the collection, or the name on page 48 (String) of the worksheet.

Returns

HRESULT
Remarks

The same results can be achieved using the `Delete` method of the `Worksheet` object. To remove all worksheets, use the `Delete` method of the `Worksheets` collection object.

For more information on the `Delete` method for a `Worksheet` object, go to `Worksheet method - Delete` on page 49. For more information on the `Worksheet` object, go to `Worksheet object` on page 44. For more information on the `Delete` method of the `Worksheets` collection object, go to `Worksheets Collection method - Delete` on page 41. For more information on the `Worksheets` collection object, go to `Worksheets Collection object` on page 40.

Example

Remove the first worksheet and the worksheet named "First Year" from the `Worksheets` collection.

```csharp
mtbSheets.Remove(1);
mtbSheets.Remove("First Year");
```

Worksheet object

The `Worksheet` object contains all the information related to an individual worksheet, including the `Columns`, `Constants`, and `Matrices` collections, which provide access to all the columns, constants, and matrices in the worksheet.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns</td>
<td>Returns the <code>Columns</code> collection of the worksheet.</td>
</tr>
<tr>
<td>Comment</td>
<td>Comment for the <code>Worksheet</code> object, saved as part of the Minitab worksheet description.</td>
</tr>
<tr>
<td>Constants</td>
<td>Returns the <code>Constants</code> collection of the worksheet.</td>
</tr>
<tr>
<td>Creator</td>
<td>Creator of the <code>Worksheet</code> object, saved as part of the Minitab worksheet description.</td>
</tr>
<tr>
<td>Date</td>
<td>Date of the <code>Worksheet</code> object description, saved as part of the Minitab worksheet description.</td>
</tr>
<tr>
<td>FullName</td>
<td>Full name of the <code>Worksheet</code> object disk file, including the path name and/or drive name, set when a <code>Worksheet</code> object is opened or saved.</td>
</tr>
<tr>
<td>Matrices</td>
<td>Returns the <code>Matrices</code> collection of the worksheet.</td>
</tr>
<tr>
<td>Name</td>
<td>Name of the <code>Worksheet</code> object and its disk file.</td>
</tr>
<tr>
<td>Path</td>
<td>Path of the <code>Worksheet</code> object disk file, set when a <code>Worksheet</code> object is opened or saved.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete</td>
<td>Use to delete a <code>Worksheet</code> object and remove it from the <code>Worksheets</code> collection.</td>
</tr>
<tr>
<td>Save</td>
<td>Use to save a <code>Worksheet</code> object to disk with the file name specified in the <code>FullName</code> property.</td>
</tr>
</tbody>
</table>
**Method**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SaveAs</strong> on page 50</td>
</tr>
</tbody>
</table>

**Example**

Retrieve the Worksheet object named "First Year", set the creator, date, and comment for the worksheet, then save the worksheet as "Year1."

```plaintext
Mtb.Worksheet mtbSheet;
mtbSheet = mtbSheets.Item("First Year");
mtbSheet.Creator = "M. Smith";
mtbSheet.Date = "6/4/2002";
mtbSheet.Comment = "1999 is the first year";
mtbSheet.SaveAs("Year1");
```

**Worksheet property - Columns**

**Description**

Returns the Columns collection of the worksheet.

**Type**

Columns on page 50

**Range**

N/A

**Access**

Read-only

**Example**

Retrieve the Columns collection.

```plaintext
mtbColumns = mtbSheet.Columns;
```

**Note**

This example assumes that the mtbSheet object was previously initialized to a valid Worksheet object as demonstrated in the Worksheet object example. For the example, go to Worksheet object on page 44.

**Worksheet property - Comment**

**Description**

Comment for the Worksheet object, saved as part of the Minitab worksheet description.

**Type**

String

**Range**

Valid string

**Access**

Read/Write
Example

Retrieve the first worksheet in the **Worksheets** collection and add a comment to the worksheet.
```vba
mtbSheet = mtbSheets.Item(1);
mtbSheet.Comment = "This worksheet has the old data. It needs to be updated by the end of the year.";
```

**Worksheet property - Constants**

**Description**
Returns the **Constants** collection of the worksheet.

**Type**
- **Constants** on page 64

**Range**
- N/A

**Access**
- Read-only

**Example**

Set **mtbConstants** to the **Constants** collection of worksheet **mtbSheet**.
```vba
mtbConstants = mtbSheet.Constants;
```

**Note**
This example assumes that the **mtbSheet** object was previously initialized to a valid **Worksheet** object as demonstrated in the **Worksheet object** example. For the example, go to **Worksheet object** on page 44.

**Worksheet property - Creator**

**Description**
Creator of the **Worksheet** object, saved as part of the Minitab worksheet description.

**Type**
- String

**Range**
- Valid string

**Access**
- Read/Write

**Example**

Retrieve the first worksheet in the **Worksheets** collection, specify the creator of the worksheet (M. Smith), then print the creator in a message box.
```vba
mtbSheet = mtbSheets.Item(1);
mtbSheet.Creator = "M. Smith";
MessageBox.Show("The creator is " + mtbSheet.Creator);
```
Worksheet property - Date

**Description**
Date of the Worksheet object description, saved as part of the Minitab worksheet description.

**Type**
String

**Range**
Valid string

**Access**
Read/Write

**Example**
Retrieve the first worksheet in the **Worksheets** collection, specify the date of the worksheet, then print the date in a message box.
```
mtbSheet = mtbSheets.Item(1);
mtbSheet.Date = "6/4/2018";
MessageBox.Show("The date is " + mtbSheet.Date);
```

Worksheet property - FullName

**Description**
Full name of the Worksheet object disk file, including the path name and/or drive name, set when a Worksheet object is opened or saved.

**Type**
String

**Range**
Valid file name, including path name and/or drive name

**Access**
Read/Write

**Example**
Display the **FullName** property (path and name) of the worksheet in a message box.
```
MessageBox.Show("The FullName is " + mtbSheet.FullName);
```

Worksheet property - Matrices

**Description**
Returns the Matrices collection of the worksheet.

**Type**
Matrices on page 73
Example

Set `mtbMatrices` to the `Matrices` collection of worksheet `mtbSheet`.

```
mtbMatrices = mtbSheet.Matrices;
```

**Note** This example assumes that the `mtbSheet` object was previously initialized to a valid `Worksheet` object as demonstrated in the `Worksheet` object example. For the example, go to Worksheet object on page 44.

---

### Worksheet property - Name

**Description**

Name of the `Worksheet` object and its disk file. It is also the name of the file if the `Worksheet` object is saved to disk. Setting `Name` automatically updates the file name portion of the `FullName` property.

**Type**

String

**Range**

Any valid file name

**Access**

Read/Write

Do not include the path when setting `Name`.

**Example**

Retrieve the second worksheet in the `Worksheets` collection, name the worksheet "Second Year," then print the name in a message box.

```
mtbSheet = mtbSheets.Item(2);
mtbSheet.Name = "Second Year";
MessageBox.Show("The second worksheet is " + mtbSheet.Name);
```

---

### Worksheet property - Path

**Description**

Path of the `Worksheet` object disk file, set when a `Worksheet` object is opened or saved.

**Type**

String

**Range**

Valid path name. It may include the drive name.

**Access**

Read-only
Example
Display the Path property of the worksheet in a message box.
MessageBox.Show("The Path is " + mtbSheet.Path);

Worksheet method - Delete
Use to delete a Worksheet object and remove it from the Worksheets collection.

Syntax
Delete()

Returns
HRESULT

Remarks
The same results can be achieved using the Remove method of the Worksheets collection object. To remove all worksheets, use the Delete method of the Worksheets collection object.

For more information on the Remove method for a Worksheets collection object, go to Worksheets Collection method - Remove on page 43. For more information on the Delete method of the Worksheets collection object, go to Worksheets Collection method - Delete on page 41. For more information on the Worksheets collection object, go to Worksheets Collection object on page 40.

Example
Delete the Worksheet object from the Worksheets collection.
mtbSheet.Delete();

Worksheet method - Save
Use to save a Worksheet object to disk with the file name specified in the FullName property.

Syntax
Save(Filename as String)

Returns
HRESULT

Remarks
When you save a worksheet, the Name, Path, and FullName properties of the Worksheet object are automatically updated. If you don't specify an extension, .MWX is automatically added to the worksheet name. If the FullName property is null, then the worksheet is saved to Minitab.MWX at the default file path.
Example

Save the current worksheet.
mtbSheet.Save();

Worksheet method - SaveAs

Use to save a Worksheet object to disk with the file name specified in the Filename argument.

Syntax

SaveAs(Filename as String, Replace as Boolean, Version as Long)

Arguments

Filename
Optional. Path and file name to use when saving the file. If a path is not specified, then the DefaultFilePath is used. For more information on the DefaultFilePath, go to ApplicationOptions property - DefaultFilePath on page 28.

Replace
Optional. If True, an existing file with the same name will be overwritten. The default is False.

Version
Optional. The Minitab version number to save the worksheet as. Valid parameters are 19 or later.

Returns

HRESULT

Remarks

When you save a worksheet, the Name, Path, and FullName properties of the Worksheet object are automatically updated. With the release of Minitab 19 and tabbed output, if you don't specify an extension, .MWX is automatically added to the worksheet name. Earlier versions of Minitab use .MTW. You cannot save worksheets as .MTW files in versions of Minitab that do not support tabbed output.

Examples

Save the current worksheet as "April Totals" at the file path C:\MTBsheets.
mtbSheet.SaveAs("C:\MTBsheets\April Totals");

Save the current worksheet as “April Totals” at the default file path, overwriting the existing "April Totals".
mtbSheet.SaveAs("April Totals",true);

Columns Collection object

The Columns collection is a set of all the Column objects within a Worksheet object. It supports the standard collection properties and methods.

