RAYLASE
focus on laser

Reference Manual
weldMARK™ COM Automation Server

This manual has been compiled by RAYLASE for its customers and employees.

RAYLASE reserves the right to change the information contained in this manual without prior notice.

LIST OF CONTENT

1 INTRODUCTION

Thank you for purchasing the RAYLASE AG weldMARK™ 3 marking software suite. The following information will assist you in properly installing the software in your computer and configuring your software to communicate with the Automation server that weldMARK™ exposes.

1.1 About this Manual

The weldMARK™ Automation Server Interface Manual contains detailed information about interfacing to the COM Automation server provided by RAYLASE AG, and is meant to be a reference tool. This manual assumes you have a working knowledge of the COM specification and programming languages compatible with COM objects.

1.2 Technical Support

If you are experiencing problems installing this package and you need help, you should:

- Retry the action, carefully following the instructions given for that task in this guide.
- Try to determine the nature of the problem. By eliminating variables, the problem can be narrowed down. If it appears to be hardware problems, check the documentation that came with your hardware for maintenance or hardware-related issues. Contact your hardware representative if necessary.
- Contact RAYLASE AG Customer Service department for additional technical support.

1.3 Manufacturer

RAYLASE AG
Argelsrieder Feld 2+4
82234 Wessling
Deutschland
Tel.: +49 (0) 81 53 - 88 98 - 0
Fax: +49 (0) 81 53 - 88 98 - 10
http://www.raylase.de
E-Mail: info@raylase.de

2 COM-INTERFACE – BACKGROUND

This chapter gives a brief overview of the Microsoft COM specification, and the implementation of the RAYLASE AG Automation object.

2.1 Overview of COM

COM is the Component Object Model, an object-based programming specification designed by Microsoft to provide robust object interoperability through sets of predefined routines called interfaces. COM is based on a binary standard, rather than a source code standard, thereby enabling objects written in different languages, running in different process spaces and on different platforms to communicate. COM objects can also be transparently extended, modified and updated because unique identifiers are used to create them and to access their interfaces. COM also has a library containing a set of standard interfaces that define the core functionality of a COM object, and a small set of API functions designed for the purpose of creating and managing COM objects.

As extensible systems software architecture, COM is the basis for other technologies such as OLE and ActiveX. These technologies are operating system extensions that define their own rules and provide their own libraries for creating and manipulating objects of those types. Using COM as a foundation, developers can create their own extensions so that objects created according to their rules can interact with other COM-based technologies.

2.2 COM Objects

A COM object is an object that is instantiated from a CoClass, which is a class that implements one or more interfaces. The COM object provides the services indicated by each interface its CoClass supports. Any time a COM object is used it is referenced by a pointer to one of its interfaces. This establishes two important features of a COM object, which are:

- With access only through function pointers, no external manipulation of a COM object can directly modify its data.
- Because an interface reference is a pointer, any language, with any internal state representation, can use COM objects as long as that language can create pointers to structures, or arrays, of function pointers.

A CoClass must have a class factory and a class identifier (CLSID) so that its COM object can be externally instantiated (from another module). Using these unique identifiers for CoClasses means that they can be transparently updated whenever new interfaces are implemented in their class. Because interfaces are also accessed by unique identifiers rather than by names, CoClasses can support both old and new versions of an interface (as a collection of methods and implementations). The new interface can modify or add methods without a conflict of versions, which is a common problem when using DLLs. Moreover, interface pointers are polymorphic, allowing any kind of interface pointer to manipulate any kind of COM object. With COM, even objects built by different vendors at different times can interact without conflict.

2.3 COM and ActiveX Servers

A COM server (like the RAYLASE AG Automation object) is an application or a library that provides services to a client application or library. A COM server can be an in-process server, meaning a DLL running in the same process space as the client, a local server, meaning an EXE running in a different process space but on the same machine as the client, or a remote server, meaning an application running on a different machine from that of the client. COM servers are the modules in which COM objects exist. A COM server that contains the code for automation objects and ActiveX controls is an ActiveX server. The RAYLASE AG Automation object is implemented as a local or Automation server, and is identified by the filename wmCOM.exe.

2.4 Automation

Automation refers to the ability of an application to control the objects in another application, programmatically. The client of an Automation object is referred to as an Automation controller and the server object being manipulated is called the Automation object. Automation can be used on in-process, local, and remote servers.

Automation is characterized by two key points:

- The Automation object must be able to define a set of properties and commands, and to describe their capabilities through type descriptions. In order to do this they must have a way to provide information about the object's interfaces, the interface methods, and those methods' arguments. Typically this information is available in type libraries. The type library for the RAYLASE AG Automation object, wmCOM.tlb, is included on the distribution CD.
- Automation objects must make these methods accessible so that other applications can use them. For this they must implement the IDispatch interface. Through this interface an object can expose all of its methods and properties. Through the primary method of this interface, the object's methods can be invoked, once having been identified through type information.

Developers wanting to create and use non-visual OLE objects that can run in any process space can use Automation. One of the reasons for this is that Automation is a mechanism that provides an automatic way to allow cross-process applications to communicate by implementing the IDispatch interface, which automates the marshaling process. Automation does, however, restrict the types that you can use. For a list of allowed types, refer to your compiler documentation.

2.5 Writing the Automation Controller

There are several different ways you can access the methods of an Automation object from a remote program. One method enables you to access the methods via Variants, and a second lets you access them via something called a dispinterface. A simpler approach is to use the Type Library, wmCOM.tlb, available after a weldMARK™ installation.

What is a Type Library?

If the only people who used COM objects were C++ programmers, one could pass around a header file in order to define the proper way to access an interface. However, COM needs to be available to a wide range of programmers using a diverse set of tools.

Fortunately, the declarations for COM objects can be stored in a type library. Type libraries contain code that can be called to describe the structure of a particular object, including its names, methods, the parameters passed to the methods, and some of the types used by the object.

In short, a type library is just a header file packaged so multiple languages can use it. A type library defines the types found in a particular interface or set of interfaces. Each language – VB, Object Pascal, C++, and so on – can open a type library, read its contents, and generate code or other symbols of use to programmers accessing that object from a particular language or tool.

Please refer to your compiler documentation for details on the importation and use of type libraries.

3 REQUIREMENTS AND INSTALLATION

This chapter gives a set by step description on installing the weldMARK™ Automation object software on your computer.

3.1 System Requirements

You need the following hardware:

- Intel Pentium Computer running Microsoft Windows 2000 Service Pack 3 or higher up to Microsoft Windows 7 Professional. To determine the version of the operating system and the amount of memory installed on your computer, on the desktop right-click My Computer, then select Properties.
- CD-ROM drive for software installation.
- 1 GB RAM minimum.
- 100 MB of local disk space available.

3.2 Hardware Dongle Installation

The weldMARK™ Automation object requires a hardware dongle. A dongle is a security device attached to a valid USB port. If you do not have a hardware dongle, please consult RAYLASE.

3.3 Software Installation

The weldMARK™ Automation object files are included in the standard installation of weldMARK™ 3. The executable **wmCOM.exe** is placed in the “bin”-folder of the weldMARK™ install directory, and the Type Library **wmCOM.tlb** is placed in the “activex”-folder of the weldMARK install directory.

3.4 Registering the weldMARK™ Automation Server Object

Before accessing the weldMARK™ Automation object, it must be registered on the local machine. The installation program automatically registers the COM server object when the program installs. If for some reason you need to register a COM server object, the following is included for reference:

To register an out-of-process server

- Run the server with the **/regserver** command-line option.
- You can also register the server by running it.

To unregister an out-of-process server:

- Run the server with the **/unregserver** command-line option.