The Columns collection for a worksheet is empty by default.
Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count on page 51</td>
<td>Number of Column objects within the Columns collection.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add on page 52</td>
<td>Use to add Quantity Column objects to the Columns collection in the position before Before or after After.</td>
</tr>
<tr>
<td>Delete on page 53</td>
<td>Use to remove all Column objects from the Columns collection.</td>
</tr>
<tr>
<td>Item on page 53</td>
<td>Use to return a Column object within the Columns collection.</td>
</tr>
<tr>
<td>Remove on page 54</td>
<td>Use to delete a Column object and remove it from the Columns collection.</td>
</tr>
</tbody>
</table>

Example

Retrieve the Columns collection (mtbColumns), add two columns to the end of the collection and name the first one “Sales,” then remove the second column from the Columns collection.

```csharp
Mtb.Application mtbApp;
Mtb.Worksheet mtbSheet;
Mtb.Columns mtbColumns;

mtbApp = new Mtb.Application();
mtbSheet = mtbApp.ActiveProject.ActiveWorksheet;
mtbColumns = mtbSheet.Columns;

mtbApp.UserInterface.Visible = true;
mtbApp.UserInterface.UserControl = true;

mtbColumns.Add(null, null, 2).Name = "Sales";
mtbColumns.Remove(2);
```

Columns Collection property - Count

Description
Number of Column objects within the Columns collection.

Type
Long

Range
0 - number of Column objects within the Columns collection

Access
Read-only

Example

Retrieve the Columns collection, then display in a message box the number of Column objects in the Columns collection.

```csharp
Mtb.Application mtbApp;
Mtb.Worksheet mtbSheet;
```
Mtb.Columns mtbColumns;
Mtb.Column mtbColumn;

mtbApp = new Mtb.Application();
mtbSheet = mtbApp.ActiveProject.ActiveWorksheet;
mtbColumns = mtbSheet.Columns;
MessageBox.Show(mtbColumns.Count.ToString());

Columns Collection method - Add

Use to add Quantity Column objects to the Columns collection in the position before Before or after After.

Syntax

Add(Before as Variant, After as Variant, Quantity as Long)

Arguments

Before
Optional. Column object to add new columns before.

After
Optional. Column object to add new columns after.

Quantity
Optional. Number of columns to add. The default is 1.

Returns

Column on page 55

Remarks

You can specify either Before or After, but not both. Use an integer (Long) from 1 - the number of columns in the collection, or the name(String) of a column. If neither Before nor After is specified, then the columns are added after the last column in the collection. For more information on the Name property, go to Column property - Name on page 59.

The first column added is returned.

Examples

Create a Minitab Application object, add four columns to the active worksheet, and name the first column “Year.”

Mtb.Application mtbApp;
Mtb.Worksheet mtbSheet;
Mtb.Columns mtbColumns;
Mtb.Column mtbColumn;

mtbApp = new Mtb.Application();
mtbSheet = mtbApp.ActiveProject.ActiveWorksheet;
mtbColumns = mtbSheet.Columns;
mtbColumns.Add(null, null, 4).Name = "Year";
Add one column to it after the last column.
mtbColumns.Add();

Add two columns to the Columns collection before column three, and name the first column "First Time."
mtbColumns.Add(3, null, 2).Name = "First Time";

Add two columns to the Columns collection after column three.
mtbColumns.Add(null, 3, 2);

Add four columns to the Columns collection before the “Year” column and name the first column “Next Year.”
mtbColumns.Add("Year", null, 4).Name = "Next Year";

Add two columns to the Columns collection after the “Year” column.
mtbColumns.Add(null, "Year", 2);

Columns Collection method - Delete
Use to remove all Column objects from the Columns collection.

Syntax
Delete()

Returns
HRESULT

Remarks
To remove a single column, use Remove on page 54 or the Column object method, Delete on page 60.

Example
Delete the Columns collection (MtbColumns), including all its columns.
Mtb.Application mtbApp;
Mtb.Worksheet mtbSheet;
Mtb.Columns mtbColumns;

mtbApp = new Mtb.Application();
mtbSheet = mtbApp.ActiveProject.ActiveWorksheet;
mtbColumns = mtbSheet.Columns;

mtbColumns.Delete();

Columns Collection method - Item
Use to return a Column object within the Columns collection.

Syntax
Item(Index as Variant)
Arguments

**Index**

Required. The index of the column as an integer (**Long**) from 1 - the number of columns in the collection, or the name on page 48 (**String**) of the column.

Returns

**Column** on page 55

Examples

Create a Minitab **Application** object, add four columns to the active worksheet, and name the first column "Mileage."

```csharp
Mtb.Application mtbApp;
Mtb.Worksheet mtbSheet;
Mtb.Columns mtbColumns;
Mtb.Column mtbColumn;

mtbApp = new Mtb.Application();
mtbSheet = mtbApp.ActiveProject.ActiveWorksheet;
mtbColumns = mtbSheet.Columns;

mtbColumns.Add(null, null, 4).Name = "Mileage";
```

Retrieve the second column in the **Columns** collection, name the column "Range," then print the name in a message box.

```csharp
mtbColumn = mtbColumns.Item(2);
mtbColumn.Name = "Range";
MessageBox.Show("The second column is " + mtbColumn.Name);
```

Retrieve the column called "Mileage" then print the name in a message box.

```csharp
mtbColumn = mtbColumns.Item("Mileage");
MessageBox.Show("The current column is " + mtbColumn.Name);
```

Columns Collection method - Remove

Use to **delete** a **Column** object and remove it from the **Columns** collection.

Syntax

```csharp
Remove(Index as Variant)
```

Arguments

**Index**

Required. The index of the column as an integer (**Long**) from 1 - the number of columns in the collection, or the name (**String**) of the column. For more information on the **Name** property, go to **Column property - Name** on page 59.

Returns

**HRESULT**
Remarks

The same results can be achieved using the Delete method of the Column object. To remove all columns, use the Delete method of the Columns collection object.

For more information on the Delete method of the Column object, go to Column method - Delete on page 60. For more information on the Column object, go to Column object on page 55. For more information on the Delete method of the Columns collection object, go to Columns Collection method - Delete on page 53. For more information on the Columns collection object, go to Columns Collection object on page 50.

Examples

Remove the second column and the column named "First Year" from the Columns collection:

```plaintext
tmbColumns.Remove(2);
tmbColumns.Remove("First Year");
```

Column object

The Column object contains all the information related to an individual column. The data type on page 6 for each Column object can be Text, Numeric, DateTime, or DataUnassigned.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment</td>
<td>Description of the Column object.</td>
</tr>
<tr>
<td>DataType</td>
<td>Type of data in the Column object.</td>
</tr>
<tr>
<td>Formula</td>
<td>Formula for the Column object.</td>
</tr>
<tr>
<td>FormulaStatus</td>
<td>Status of the Formula property for the Column object.</td>
</tr>
<tr>
<td>MissingCount</td>
<td>Number of missing data rows in the Column object</td>
</tr>
<tr>
<td>Name</td>
<td>Name of the Column object.</td>
</tr>
<tr>
<td>Number</td>
<td>Number of the Column object within the Columns collection.</td>
</tr>
<tr>
<td>RowCount</td>
<td>Number of rows in the Column object</td>
</tr>
<tr>
<td>ValueOrderType</td>
<td>Order in which values from text columns will be displayed in output.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear</td>
<td>Use to clear the data in the Column object without deleting the Column object from the Columns collection.</td>
</tr>
<tr>
<td>Delete</td>
<td>Use to delete a Column object and remove it from the Columns collection.</td>
</tr>
<tr>
<td>GetData</td>
<td>Use to get NumRows of data from a Column object, starting at StartRow.</td>
</tr>
<tr>
<td>SetData</td>
<td>Use to set NumRows of Data in the Column object, beginning at StartRow.</td>
</tr>
<tr>
<td>SetValueOrder</td>
<td>Use to set the order in which text values are displayed in output. Column must be of type Text or DataUnassigned.</td>
</tr>
</tbody>
</table>
Examples

Create a Minitab Application object and add a Column object to the Columns collection of the active worksheet. Define and populate the array “arrSales” with the column information, retrieve the Column object (mtbColumn), name it “Sales,” and place the information in arrSales into the “Sales” column. Finally, set the Comment property of the new column.

Mtb.Application mtbApp;
Mtb.Columns mtbColumns;
Mtb.Column mtbColumn;

mtbApp = new Mtb.Application();
mtbColumns = mtbApp.ActiveProject.ActiveWorksheet.Columns;
mtbColumn = mtbColumns.Add(null, null, 1);
mtbColumn.Name = "Sales";
mtbColumn.SetData(arraySales);
mtbColumn.Comment = "Sales data for 1999";

Create a Minitab Application object and add two Column objects to the Columns collection of the active worksheet. Retrieve the second column, add the value “1993” to the third row, set the Name property to “Second Year,” set the ValueOrderType property to 0, and set the Comment property (column description) to “New column for second year data.” Finally, display message boxes with the values of the column's Name, Number, RowCount, MissingCount, DataType, ValueOrderType, and Comment properties.

Mtb.Application mtbApp = new Mtb.Application();
Mtb.Columns mtbColumns;
Mtb.Column mtbColumn;

mtbColumns = mtbApp.ActiveProject.ActiveWorksheet.Columns;
mtbColumns.Add(null, null, 2);
mtbColumn = mtbColumns.Item(2);
object newval = "1993";
mtbColumn.SetData(ref newval, 3, 1);

//Set property values
mtbColumn.Name = "Second Year";
mtbColumn.SetValueOrder(0);
mtbColumn.Comment = "New column for second year data."