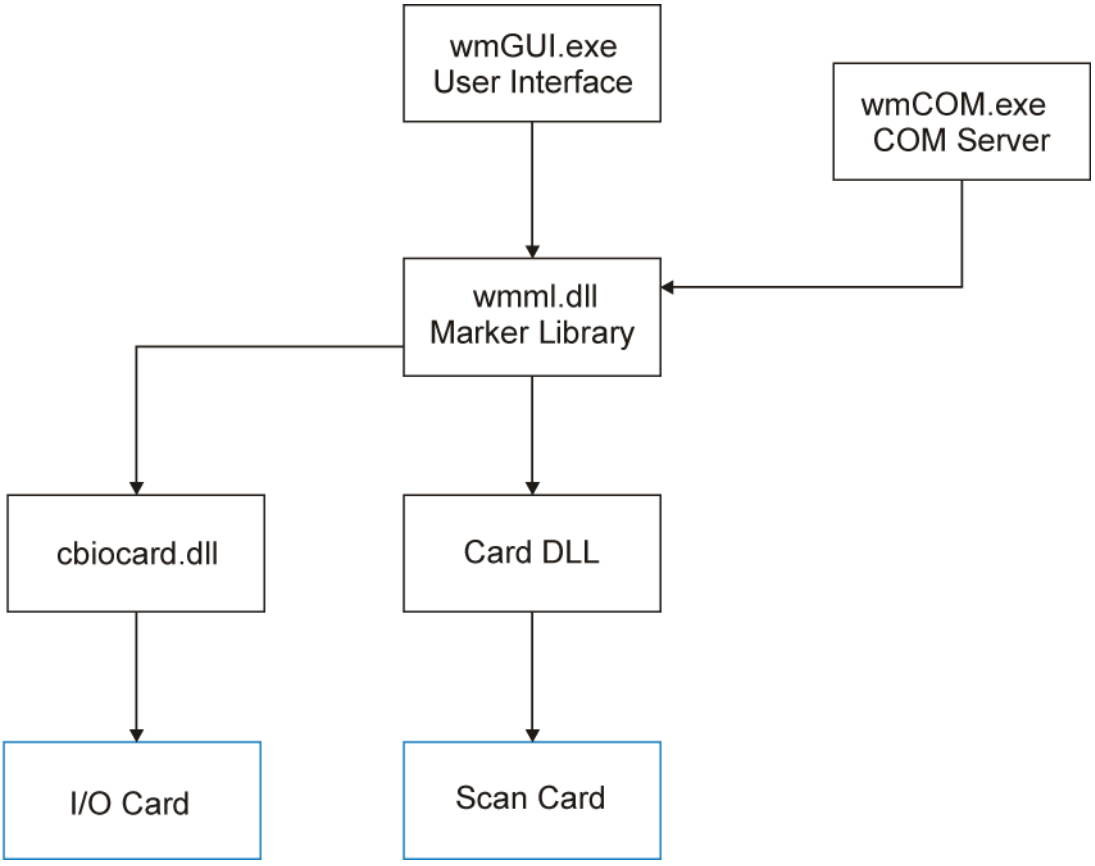
In addition, many IDEs allow the registering of COM objects through their user interface. For example, Visual Basic supports this function through the Add References dialog box.

4 WELDMARK™® AUTOMATION OBJECT MODEL CONCEPTS

This chapter gives an overview of the COM component structure, the Automation object design, and offers some tips on interfacing with the COM component.

4.1 Overview

The RAYLASE AG Automation object component (CoClass) is a piece of binary code packaged in the executable file wmMARK.exe (*weldMARK*). The name of the CoClass contained in this component is *Automate*, and therefore the ProgID is *weldMARK.Automate*. The COM interfaces provided by *weldMARK* are IUnknown and IAutomate, and are the only means of getting access to the functionality of the COM component. IAutomate is essentially a wrapper around the RAYLASE AG Marker Library, which provides all the services needed to interact with the laser marker. The following diagram illustrates the relationship between the various software components.



The COM Automation server is meant for advanced integration applications, and as such, does not support:

- Text object special processing – The “Source” property of the text object is ignored.
- Barcode object special processing – The “Source” property of the barcode object is ignored.
- weldMARK™ Automation objects – The object will load from a job, but calling MarkObject has no effect.
- Mark in Progress – The Mark in Progress port will not automatically toggle when the system is marking. The programmer must explicitly set the state of the port with *SetScanCardOutput*.
- External Start – The External Start port settings in the job are ignored, and the port is not automatically checked. The programmer must explicitly check the port with *GetScanCardInput*.

When running jobs from the weldMARK™ GUI environment, features such as step and repeat, serialization, external start, etc. are handled for the user by the GUI. Since the weldMARK™ GUI is not used when interfacing with the COM server, the programmer must code these features themselves. Access to the I/O card ports is exposed by the COM server as a convenience to the programmer. All other control, such as Motor Control, needs to be accomplished by the programmer directly with the third party libraries.

The *weldMARK*™ component is an out-of-process server (or local server), which is an application (wmCOM.exe) running in a different process space but on the same machine as the client. For example, an Excel spreadsheet embedded in a Word document are two separate applications running on the same machine. The local server uses COM to communicate with the client.

Additionally, *Automate* has a **dual interface**, which is a custom (VTable) interface and a dispinterface at the same time. It is implemented as a COM VTable interface that derives from IDispatch. For those controllers that can access the object only at runtime, such as VBScript and JScript, the dispinterface is available. For controllers that can take advantage of compile-time binding, the more efficient VTable interface can be used.

Dual interfaces offer the following combined advantages of VTable interfaces and dispinterfaces:

- For Automation controllers that cannot obtain type information, the dispinterface provides runtime access to the object.
- For in-process servers, you have the benefit of fast access through VTable interfaces.
- For out-of-process servers, COM marshals data for both VTable interfaces and dispinterfaces. COM provides a generic proxy/stub implementation that can marshal the interface based on the information contained in a type library.

4.2 Creating the COM Object

Before you can control the weldMARK™ Automation object library from your client application, you must obtain a pointer to an interface it supports. Typically, you connect to a server through its main interface (IAutomate in this case).

Depending on the development environment you are using, you obtain a pointer to the interface differently. Some IDEs provide the means to import the Type Library (wmCOM.tlb) provided. After importing, the interface becomes available through the IDE itself. Others controllers, such as VBScript, only allow late binding (at run time).

Consult your development environment help files for documentation on using COM objects.

4.2.1 C++ Example

The following code can be used to create the COM object in a C++ program. The functions CoInitialize() and CoCreateInstance() are COM library calls supported in Windows operating system. A successful installation of weldMARK™ and the ActiveX directory is required to access the IAutomate pointer, the symbolic constants CLSID_Automate and IID_IAutomate.

```
// Initialize Windows COM libraries
::CoInitialize(NULL);

// Create an interface pointer
IAutomate* pMarker=NULL;
::CoCreateInstance(CLSID_Automate,
     NULL,
     CLSCTX_LOCAL_SERVER,
     IID_IAutomate,
     reinterpret_cast<void**>(&pMarker));

// Code to use the COM object goes here
pMarker->AttachToMarker();

etc…
// Unload the Windows COM libraries, we are finished with the object
::CoUninitialize();
```

4.2.2 Borland C++ Builder 5.0 Example

```
// This method creates the COM object, and keeps it loaded until the
// application closes
TCOMIAutomate Marker;
HRESULT hr=CoAutomate::Create(Marker);
if (FAILED(hr))
     {
     Application->MessageBox("Cannot start or locate the weldMARK COM
Automation server.","", MB_OK);
     Application->Terminate();
     }
```

4.2.3 Visual C++ 6.0 Example

You can use the Visual Studio ClassWizard to wrap the elements of the weldMARK™ type library in an MFC C++ class and add the new class to a project. Your project must be created as an MFC application in order to allow Class Wizard to generate the wrapper class. You should also check the **Automation** and **ActiveX Controls** support options during project creation in order to generate some of the necessary OLE initialization code.

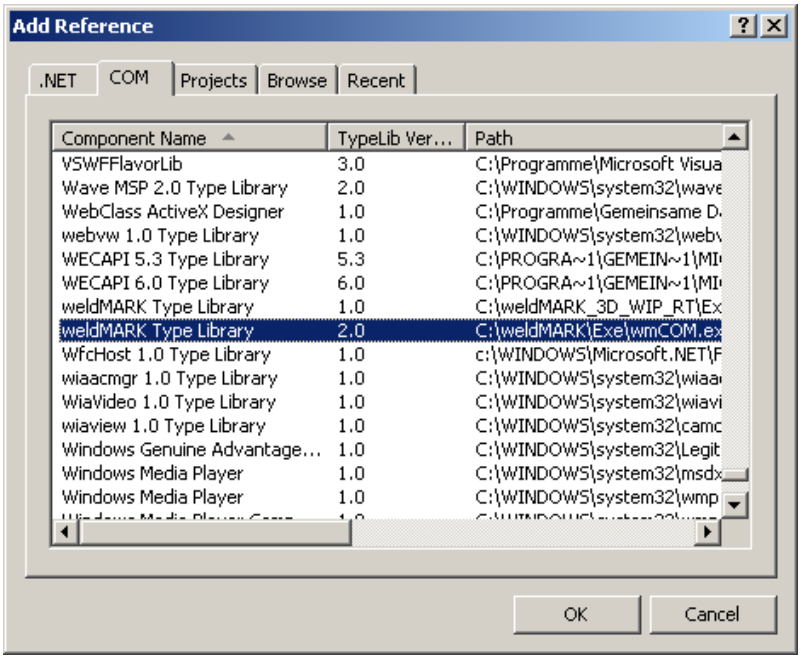
To import the elements of the weldMARK™ type library