//Display messages with column property values
MessageBox.Show("The second column is " + mtbColumn.Name);
MessageBox.Show("This is column number: " + mtbColumn.Number.ToString());
MessageBox.Show("This column has this many rows: " + mtbColumn.RowCount.ToString());
MessageBox.Show("This column has this many missing rows: " + mtbColumn.MissingCount.ToString());
MessageBox.Show("The data type of the column is " + mtbColumn.DataType.ToString());
MessageBox.Show("The ValueOrderType is " + mtbColumn.ValueOrderType.ToString());
MessageBox.Show("The column description is " + mtbColumn.Comment);

Column property - Comment

Description
Description of the Column object.

Type
String

Range
Valid string
Access
Read/Write

Column property - DataType

Description
Type of data in the Column object.

Type
MtbDataTypes on page 6

Range
Any MtbDataTypes constant

Access
Read-only

Column property - Formula

Description
Formula for the Column object.

Type
String

Range
Valid string

Access
Read-only

Example
Create 30 rows of random data in column C1, then create a formula that sets the value of C2 equal to the square of C1. Display a message box showing the value of the Formula and FormulaStatus properties for C2. Finally, change to manual formula calculation using the CFMANUALLY session command, change a value in C1, and display the same message. Notice that FormulaStatus changes from 1 to 2.

```csharp
Mtb.Application mtbApp = new Mtb.Application();
Mtb.Worksheet mtbSheet;
Mtb.Column mtbCol1, mtbCol2;

mtbApp.UserInterface.Visible = true;
mtbApp.ActiveProject.ExecuteCommand("rand 30 c1");
mtbApp.ActiveProject.ExecuteCommand("formula c2=c1**2");
mtbSheet = mtbApp.ActiveProject.Worksheets.Item(1);
mtbCol1 = mtbSheet.Columns.Item(1);
mtbCol2 = mtbSheet.Columns.Item(2);

mtbApp.ActiveProject.ExecuteCommand("cfmanually");
object newval = 20;
```
Column property - FormulaStatus

**Description**

Status of the Formula property for the Column object.

**Type**

*MtbFormulaStatusTypes* on page 6

**Range**

Any *MtbFormulaStatusTypes* constant

**Access**

Read-only

**Example**

Create 30 rows of random data in column C1, then create a formula that sets the value of C2 equal to the square of C1. Display a message box showing the value of the Formula and FormulaStatus properties for C2. Finally, change to manual formula calculation using the CFMANUALLY session command, change a value in C1, and display the same message. Notice that FormulaStatus changes from 1 to 2.

```csharp
Mtb.Application mtbApp = new Mtb.Application();
Mtb.Worksheet mtbSheet;
Mtb.Column mtbCol1, mtbCol2;

mtbApp.UserInterface.Visible = true;
mtbApp.ActiveProject.ExecuteCommand("rand 30 c1");
mtbApp.ActiveProject.ExecuteCommand("formula c2=c1**2");
mtbSheet = mtbApp.ActiveProject.Worksheets.Item(1);

mtbCol1 = mtbSheet.Columns.Item(1);
mtbCol2 = mtbSheet.Columns.Item(2);


mtbApp.ActiveProject.ExecuteCommand("cfmanually");
object newval = 20;
mtbCol1.SetData(ref newval,3,1);

```

Column property - MissingCount

**Description**

Number of missing data rows in the Column object.

**Type**

Long
Range
  N/A

Access
  Read-only

Column property - Name

Description
  Name of the Column object.

Type
  String

Range
  Valid string

Access
  Read/Write

Column property - Number

Description
  Number of the Column object within the Columns collection.

Type
  Long

Range
  1 - number of Column objects within the Columns collection (current Minitab limit is 4000)

Access
  Read-only

Column property - RowCount

Description
  Number of rows in the Column object

Type
  Long

Range
  N/A

Access
  Read-only
Column property - ValueOrderType

**Description**
Order in which values from text columns will be displayed in output.

**Type**
*MtbValueOrderTypes* on page 7

**Range**
Any *MtbValueOrderTypes* constant

**Access**
Read-only

Column method - Clear

*Use to clear the data in the Column object without deleting the Column object from the Columns collection.*

**Syntax**
Clear()

**Returns**
HRESULT

**Examples**
Create a Minitab Application object and add two columns to the active worksheet. Retrieve the second column in the Columns collection, name the column "Second Year," and add the value "1993" to the third row.

```csharp
Mtb.Application mtbApp;
Mtb.Columns mtbColumns;
Mtb.Column mtbColumn;

mtbApp = new Mtb.Application();
mtbApp.UserInterface.Visible = true;
mtbApp.UserInterface.UserControl = true;
mtbColumns = mtbApp.ActiveProject.ActiveWorksheet.Columns;
mtbColumns.Add(null,null,2);

mtbColumn = mtbColumns.Item(2);
mtbColumn.Name = "Second Year";
object newval = "1993";
mtbColumn.SetData(ref newval,3,1);

Clear the data from the Column object without deleting the column itself.
mtbColumn.Clear();
```

Column method - Delete

*Use to delete a Column object and remove it from the Columns collection.*
Syntax
Delete()

Returns
HRESULT

Remarks
The same results can be achieved using the Remove method of the Columns collection object. To delete all columns, use the Delete method of the Columns collection object.

For more information on the Remove method of the Columns collection object, go to Columns Collection method - Remove on page 54. For more information on the Delete method of the Columns collection object, go to Columns Collection method - Delete on page 53. For more information on the Columns collection object, go to Columns Collection object on page 50.

Example
Create a Minitab Application object and add two columns to the active worksheet. Retrieve the second column in the Columns collection, name the column "Second Year," and add the value "1993" to the third row. Finally, delete the Column object from the Columns collection.

Mtb.Application mtbApp;
Mtb.Columns mtbColumns;
Mtb.Column mtbColumn;

mtbApp = new Mtb.Application();
mtbApp.UserInterface.Visible = true;
mtbApp.UserInterface.UserControl = true;
mtbColumns = mtbApp.ActiveProject.ActiveWorksheet.Columns;
mtbColumns.Add(null,null,2);

mtbColumn = mtbColumns.Item(2);
mtbColumn.Name = "Second Year";
object newval = "1993";
mtbColumn.SetData(ref newval,3,1);
mtbColumn.Delete();

Column method - GetData

Use to get NumRows of data from a Column object, starting at StartRow.

Syntax
GetData(StartRow as Long, NumRows as Long)

Arguments
StartRow
Optional. First row to get. The default is 1.

NumRows
Optional. Number of rows to get. The default is 1.
Returns
Variant

Remarks
If neither \textit{StartRow} nor \textit{NumRows} is specified, then \textit{GetData} gets all rows.

Examples
Use the \textit{GetData} method to populate an array with all the values from the current column, then use a loop to print the values in message boxes.

```csharp
double [] cvQ1sales;
cvQ1sales = mtbColumn.GetData();
for (int i = 0; i < (cvQ1sales.Length); i++)
{
    MessageBox.Show(cvQ1sales[i].ToString());
}
```

Get one value (the first value) from the column and print it in a message box.

```csharp
MessageBox.Show("The value is " + mtbColumn.GetData(1, 1).ToString());
```

Get one value (the second value) from the column and print it in a message box.

```csharp
MessageBox.Show("The value is " + mtbColumn.GetData(2, 1).ToString());
```

Column method - \textit{SetData}

Use to set \textit{NumRows} of \textit{Data} in the \textit{Column} object, beginning at \textit{StartRow}.

Syntax
\texttt{SetData(Data as Variant, StartRow as Long, NumRows as Long)}

Arguments

\textbf{Data}

Data to write to the column. Can be numeric, text, or date/time.

\textbf{StartRow}

Optional. First row to set. The default is 1.

\textbf{NumRows}

Optional. Number of rows to set. The default is 1.

Returns
\texttt{HRESULT}

Remarks
If neither \textit{StartRow} nor \textit{NumRows} is specified, \textit{SetData} sets the entire column and deletes all previous entries. Otherwise, entries outside the specified range of rows are not affected.
Examples

Retrieve the first column in the Columns collection, then populate that column with the contents of the array arrIndex.

```csharp
mtbColumn = mtbColumns.Item(1);
mtbColumn.SetData(arrIndex);
```

Place the value 10 in row 20 of the current Minitab column, then print the value for row 20 in the Immediate window.

```csharp
object newval = 10;
mtbColumn.SetData(ref newval, 20, 1);
MessageBox.Show(mtbColumn.GetData(20, 1).ToString());
```

Change the data type of the column from numeric to text, place “Green” in the first row of the column, and print it in a messagebox.

```csharp
mtbProj.ExecuteCommand("text c1 c1");
object newval2 = "Green";
mtbColumn.SetData(ref newval2, 1, 1);
MessageBox.Show("The value is " + mtbColumn.GetData(1, 1).ToString());
```

Column method - SetValueOrder

Use to set the order in which text values are displayed in output. Column must be of type Text or DataUnassigned.

For more information on types of columns, go to MtbDataTypes on page 6.

Syntax

```csharp
SetValueOrder(ValueOrderType as MtbValueOrderTypes, UserDefinedOrder as Variant)
```

Arguments

- **ValueOrderType**
  - Required. Value order type for column. May be any MtbValueOrderTypes constant. For constants of MtbValueOrderTypes, go to MtbValueOrderTypes on page 7.

- **UserDefinedOrder**
  - Optional. Variant array specifying user defined value order. Required for MtbValueOrderTypes = 2, ignored otherwise.