- On the View menu, click **ClassWizard**. ClassWizard will appear.
- Click the **Add Class** button, then click **From a type library…** from the drop down list. The **Import from Type Library** dialog box appears.
- Select the wmCOM.tlb type library from the programme\raylase\weldmark\activex directory and click Open. The **Confirm Classes** dialog box appears. This dialog box contains a list of classes that ClassWizard can create from information in the type library. The class names are generated by ClassWizard.
- Optional: Use the **Name** text box to rename the class that is currently selected from the list.
- Optional: Use the **Header File** and **Implementation File** text boxes to rename the .h and .cpp files, if you choose to. Also, you can use the **Browse** buttons to rename the files or cause the files to be generated in a different directory.
- All classes selected from the class list are added to these two files.
- Click **OK**. ClassWizard generates the class and adds the .h and .cpp files to your project.

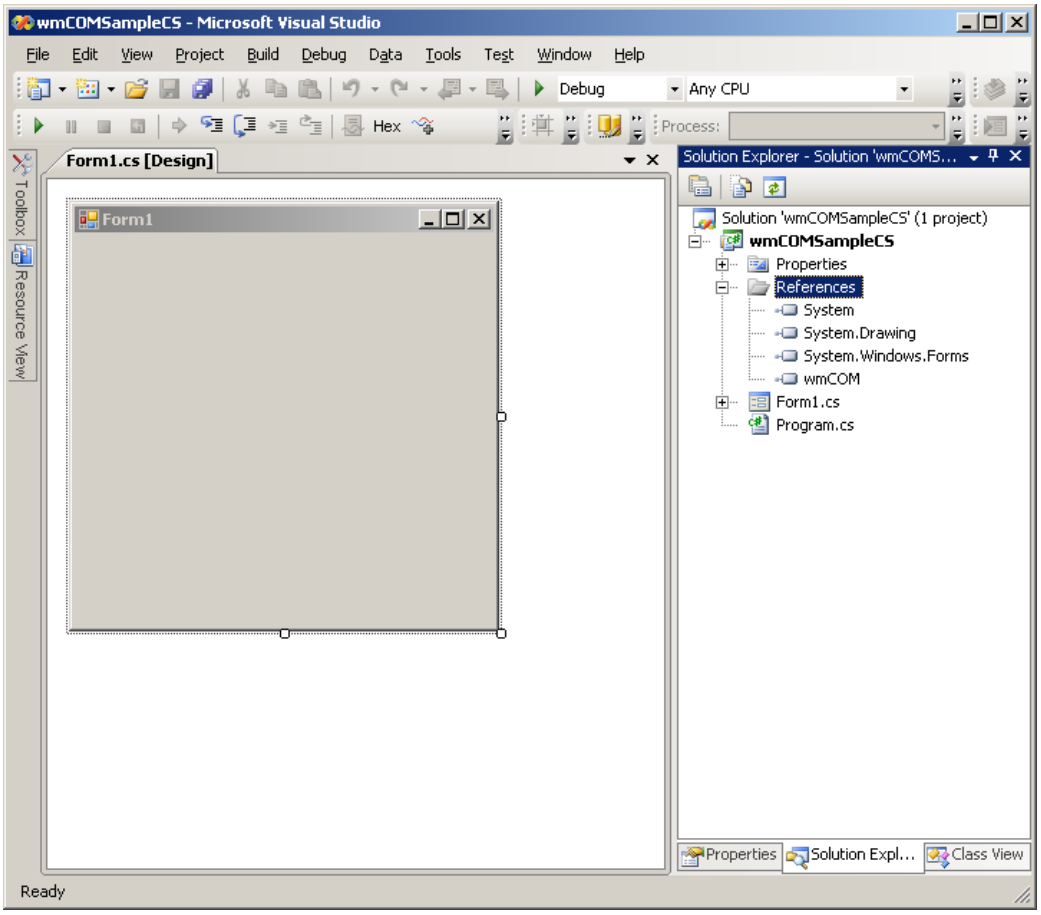
4.2.4 Example for C#.NET

Create a new Visual C# Project in the Visual Studio IDE.

Add a new weldMARK-COM-Server reference. Accessible via the „COM“ tab, select „weldMARK Type Library“ and click <OK>.



The component „wmCOM“ will be visible in the references of the „Solution Explorer“ now.



reate a wmCOM.Automate type object in you application and initialize it as following:

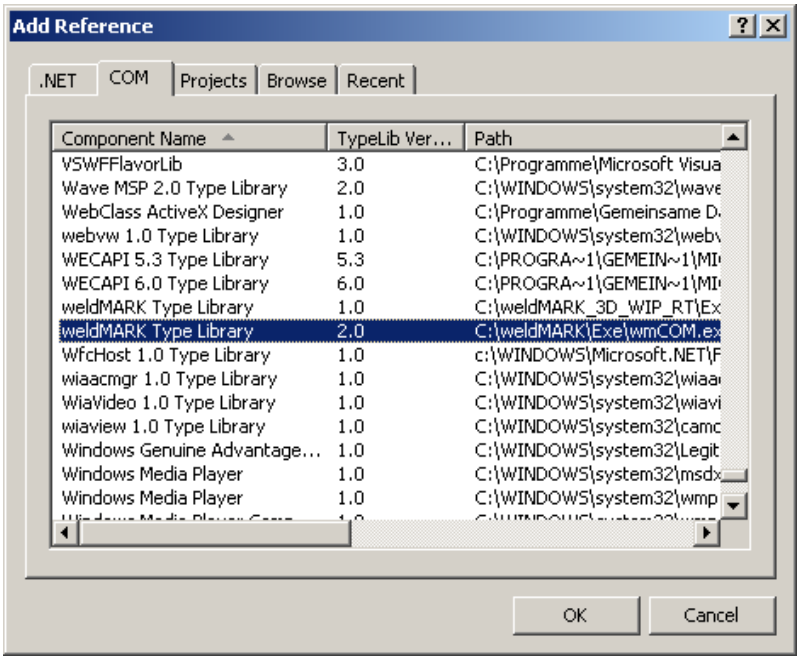
```
using System;
using System.Windows.Forms;


namespace wmCOMSampleCS

   public partial class Form1 : Form
   {
      private wmCOM.Automate wmCOMObj = null;

      public Form1()
      {
         InitializeComponent();
      }

      private void Form1_Load(object sender, EventArgs e)
      {
         wmCOMObj = new wmCOM.Automate();
         try
         {
            wmCOMObj.AttachToMarker();

            int nAvailableCards = 0;
            wmCOMObj.GetScanCardCount(out nAvailableCards);
            if (nAvailableCards > 0)
            {
               // add code here to start the actual work of your application ..
            }
            else
            {
               MessageBox.Show (
                                  "COM-Server did not find any controller
ard!",
                                  "Application Error",
                                  MessageBoxButtons.OK,
                                  MessageBoxIcon.Error
                                );
            }
         }
         catch (Exception ex)
         {
            MessageBox.Show (
                                 "COM-Server Initialization failed ..\r\n\r\n" +
x.Message,
                                 "COM-Server Error",
                                 MessageBoxButtons.OK,
                                 MessageBoxIcon.Error
                               );
         }
      }

      private void Form1_FormClosing(object sender, FormClosingEventArgs e)
      {
         try
         {
            wmCOMObj.ReleaseMarker();
         }
         catch
         {
         }
      }
   }
}
```

The function ReleaseMarker() should occur upon ending the application [Event FormClosing()] at the latest, as shown in the example.