Returns

HRESULT

Example

Create a new Minitab Application object, create and add text data to column 1, then set a user-defined value order for the column.

```csharp
Mtb.Application mtbApp = new Mtb.Application();

string[] arData = new string[6];
object[] arOrder = new object[3];

arData[0] = "a";
arData[1] = "a";
arData[2] = "b";
arData[3] = "b";
arData[4] = "c";
arData[5] = "c";
arOrder[0] = "c";
arOrder[1] = "a";
arOrder[2] = "b";

Mtb.Project mtbProj = mtbApp.ActiveProject;
mtbProj.ActiveWorksheet.Columns.Add(null, null, 3);
mtbProj.ActiveWorksheet.Columns.Item(1).SetData(arData);
arOrder);

Constants Collection object

The Constants collection is a set of all the Constant objects within a Worksheet object. It supports the standard
collection properties and methods.

The Constants collection for a worksheet is empty by default.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Number of Constant objects within the Constants collection.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Use to add Count Constant objects to the Constants collection in the position before Before or after After.</td>
</tr>
<tr>
<td>Delete</td>
<td>Use to remove all Constant objects from the Constants collection.</td>
</tr>
<tr>
<td>Item</td>
<td>Use to return a Constant object within the Constants collection.</td>
</tr>
<tr>
<td>Remove</td>
<td>Use to delete a Constant object and remove it from the Constants collection.</td>
</tr>
</tbody>
</table>

Example

Retrieve the Constants collection and add a constant, then name it "SalesFactor."

Mtb.Constants mtbConstants;
mtbConstants = mtbSheet.Constants;
mtbConstants.Add().Name = "SalesFactor";

Constants Collection property - Count

Description

Number of Constant objects within the Constants collection.

Type

Long

Range

0 - number of Constant objects in the Constants collection
Access
   Read-only

Example
Retrieve the Constants collection, add four Constant objects to it, then display the number of Constant objects in the Constants collection in a message box.
mtbConstants = mtbSheet.Constants;
mtbConstants.Add(null, null, 4);
MessageBox.Show("Number of constants in collection: " + mtbConstants.Count.ToString());

Constants Collection method - Add

Use to add Count Constant objects to the Constants collection in the position before Before or after After.

Syntax
Add(Before as Variant, After as Variant, Quantity as Long)

Arguments

Before
   Optional. Constant object to add new constants before.

After
   Optional. Constant object to add new constants after.

Quantity
   Optional. Number of constants to add. The default is 1.

Returns
   Constant on page 68

Remarks
You can specify either Before or After, but not both. Use an integer (Long) from 1 - the number of constants in the collection, or the name(String) of a constant. If neither Before nor After is specified, then the constants are added after the last constant in the collection. For more information on the Name property, go to Column property - Name on page 59.

The first constant added is returned.

Examples
Retrieve the Constants collection and add one constant to it after the last constant.
mtbConstants = mtbSheet.Constants;
mtbConstants.Add();

Add two constants to the Constants collection before the third constant, then name the first constant "Factor1."
mtbConstants.Add(3,null,2).Name = "Factor1";
Add two constants to the Constants collection after the third constant.
mtbConstants.Add(null, 3, 2);

Add four constants to the Constants collection before the “Factor1” constant, then name the first constant “NewFactor1.”
mtbConstants.Add("Factor1", null, 4).Name = "NewFactor1";

Add two constants to the Constants collection after the “Factor1” constant.
mtbConstants.Add(null, "Factor1", 2);

Constants Collection method - Delete

Use to remove all Constant objects from the Constants collection.

Syntax
Delete()

Returns
HRESULT

Remarks
To remove a single constant, use Remove or the Delete method of the Constant object.

For more information on the Remove method, go to Constants Collection method - Remove on page 67. For more information on the Delete method, go to Constant method - Delete on page 71. For more information on the Constant object, go to Constant object on page 68.

Example
Delete the Constants collection, including all its constants.
mtbConstants.Delete();

Constants Collection method - Item

Use to return a Constant object within the Constants collection.

Syntax
Item(Index as Variant)

Arguments

Index
Required. The index of the constant as an integer (Long) from 1 - the number of constants in the collection, or the name on page 71 (String) of the constant.
Returns

Constant on page 68

Examples

Retrieve the second constant in the Constants collection, name the constant "Conversion Factor", and print the name in a message box.

```vbnet
mtbConstant = mtbConstants.Item(2); 
mtbConstant.Name = "Conversion Factor";
MessageBox.Show("The second constant is " + mtbConstant.Name);
```

Retrieve the constant called "Metric" and print the name in a message box.

```vbnet
mtbConstant = mtbConstants.Item("Metric"); 
MessageBox.Show("The current constant is " + mtbConstant.Name);
```

Constants Collection method - Remove

Use to delete a Constant object and remove it from the Constants collection.

Syntax

Remove(Index as Variant)

Arguments

Index

Required. The index of the constant as an integer (Long) from 1 - the number of constants in the collection, or the name (String) of the constant.

For more information on name, go to Constant property - Name on page 71.

Returns

HRESULT

Remarks

The same results can be achieved using the Delete method of the Constant object. To remove all constants, use the Delete method of the Constants collection object.

For more information on the Delete method for the Constant object, go to page 71. For more information on the Constant object, go to page 68. For more information on the Delete Method of the Constants collection object, go to page 66. For more information on the Constants collection object, go to page 64.

Example

Remove the first constant and the constant named “Factor1” from the Constants collection.

```vbnet
mtbConstants.Remove(1); 
mtbConstants.Remove("Factor1");
```
The Constant object contains all the information related to an individual constant. The Constant object can contain numeric or text values.

**Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment</td>
<td>Description of the Constant object.</td>
</tr>
<tr>
<td>DataType</td>
<td>Type of the data in the Constant object.</td>
</tr>
<tr>
<td>Formula</td>
<td>Formula for the Constant object.</td>
</tr>
<tr>
<td>FormulaStatus</td>
<td>Status of the Formula property for the Constant object.</td>
</tr>
<tr>
<td>Name</td>
<td>Name of the Constant object</td>
</tr>
<tr>
<td>Number</td>
<td>Number of the Constant object</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete</td>
<td>Use to delete a Constant object and remove it from the Constants collection.</td>
</tr>
<tr>
<td>GetData</td>
<td>Use to return the value stored in the constant.</td>
</tr>
<tr>
<td>SetData</td>
<td>Use to set the value of the Constant object using the value in Data.</td>
</tr>
</tbody>
</table>

**Example**

Define the Constant object (mtbConstant) and retrieve the first constant in the Constants collection, set the value of the constant to 22.2, then print the constant in a message box.

```plaintext
MtbConstant mtbConstant;
mtbConstant = mtbSheet.Constants.Item(1);
mtbConstant.SetData(22.2);
MessageBox.Show(mtbConstant.GetData().ToString());
```

**Constant property - Comment**

**Description**

Description of the Constant object.

**Type**

String

**Range**

Valid string

**Access**

Read/Write
Example
Add a comment to the constant and print the comment in a message box.
```csharp
mtbConstant.Comment = "This constant converts English to metric.";
MessageBox.Show(mtbConstant.Comment);
```

Constant property - DataType

**Description**
Type of the data in the Constant object.

**Type**
- `MtbDataTypes` on page 6

**Range**
- Any `MtbDataTypes` constant except `DateTime`

**Access**
- Read-only

Example
Display the data type of the constant in a message box.
```csharp
MessageBox.Show("The data type of the constant is ", + mtbConstant.DataType);
```

Constant property - Formula

**Description**
Formula for the Constant object.

**Type**
- String

**Range**
- Valid string

**Access**
- Read-only

Example
Create two constants, K1 and K2. Set the value of K1 to 3 and create a formula that makes the value of K2 equal to K1 squared. Display a message box showing the values of both constants as well as the Formula and FormulaStatus properties of K2. Finally, change to manual formula calculation using the `CFMANUALLY` session command, change the value of K1, then display the same message. Notice that FormulaStatus changes from 1 to 2.
```csharp
Mtb.Application mtbApp = new Mtb.Application();
Mtb.Project mtbProj = mtbApp.ActiveProject;
Mtb.Worksheet mtbSheet = mtbProj.ActiveWorksheet;
Mtb.Constants mtbConstants = mtbSheet.Constants;

Mtb.Constant mtbConst1, mtbConst2;
mtbConst1 = mtbConstants.Add();
```
mtbConst2 = mtbConstants.Add();
mtbConst1.SetData(3);
mtbProj.ExecuteCommand("FORMULA K2 = K1**2");
MessageBox.Show("K1 = " + mtbConst1.GetData() + "\r\n" +
"K2 formula = " + mtbConst2.Formula + "\r\n" +
"K2 = " + mtbConst2.GetData() + "\r\n" +
mtbProj.ExecuteCommand("CFMANUALLY");
mtbConst1.SetData(5);
MessageBox.Show("K1 = " + mtbConst1.GetData() + "\r\n" +
"K2 formula = " + mtbConst2.Formula + "\r\n" +
"K2 = " + mtbConst2.GetData() + "\r\n" +

Constant property - FormulaStatus

**Description**
Status of the Formula property for the Constant object.

**Type**
MtbFormulaStatusTypes on page 6

**Range**
Any MtbFormulaStatusTypes constant

**Access**
Read-only

**Example**
Create two constants, K1 and K2. Set the value of K1 to 3 and create a formula that makes the value of K2 equal to K1 squared. Display a message box showing the values of both constants as well as the Formula and FormulaStatus properties of K2. Finally, change to manual formula calculation using the CFMANUALLY session command, change the value of K1, then display the same message. Notice that FormulaStatus changes from 1 to 2.