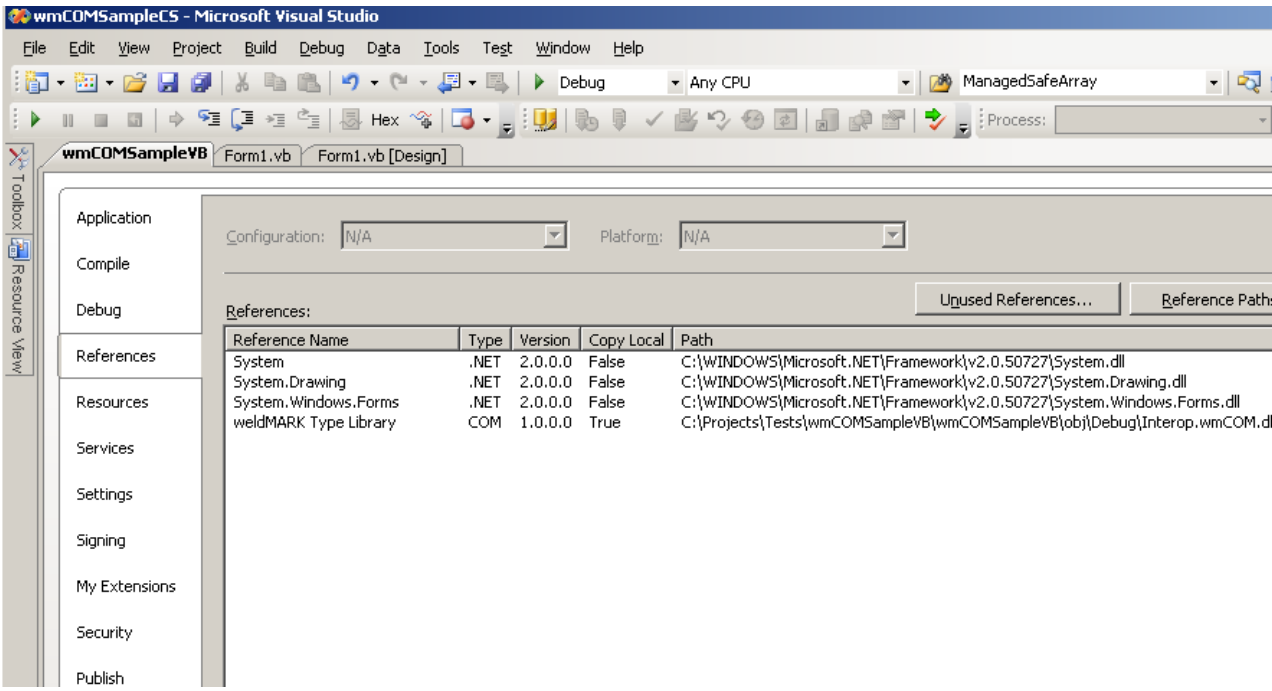
4.2.5 Example for VB.NET

Create a new Visual Basic. NET Project in Visual Studio IDE.

Add a new weldMARK-COM-Server reference in the “COM” tab. It is found under "weldMARK Type Library" after clicking <OK>.



The component „wmCOM“ will be visible now in the project properties under references.



Create a new wmCOM.Automate object in you application and initialize it as follows:

```
Public Class Form1
   Private wmCOMObj As wmCOM.Automate = Nothing

   Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
ystem.EventArgs) _ Handles MyBase.Load
      wmCOMObj = New wmCOM.Automate

      Try

         wmCOMObj.AttachToMarker()

         Dim nAvailableCards As Integer = 0
         wmCOMObj.GetScanCardCount(nAvailableCards)

         If (nAvailableCards > 0) Then
            ' add code here to start the actual work of your application ..
         Else
            MessageBox.Show( _
             "COM-Server did not find any controller card!", _
             "Application Error", _
             MessageBoxButtons.OK, _
             MessageBoxIcon.Error _
            )
         End If

      Catch ex As Exception

         MessageBox.Show( _
          "COM-Server Initialization failed .." + vbCrLf + vbCrLf +
x.Message, _
          "COM-Server Error", _
          MessageBoxButtons.OK, _
          MessageBoxIcon.Error _
         )

      End Try
   End Sub

   Private Sub Form1_FormClosing(ByVal sender As System.Object, ByVal e As _
System.Windows.Forms.FormClosingEventArgs) Handles MyBase.FormClosing
      Try
         wmCOMObj.ReleaseMarker()

      Catch

      End Try
   End Sub
End Class
```

The function ReleaseMarker() should occur upon ending the application [Event FormClosing()] at the latest, as shown in the example.

Dealing with COM-Server Error messages in .NET languages is fairly easy.
Here is an example:

```
Private Sub btnPrepareJob_Click( _
ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
tnPrepare.Click

   Dim nObjectsInJob As Integer = 0

   Try
```