```csharp
Mtb.Application mtbApp = new Mtb.Application();
Mtb.Project mtbProj = mtbApp.ActiveProject;
Mtb.Worksheet mtbSheet = mtbProj.ActiveWorksheet;
Mtb.Constants mtbConstants = mtbSheet.Constants;
Mtb.Constant mtbConst1, mtbConst2;
mtbConst1 = mtbConstants.Add();
mtbConst2 = mtbConstants.Add();
mtbConst1.SetData(3);
mtbProj.ExecuteCommand("FORMULA K2 = K1**2");
MessageBox.Show("K1 = " + mtbConst1.GetData() + "\r\n" +
"K2 formula = " + mtbConst2.Formula + "\r\n" +
"K2 = " + mtbConst2.GetData() + "\r\n" +
mtbProj.ExecuteCommand("CFMANUALLY");
```
mtbConst1.SetData(5);
MessageBox.Show("K1 = " + mtbConst1.GetData() + "\r\n" + 
"K2 formula = " + mtbConst2.Formula + "\r\n" + 
"K2 = " + mtbConst2.GetData() + "\r\n" + 

Constant property - Name

**Description**
Name of the Constant object

**Type**
String

**Range**
Valid string

**Access**
Read/Write

**Example**
Retrieve the second constant in the Constants collection, name the constant "Factor2," and print the name in a message box.
mtbConstant = mtbConstants.Item(2);
mtbConstant.Name = "Factor2";
MessageBox.Show("The second constant is " + mtbConstant.Name);

Constant property - Number

**Description**
Number of the Constant object.

**Type**
Long

**Range**
1 - number of Constant objects in the Constants collection (current Minitab limit is 1000)

**Access**
Read-only

**Example**
Display in a message box the number of the Constant object within the Constants collection.
MessageBox.Show(mtbConstant.Number.ToString());

Constant method - Delete

Use to delete a Constant object and remove it from the Constants collection.
Syntax
Delete()

Returns
HRESULT

Remarks
The same results can be achieved using the Remove method of the Constants collection object. To delete all constants, use the Delete method of the Constants collection object.

For more information on the Remove method, go to page 64. For more information on the Delete method, go to page 66. For more information on the Constants collection object, go to page 64.

Example
Delete the Constant object from the Constants collection.
mtbConstant.Delete();

Constant method - GetData
Use to return the value stored in the constant.

Syntax
GetData()

Returns
Variant

Example
Print the value of the constant in a message box.
MessageBox.Show("The constant value is " + mtbConstant.GetData().ToString());

Constant method - SetData
Use to set the value of the Constant object using the value in Data.

Syntax
SetData(Data as Variant)

Arguments

Data
Required. Value to be stored in constant. Can be numeric or text.
Examples

Retrieve the first constant in the Constants collection, then set it equal to "Purple."

```csharp
mtbConstant = mtbConstants.Item(1);
mtbConstant.SetData("Purple");
```

Set the value of the current constant to 4.275, then print the value in the Immediate window.

```csharp
mtbConstant.SetData(4.275);
string writeValue = mtbConstant.GetData().ToString();
Console.WriteLine(writeValue);
```

Matrices Collection object

The Matrices collection is a set of all the Matrix objects within a Worksheet object. It supports the standard collection properties and methods.

The Matrices collection for a worksheet is empty by default.

## Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Number of Matrix objects within the Matrices collection.</td>
</tr>
</tbody>
</table>

## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Use to add Count Matrix objects to the Matrices collection in the position before Before or after After.</td>
</tr>
<tr>
<td>Delete</td>
<td>Use to remove all Matrix objects from the Matrices collection.</td>
</tr>
<tr>
<td>Item</td>
<td>Use to return a Matrix object within the Matrices collection.</td>
</tr>
<tr>
<td>Remove</td>
<td>Use to delete a Matrix object and remove it from the Matrices collection.</td>
</tr>
</tbody>
</table>

## Example

Retrieve the Matrices collection (mtbMatrices) and add a matrix, naming it “Weather Factors.”

```csharp
Mtb.Matrices mtbMatrices;
mtbMatrices = mtbSheet.Matrices;
mtbMatrices.Add().Name = "Weather Factors";
```

Matrices Collection property - Count

Description

Number of Matrix objects within the Matrices collection.
**Type**
Long

**Range**
0 - number of Matrix objects in the Matrices collection

**Access**
Read-only

**Example**
Retrieve the Matrices collection, add four Matrix objects to it, then display in a message box the number of Matrix objects in the Matrices collection.

```csharp
mtbMatrices = mtbSheet.Matrices;
mtbMatrices.Add(null, null, 4);
MessageBox.Show("Number of matrices in collection: " + mtbMatrices.Count.ToString());
```

**Matrices Collection method - Add**

Use to add Count Matrix objects to the Matrices collection in the position before Before or after After.

**Syntax**
`Add(Before as Variant, After as Variant, Quantity as Long)`

**Arguments**

*Before*
Optional. Matrix object to add new matrices before.

*After*
Optional. Matrix object to add new matrices after.

*Quantity*
Optional. Number of matrices to add. The default is 1.

**Returns**
*Matrix* on page 77

**Remarks**
You can specify either *Before* or *After*, but not both. Use an integer (*Long*) from 1 - the number of matrices in the collection, or the name on page 79 (*String*) of a matrix. If neither *Before* nor *After* is specified, then the matrices are added after the last matrix in the collection.

The first matrix added is returned.

For more information on name, go to *Matrix property - Name* on page 79.
Examples

Retrieve the Matrices collection and add one matrix to it after the last matrix.
```
mtbMatrices = mtbSheet.Matrices;
mtbMatrices.Add();
```

Add two matrices to the Matrices collection before the third matrix, and name the first added matrix "Gradient1."
```
mtbMatrices.Add(3,null,2).Name = "Gradient1";
```

Add two matrices to the Matrices collection after the third matrix.
```
mtbMatrices.Add(null,3,2);
```

Add four matrices to the Matrices collection before the "Gradient1" matrix, and name the first added matrix "NewGradient1."
```
mtbMatrices.Add("Gradient1", null, 4).Name = "NewGradient1";
```

Add two matrices to the Matrices collection after the "Gradient1" matrix.
```
mtbMatrices.Add(null,"Gradient1",2);
```

Matrices Collection method - Delete

Use to remove all Matrix objects from the Matrices collection.

Syntax
```
Delete()
```

Returns

HRESULT

Remarks

To remove a single matrix, use Remove or the Delete method of the Matrix object.

For more information on the Remove method, go to Matrices Collection method - Remove on page 76. For more information on the Delete method, go to Matrix method - Delete on page 80. For more information on the Matrix object, go to Matrix object on page 77.

Example

Delete the Matrices collection, including all its Matrix objects.
```
mtbMatrices.Delete();
```

Matrices Collection method - Item

Use to return a Matrix object within the Matrices collection.

Syntax
```
Item(Index as Variant)
```
Arguments

**Index**

Required. The index of the matrix as an integer (Long) from 1 - the number of matrices in the collection, or the name (String) of the matrix.

For more information on name, go to Matrix property - Name on page 79.

Returns

Matrix on page 77

Examples

Retrieve the second matrix in the Matrices collection, name the matrix "gradient," then print the name in a message box.

```vbnet
mtbMatrix = mtbMatrices.Item(2);
mtbMatrix.Name = "gradient";
MessageBox.Show("The second matrix is " + mtbMatrix.Name);
```

Retrieve the matrix called "Gradient" and print the name in a message box.

```vbnet
mtbMatrix = mtbMatrices.Item("Gradient");
MessageBox.Show("The current matrix is " + mtbMatrix.Name);
```

Matrices Collection method - Remove

Use to delete a Matrix object and remove it from the Matrices collection.

Syntax

Remove(Index as Variant)

Arguments

**Index**

Required. The index of the matrix as an integer (Long) from 1 - the number of matrices in the collection, or the name (String) of the matrix.

For more information on name, go to Matrix property - Name on page 79.

Returns

HRESULT

Remarks

The same results can be achieved using the Delete method of the Matrix object. To remove all matrices, use the Delete method of the Matrices collection object.

For more information on the Delete method of the Matrix object, go to Matrix method - Delete on page 80. For more information on the Matrix object, go to Matrix object on page 77. For more information on the Matrices Collection method, go to Matrices Collection method - Delete on page 75. For more information on the Matrices collection object, go to Matrices Collection object on page 73.
Example

Remove the first matrix and the matrix named “Gradient1” from the Matrices collection.
```csharp
mtbMatrices.Remove(1);
mtbMatrices.Remove("Gradient1");
```

Matrix object

The Matrix object contains all the information related to an individual matrix. The Matrix object can contain only numeric data values.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColumnCount</td>
<td>Number of columns in the Matrix object</td>
</tr>
<tr>
<td>Comment</td>
<td>Description of the Matrix object</td>
</tr>
<tr>
<td>MissingCount</td>
<td>Number of missing values in the Matrix object</td>
</tr>
<tr>
<td>Name</td>
<td>Name of the Matrix object</td>
</tr>
<tr>
<td>Number</td>
<td>Number of the Matrix object in the Matrices collection.</td>
</tr>
<tr>
<td>RowCount</td>
<td>Number of rows in the Matrix object</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete</td>
<td>Use to delete a Matrix object and remove it from the Matrices collection.</td>
</tr>
<tr>
<td>GetData</td>
<td>Use to return the matrix value or values in the specified row and column.</td>
</tr>
<tr>
<td>SetData</td>
<td>Use to set values for a Matrix object using the elements specified in Data.</td>
</tr>
</tbody>
</table>

Example

Define and populate the array “arrTemps” with data values, retrieve the first matrix, name the matrix “Temperatures” and place the information in arrTemps into the “Temperatures” matrix with 4 rows and 3 columns, then add a comment:
```csharp
object [] arrTemps = new object [12];
arrTemps[0] = 72;
arrTemps[1] = 95;
arrTemps[2] = 69;
arrTemps[3] = 87;
arrTemps[4] = 86;
arrTemps[5] = 75;
arrTemps[6] = 58;
arrTemps[7] = 92;
arrTemps[8] = 89;
arrTemps[9] = 66;
arrTemps[10] = 70;

Mtb.Matrix mtbMatrix;
mtbMatrix = mtbMatrices.Item(1);
mtbMatrix.Name = "Temperatures";
```
mtbMatrix.SetData(ref arrTemps, 4, 3);
mtbMatrix.Comment = "Temperatures for experiment 1";

Matrix property - ColumnCount

Description
Number of columns in the Matrix object

Type
Long

Range
1 - N

Access
Read-only

Example
Display in a message box the number of columns in the Matrix object.
MessageBox.Show("This matrix has this many columns: " + mtbMatrix.ColumnCount.ToString());

Matrix property - Comment

Description
Description of the Matrix object

Type
String

Range
Valid string

Access
Read/Write

Example
Add a comment to the matrix.
mtbMatrix.Comment = "Temperature gradient values.";

Matrix property - MissingCount

Description
Number of missing values in the Matrix object

Type
Long
Range
N/A

Access
Read-only

Example
Display in a message box the number of missing values in the Matrix object.
MessageBox.Show("This matrix has this many missing values: " + mtbMatrix.MissingCount.ToString());

Matrix property - Name

Description
Name of the Matrix object

Type
String

Range
Valid string

Access
Read/Write

Example
Retrieve the second matrix in the Matrices collection, name the matrix "Gradient 2," then print the name in a message box:
mtbMatrix = mtbMatrices.Item(2);
mtbMatrix.Name = "Gradient 2";
MessageBox.Show("The second matrix is " + mtbMatrix.Name);

Matrix property - Number

Description
Number of the Matrix object in the Matrices collection.

Type
Long

Range
1 - number of the Matrix objects in the Matrices collection (current Minitab limit is 1000)

Access
Read-only
Example
Display in a message box the number of the Matrix object within the Matrices collection.
MessageBox.Show("This is matrix number: " + mtbMatrix.Number.ToString());

Matrix property - RowCount

Description
Number of rows in the Matrix object

Type
Long

Range
1 - N

Access
Read-only

Example
Display in a message box the number of rows in the Matrix object.
MessageBox.Show("This matrix has this many rows: " + mtbMatrix.RowCount.ToString());

Matrix method - Delete

Use to delete a Matrix object and remove it from the Matrices collection.

Syntax
Delete()

Returns
HRESULT

Remarks
The same results can be achieved using the Remove method of the Matrices collection object. To delete all matrices, use the Delete method of the Matrices Collection object.

For more information on the Remove method on the Matrices collection object, go to Matrices Collection method - Remove on page 76. For more information on the Delete method, go to Matrices Collection method - Delete on page 75. For more information on the Matrices collection object, go to Matrices Collection object on page 73.

Example
Delete the Matrix object from the Matrix collection.
mtbMatrix.Delete();
Matrix method - GetData

Use to return the matrix value or values in the specified row and column.

Syntax

GetData(Row as Long, Col as Long)

Arguments

Row
- Optional. Row of the value to get.

Col
- Optional. Column of the value to get.

Returns

Variant

Remarks

If you specify Row you must specify Col, and vice versa. If neither Row nor Col is specified, then GetData gets the entire matrix.

Multiple data values are returned in a vector in column major order, that is, all rows of column 1 are placed in the vector first, followed by column 2 rows, column 3 rows, etc.

Example

Get the value in the second row, second column of the matrix and print it in a message box.
MessageBox.Show("The value is " + mtbMatrix.GetData(2, 2));

Matrix method - SetData

Use to set values for a Matrix object using the elements specified in Data.

Syntax

SetData(Data as Variant, Rows as Long, Cols as Long)

Arguments

Data
- Required. Numeric value or values to set in the matrix.

Rows
- Required. Either the row of the matrix where a single value is to be set, or the number of rows to be set, starting at row 1.
**Cols**

Required. Either the column of the matrix where a single value is to be set, or the number of columns to be set, starting at column 1.

**Returns**

HRESULT

**Remarks**

If `Data` holds an individual numeric value, then `Rows` and `Cols` refer to an individual cell in the matrix that will be set. If `Data` holds multiple numeric values, then all previous data in the matrix is deleted and the matrix is set starting at position 1,1.

`Data` is read into the `Matrix` object column by column. Therefore, to set multiple data values in the matrix, `Data` must be a vector in column major order; that is, place all rows of column 1 in the vector first, followed by column 2 rows, column 3 rows, etc.

**Examples**

Retrieve the first matrix in the `Matrices` collection and populate four rows and three columns in the matrix with the array "arrIndex."

```csharp
Mtb.Matrix mtbMatrix = mtbMatrices.Item(1);

int [] arrIndex1 = new int [12];
arrIndex1[0] = 72;
arrIndex1[1] = 95;
arrIndex1[2] = 69;
arrIndex1[3] = 87;
arrIndex1[4] = 86;
arrIndex1[5] = 75;
arrIndex1[6] = 58;
arrIndex1[7] = 92;
arrIndex1[8] = 89;
arrIndex1[9] = 66;
arrIndex1[10] = 70;
arrIndex1[11] = 91;

object arrIndex = arrIndex1;
int rows = 4;
int cols = 3;
mtbMatrix.SetData(ref arrIndex, rows, cols);
```

Place the value 10 in row 2, column 2 of the current matrix.

```csharp
object newval = 10;
mtbMatrix.SetData(ref newval, 2, 2);
```
C Command Object Reference

Commands Collection object

The Commands collection contains the commands that have been issued to Minitab during the session.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count on page 84</td>
<td>Number of Command objects.</td>
</tr>
<tr>
<td>OutputDocument on page 84</td>
<td>Returns an OutputDocument object containing all output generated from all commands in the Commands collection.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete on page 84</td>
<td>Use to remove all Command objects from the Commands collection.</td>
</tr>
<tr>
<td>Item on page 85</td>
<td>Use to return a Command object within the Commands collection.</td>
</tr>
<tr>
<td>Remove on page 85</td>
<td>Use to delete a Command object and remove it from the Commands collection.</td>
</tr>
</tbody>
</table>

Example

Create a Minitab Application object and execute three Minitab commands. Then delete the first command, and loop through the remaining two, displaying a message box with the values of the following properties for each, as well as the name of the worksheet:

- CommandLanguage
- Name
- Tag
- CreatedBy
- CreateDate

Save the output document for each command and delete all commands at the end.

```csharp
Mtb.Application mtbApp = new Mtb.Application();
Mtb.Project mtbProj;
Mtb.Command mtbCom;

mtbApp.UserInterface.Visible = true;
mtbProj = mtbApp.ActiveProject;

mtbProj.ExecuteCommand("RAND 30 C1-C2");
mtbProj.ExecuteCommand("REGRESS C1 1 C2");

//For the next command, use the ZTAG subcommand to set the Tag property.
mtbProj.ExecuteCommand("CORR C1 C2; ZTAG \"My Correlation\"");
mtbProj.Commands.Item(1).Delete();
```
for (int i = 1; i <= mtbProj.Commands.Count; i++)
{
    mtbCom = mtbProj.Commands.Item(i);
    MessageBox.Show("CommandLanguage = " + mtbCom.CommandLanguage + "\nCommand Name = " + mtbCom.Name + "\nTag = " + mtbCom.Tag + "\nCreated by " + mtbCom.CreatedBy + "\nCreated on " + mtbCom.CreateDate + "\nWorksheet = " + mtbCom.Worksheet.Name);
    mtbCom.OutputDocument.SaveAs("C:\Output for Command " + i, true, Mtb.MtbOutputFileTypes.OFRTF);
}
mtbProj.Commands.Delete();

Commands Collection property - Count

Description
- Number of Command objects.

Type
- Long

Range
- Any valid long integer

Access
- Read-only

Commands Collection property - OutputDocument

Description
- Returns an OutputDocument object containing all output generated from all commands in the Commands collection.

Type
- OutputDocument on page 97

Range
- N/A

Access
- Read-only

Commands Collection method - Delete

Use to remove all Command objects from the Commands collection.

Syntax
- Delete()
Returns
HRESULT

Remarks
To remove a single command, use `Remove` or the `Delete` method of the `Command` object.

For more information on the `Remove` method, go to `Commands Collection method - Remove` on page 85. For more information on the `Delete` method, go to `Command method - Delete` on page 89. For more information on the `Command` object, go to `Command object` on page 86.

**Commands Collection method - Item**

Use to return a `Command` object within the `Commands` collection.

Syntax
Item(Index as Variant)

Arguments

Index
Required. The index of the command as an integer (`Long`) from 1 - the number of commands in the collection.

Returns
`Command` on page 86

**Commands Collection method - Remove**

Use to delete a `Command` object and remove it from the `Commands` collection.

Syntax
Remove(Index as Variant)

Arguments

Index
Required. The index of the command as an integer (`Long`) from 1 - the number of commands in the collection.

Returns
HRESULT

Remarks
The same results can be achieved using the `Delete` method of the `Command` object. To remove all commands, use the `Delete` method of the `Commands` collection object.
For more information on the Delete method of the Command object, go to Command method - Delete on page 89. For more information on the Command object, go to Command object on page 86. For more information on the Delete method of the commands collection object, go to Commands Collection method - Delete on page 84. For more information on the Commands collection object, go to Commands Collection object on page 83.

Example
Delete the first command from the Commands collection for a Minitab application set as mtbApp.

```csharp
mtbApp.ActiveProject.Commands.Remove(1);
```

Command object

Command objects are created when you execute a Minitab command either programatically or directly in Minitab.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CommandLanguage</td>
<td>Command language utilized to create the command object.</td>
</tr>
<tr>
<td>CreateDate</td>
<td>The date and time the command was generated.</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the Minitab command that generated the output.</td>
</tr>
<tr>
<td>OutputDocument</td>
<td>Returns an OutputDocument object containing all output generated by the command.</td>
</tr>
<tr>
<td>Outputs</td>
<td>Returns the Outputs collection generated by the command.</td>
</tr>
<tr>
<td>Tag</td>
<td>A string used to identify or describe the Command object.</td>
</tr>
<tr>
<td>Worksheet</td>
<td>The worksheet that was utilized as the input for the command.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete</td>
<td>Use to delete a Command object and remove it from the Commands collection.</td>
</tr>
</tbody>
</table>

Example
Create a Minitab Application object and execute three Minitab commands. Then delete the first command, and loop through the remaining two, displaying a message box with the values of the following properties for each, as well as the name of the worksheet:

- CommandLanguage
- Name
- Tag
- CreatedBy
- CreateDate

Save the output document for each command and delete all commands at the end.

```csharp
Mtb.Application mtbApp = new Mtb.Application();
Mtb.Project mtbProj;
```
Mtb.Command mtbCom;

mtbApp.UserInterface.Visible = true;
mtbProj = mtbApp.ActiveProject;

mtbProj.ExecuteCommand("RAND 30 C1-C2");
mtbProj.ExecuteCommand("REGRESS C1 1 C2");

//For the next command, use the ZTAG subcommand to set the Tag property.
mtbProj.ExecuteCommand("CORR C1 C2; ZTAG "My Correlation".");

mtbProj.Commands.Item(1).Delete();

for (int i = 1; i <= mtbProj.Commands.Count; i++)
{
    mtbCom = mtbProj.Commands.Item(i);
    MessageBox.Show("CommandLanguage = " + mtbCom.CommandLanguage + 
    "\nCommand Name = " + mtbCom.Name + "\nTag = " + mtbCom.Tag + "\nCreated by " + mtbCom.CreatedBy + "\nCreated on " + mtbCom.CreateDate + "\nWorksheet = " + mtbCom.Worksheet.Name);
    mtbCom.OutputDocument.SaveAs("C:\Output for Command "+ i, true, Mtb.MtbOutputFileTypes.OFRTF);
}

mtbProj.Commands.Delete();

Command property - CommandLanguage

Description
Command language utilized to create the command object.

Type
String

Range
Valid string

Access
Read-only

If the command is a custom command, the value of the CommandLanguage property is "COMCUSTOM."

Command property - CreateDate

Description
The date and time the command was generated.

Type
String

Range
Valid string
Access
- Read-only

Command property - Name

Description
- The name of the Minitab command that generated the output.

Type
- String

Range
- Valid string

Access
- Read-only

Command property - OutputDocument

Description
- Returns an OutputDocument object containing all output generated by the command.

Type
- OutputDocument on page 97

Range
- N/A

Access
- Read-only

Command property - Outputs

Description
- Returns the Outputs collection generated by the command.

Type
- Outputs on page 90 collection

Range
- N/A

Access
- Read-only
Command property - Tag

**Description**
A string used to identify or describe the Command object. Null by default.

**Type**
String

**Range**
Valid string

**Access**
Read/Write

The Tag property for most commands can also be set from the Minitab itself using the ZTAG subcommand. For example, entering the following in the Command Line pane creates a Command object with the tag "My Z-Test." The argument for ZTAG may be a string or text constant.

```
OneZ 20 3;
Sigma 1;
Test 2;
ZTAG "My Z-Test".
```

Command property - Worksheet

**Description**
The worksheet that was utilized as the input for the command.

**Type**
Worksheet on page 44

**Range**
Any worksheet in the Worksheets collection

**Access**
Read-only for Minitab commands; Read/Write for custom commands

Command method - Delete

Use to delete a Command object and remove it from the Commands collection.

**Syntax**
Delete()

**Returns**
HRESULT

**Remarks**
The same results can be achieved using the Remove method of the Commands collection.
For more information on the `Remove` method, go to `Commands Collection method - Remove` on page 85. For more information on the `Commands` collection, go to `Commands Collection object` on page 83.

### Outputs Collection object

The `Outputs` collection for each `Command` object contains all the output generated by that command.

For more information on command objects, go to `Command object` on page 86.

#### Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Count</code></td>
<td>The number of <code>Output</code> objects within the <code>Outputs</code> collection. For more information, go to <code>Output object</code> on page 92.</td>
</tr>
</tbody>
</table>

#### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Delete</code></td>
<td>Use to remove all <code>Output</code> objects from the <code>Outputs</code> collection.</td>
</tr>
<tr>
<td><code>Item</code></td>
<td>Use to return an <code>Output</code> object within the <code>Outputs</code> collection.</td>
</tr>
<tr>
<td><code>Remove</code></td>
<td>Use to delete an <code>Output</code> object and remove it from the <code>Outputs</code> collection.</td>
</tr>
</tbody>
</table>

#### Example

Generate random data, then create a scatterplot and a regression analysis. Save the scatterplot to a file. Delete `Output` objects using the `Remove` method of the `Outputs` collection and the `Delete` method of an `Output` object. Finally, delete all outputs for a command at once using the `Delete` method of the `Outputs` collection.

```csharp
Mtb.Application mtbApp = new Mtb.Application();
Mtb.Project mtbProj;

mtbApp.UserInterface.Visible = true;
mtbProj = mtbApp.ActiveProject;

mtbProj.ExecuteCommand("RAND 30 C1-C2");
mtbProj.ExecuteCommand("PLOT C1*C2");
mtbProj.ExecuteCommand("REGRESS C1 1 C2");

//Save the plot to a file
mtbProj.Commands.Item(2).Outputs.Item(1).Graph.SaveAs("C:\MyGraph", true, Mtb.MtbGraphFileTypes.GFPNGColor);

Mtb.Command mtbCom = mtbProj.Commands.Item(3);

//Save the output document as an HTML file
mtbCom.OutputDocument.SaveAs("C:\MyOutput", true, Mtb.MtbOutputFileTypes.OFHTML);

//Delete the first 2 Output objects and save the output document again
mtbCom.Outputs.Remove(1);
mtbCom.Outputs.Item(1).Delete();
mtbCom.OutputDocument.SaveAs("C:\MyOutput2", true, Mtb.MtbOutputFileTypes.OFHTML);

//Delete all remaining Outputs
mtbCom.Outputs.Delete();
```
Outputs Collection property - Count

**Description**
The number of Output objects within the Outputs collection. For more information, go to Output object on page 92.

**Type**
Long

**Range**
0 to the number of Output objects within the Outputs collection

**Access**
Read-only

Outputs Collection method - Delete

*Use to remove all Output objects from the Outputs collection.*

**Syntax**
Delete()

**Returns**
HRESULT

**Remarks**
To remove a single Output object, use Remove or the Delete method of the Output object.

For more information on the Remove method, go to Outputs Collection method - Remove on page 92. For more information on the Delete method, go to Output method - Delete on page 96. For more information on the Output object, go to Output object on page 92.

Outputs Collection method - Item

*Use to return an Output object within the Outputs collection.*

**Syntax**
Item(Index as Variant)

**Arguments**

**Index**
Required. The index of the Output object as an integer (Long) from 1 - the number of Output objects in the collection.
Returns

Output on page 92

Outputs Collection method - Remove

Use to delete an Output object and remove it from the Outputs collection.

Syntax

Remove(Index as Variant)

Arguments

Index

Required. The index of the output as an integer (Long) from 1 - the number of outputs in the collection.

Returns

HRESULT

Remarks

The same results can be achieved using the Delete method of the Output object. To remove all Output objects, use the Delete method of the Outputs collection.

For more information on the Delete method of the Output object, go to Output method - Delete on page 96. For more information on the Output object, go to Output object on page 92. For more information on the Delete method of the Outputs collection, go to Constants Collection method - Delete on page 66. For more information on the Outputs collection, go to Outputs Collection object on page 90.

Output object

Each Output object contains one component of the output from a Minitab Command object.

The OutputType and Tag properties are universal and apply to all Output objects. Each of the remaining properties are valid only for one specific output type. For example, using the Formula property on an Output object of type OTFormula, returns a Formula object. Using the Formula property on any other MtbOutputTypes returns an error.

For more information on a Minitab command object, go to Command object on page 86. For more information on OTFormula, go to MtbOutputTypes on page 7. For more information on MtbOutputTypes, go to MtbOutputTypes on page 7.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph on page 94</td>
<td>If the Output object is a graph, this property returns the corresponding Graph object.</td>
</tr>
<tr>
<td>HTMLText on page 95</td>
<td>The contents of the Output object as HTML-formatted text.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OutputType</td>
<td>The Output object type.</td>
</tr>
<tr>
<td>RTFText</td>
<td>The contents of the Output object as RTF-formatted text.</td>
</tr>
<tr>
<td>Tag</td>
<td>A string used to identify or describe the Output object.</td>
</tr>
<tr>
<td>Text</td>
<td>The contents of the Output object as un-formatted text.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete</td>
<td>Use to delete an Output object and remove it from the Outputs collection.</td>
</tr>
</tbody>
</table>

### Examples

Create a Minitab Application object and execute several Minitab commands. Then loop through all Output objects in the Outputs collection for each Command, using the OutputType property to identify each output type. Display a message for each output object stating the Index number of the Command, as well as the Index number, OutputType, and the Text of the Output object. Finally, save all output for all commands as an HTM file, using the OutputDocument object of the Commands collection.

```csharp
Mtb.Application mtbApp = new Mtb.Application();
Mtb.Project mtbProj;
Mtb.Command mtbCmnd;
Mtb.Output mtbOutObj;
string msgStr;

mtbApp.UserInterface.Visible = true;
mtbProj = mtbApp.ActiveProject;

//Execute some Minitab commands
mtbProj.ExecuteCommand("RAND 30 C1-C3");
mtbProj.ExecuteCommand("PLOT C1*C2");
mtbProj.ExecuteCommand("REGRESS C1 1 C2");
mtbProj.ExecuteCommand("CORR C1 C2 C3 C4");
/*This CORR command will generate a correlation analysis and also an error message because there is no data in C4.*/

//Add a worksheet and create a DOE design
mtbProj.Worksheets.Add();
mtbProj.ExecuteCommand("FFDESIGN 4 8; CTPT c3; RANDOMIZE; SORDER C1 C2; BRIEF 4;" + "ALIAS 4; XMATRIX C5 C6 C7 C8."");

//Loop through outputs from commands, identify type of each, and display message for (int i = 1; i <= mtbProj.Commands.Count; i++)
{
    mtbCmnd = mtbProj.Commands.Item(i);
    for (int j = 1; j <= mtbCmnd.Outputs.Count; j++)
    {
        mtbOutObj = mtbCmnd.Outputs.Item(j);
        msgStr = "Command #" + i + ", " + "Output #" + j + " is OutputType ";
        int caseSwitch = mtbOutObj.OutputType.GetHashCode();
        switch (caseSwitch)
        {
            case 0: //Graph
                MessageBox.Show(msgStr + "Graph.");
                break;
```
Create a new instance of Minitab, generate two columns of random data, and run a correlation analysis. Display the Text, RTFText, and HTMLText for each output of the analysis in a message box.

```csharp
Mtb.Application mtbApp = new Mtb.Application();
Mtb.Project mtbProj;
Mtb.Outputs mtbOuts;
Mtb.Output mtbOut;

mtbApp.UserInterface.Visible = true;
mtbProj = mtbApp.ActiveProject;

mtbProj.ExecuteCommand("RAND 30 C1-C2");
mtbProj.ExecuteCommand("CORRELATION C1 C2");
mtbOuts = mtbProj.Commands.Item(2).Outputs;

for (int i = 1; i <= mtbOuts.Count; i++)
{
    mtbOut = mtbOuts.Item(i);
    MessageBox.Show("Text for Output " + i + ":\r\n\n" + mtbOut.Text);
    MessageBox.Show("RTFText for Output " + i + ":\r\n\n" + mtbOut.RTFText);
    MessageBox.Show("HTMLText for Output " + i + ":\r\n\n" + mtbOut.HTMLText);
}
```

**Output property - Graph**

**Description**

If the Output object is a graph, this property returns the corresponding Graph object. Otherwise, an error is generated.

**Type**

Graph on page 99

**Range**

N/A

**Access**

Read-only
Output property - HTMLText

**Description**
The contents of the Output object as HTML-formatted text.

**Type**
String

**Range**
Valid string

**Access**
Read-only

Output property - OutputType

**Description**
The Output object type.

**Type**
`MtbOutputTypes` on page 7

**Range**
Any `MtbOutputTypes` constant

**Access**
Read-only

Output property - RTFText

**Description**
The contents of the Output object as RTF-formatted text.

**Type**
String

**Range**
Valid string

**Access**
Read-only

Output property - Tag

**Description**
A string used to identify or describe the Output object. Null by default.

**Type**
String
**Range**
Valid string

**Access**
Read/Write

**Example**
Set the Tag text for the first Output object in mtbCommand to "This is Output number 1," then display the tag in a message box.
```csharp
mtbCommand.Outputs.Item(1).Tag = "This is Output number 1";
MessageBox.Show(mtbCommand.Outputs.Item(1).Tag);
```

**Output property - Text**

**Description**
The contents of the Output object as un-formatted text.

**Type**
String

**Range**
Valid string

**Access**
Read-only

**Output method - Delete**
Use to delete an Output object and remove it from the Outputs collection.

**Syntax**
Delete()

**Returns**
HRESULT

**Example**
Delete the first Output object from the command set as mtbCommand.
```csharp
mtbCommand.Outputs(1).Delete
```

**Remarks**
The same results can be achieved using the Remove method of the Outputs collection.

For more information on the Remove method, go to Outputs Collection method - Remove on page 92. For more information on the Outputs collection, go to Outputs Collection object on page 90.
OutputDocument object

An **OutputDocument** object contains all output generated by a single **Command** object or by all commands in the **Commands** collection.

For more information on the command object, go to **Command object** on page 86. For more information on the commands collection, go to **Commands Collection object** on page 83.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HTMLText</strong> on page 97</td>
<td>Content of <strong>OutputDocument</strong> in HTML format.</td>
</tr>
<tr>
<td><strong>RTFText</strong> on page 98</td>
<td>Content of <strong>OutputDocument</strong> in RTF format.</td>
</tr>
<tr>
<td><strong>Text</strong> on page 98</td>
<td>Content of <strong>OutputDocument</strong> as plain text.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CopyToClipboard</strong> on page 98</td>
<td>Use to copy the <strong>OutputDocument</strong> object to the Windows clipboard.</td>
</tr>
<tr>
<td><strong>SaveAs</strong> on page 99</td>
<td>Use to save a copy of the <strong>OutputDocument</strong> object.</td>
</tr>
</tbody>
</table>

Example

Create a Minitab Application object and execute some Minitab commands. Save the **OutputDocument** for the Commands collection and copy it to the Windows clipboard. Save the content of the **OutputDocument** in string variables as text, HTML formatted text, and RTF formatted text.

```csharp
Mtb.Application mtbApp = new Mtb.Application();
Mtb.Project mtbProj;
string sText, sHTML, sRTF;

mtbApp.UserInterface.Visible = true;
mtbProj = mtbApp.ActiveProject;

mtbProj.ExecuteCommand("RAND 30 C1");
mtbProj.ExecuteCommand("DESCRIBE C1");


mtbOutDoc.SaveAs("C:\\MyOutputDocument", true,
Mtb.MtbOutputFileTypes.OFRTF);
mtbOutDoc.CopyToClipboard();
sText = mtbOutDoc.Text;
sHTML = mtbOutDoc.HTMLText;
sRTF = mtbOutDoc.RTFText;
```

OutputDocument property - HTMLText

**Description**

Content of **OutputDocument** in HTML format.
OutputDocument property - RTFText

Description
Content of OutputDocument in RTF format.

OutputDocument property - Text

Description
Content of OutputDocument as plain text.

OutputDocument method - CopyToClipboard

Use to copy the OutputDocument object to the Windows clipboard.

Syntax
CopyToClipboard()

Returns
HRESULT
OutputDocument method - SaveAs

Use to save a copy of the OutputDocument object.

Syntax

SaveAs(Filename as String, Replace as Boolean, OutputFileType as MtbOutputFileTypes)

Arguments

Filename
Required. Path and file name to use when saving the file. If a path is not specified, then the DefaultFilePath is used. For more information on the DefaultFilePath, go to ApplicationOptions property - DefaultFilePath on page 28.

Replace
Optional. If True, an existing file with the same name will be overwritten. The default is False.

OutputFileType
Optional. The format to use when saving the file. May be any MtbOutputFileTypes constant. For more information on MtbOutputFileTypes, go to MtbOutputFileTypes on page 7

Returns

HRESULT

Remarks

If you don’t specify an extension matching the file type, the appropriate one (.HTM or .RTF) is automatically added to the file name.

Graph object

Each Graph object contains a single graph generated by a Minitab Command object.

For more information on the Minitab command object, go to Command object on page 86.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CopyToClipboard on page 100</td>
<td>Use to copy the Graph object to the Windows clipboard.</td>
</tr>
<tr>
<td>SaveAs on page 100</td>
<td>Use to save a copy of the Graph object.</td>
</tr>
</tbody>
</table>

Example

Create a new Minitab Application object, execute a command that generates a Graph object as Output object number 1, retrieve the graph, and then save it as a color PNG image file. Finally, copy the graph to the system clipboard.

Mtb.Application mtbApp = new Mtb.Application();
Mtb.Project mtbProj;
Mtb.Graph mtbGraph;
mtbApp.UserInterface.Visible = true;
mtbProj = mtbApp.ActiveProject;
mtbProj.ExecuteCommand("RAND 30 C1");
mtbProj.ExecuteCommand("HISTOGRAM C1");
mtbGraph = mtbProj.Commands.Item(2).Outputs.Item(1).Graph;
mtbGraph.SaveAs("C:\MyGraph", true, Mtb.MtbGraphFileTypes.GFPNGColor);
mtbGraph.CopyToClipboard();

Graph method - CopyToClipboard
Use to copy the Graph object to the Windows clipboard.

Syntax
CopyToClipboard()

Returns
HRESULT

Remarks
You can copy a Minitab Graph object, Bitmap, MetaFile, or Enhanced metafile. For Minitab Graph objects, either a paste or a paste link operation is allowed with the copied object. For all other file formats, only paste is allowed.

Graph method - SaveAs
Use to save a copy of the Graph object.

Syntax
SaveAs(Filename as String, Replace as Boolean, GraphFileType as MtbGraphFileTypes, Width as Long, Height as Long)

Arguments

Filename
Optional. Path and file name to use when saving the graph. If a path is not specified, then the DefaultFilePath is used. The default file name is Minitab. For more information on the DefaultFilePath property - DefaultFilePath on page 28.

Replace
Optional. If True, an existing file with the same name will be overwritten. The default is True.

GraphFileType
Optional. The format to use when saving the file. May be any MtbGraphFileTypes constant. The default is .PNG. For more information on the MtbGraphFileTypes, go to MtbGraphFileTypes on page 6.
**Width**
Optional. Use to set the width of the graph in pixels.

**Height**
Optional. Use to set the height of the graph in pixels.

Returns
HRESULT

Remarks
If you don't specify an extension matching the file type, the appropriate one (JPG, PNG, TIF, BMP, GIF, or EMF) is automatically added to the file name.