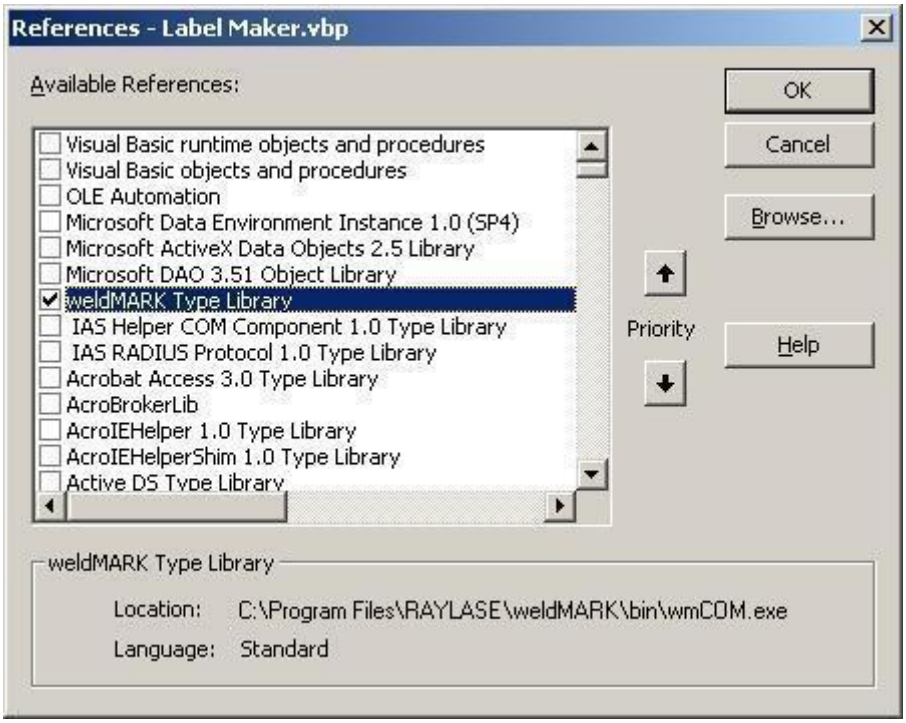
```
        wmCOMObj.LoadJobFromFile("A1.wmj", iJobID)
        wmCOMObj.GetObjCount(nObjectsInJob)
        wmCOMObj.SetObjPos(nObjectsInJob, 100, 100)
        lblTestInfo.Text = String.Format("Objects in Job = {0}", nObjectsInJob)
    Catch ex As Exception
        MessageBox.Show("Prepare Job Error:" + vbCrLf + vbCrLf + ex.Message)
    End Try
End Sub
```

- Under the assumption that a job file named A1.wmj with at least one object exists, this function generates the error message #5 because the function-SetObjPos () contains a value that is to a high for parameter #1 by one. See also 4.5 and 4.4.2!

4.2.6 Visual Basic 6.0 Example

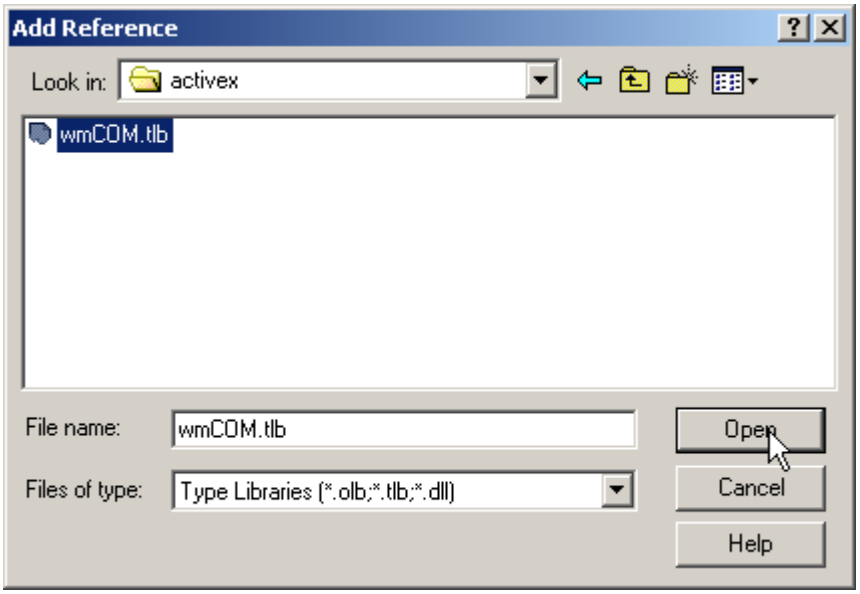
The Visual Basic IDE is informed about the Automation object type information at compile time by using the Project -> References menu item. To add the weldMARK™ Automation object reference to your Visual Basic project, use the following steps:

- In the Visual Basic IDE, from the main menu, click Project -> References.
- The References dialog appears.



- In the Available References list, locate the weldMARK™ Type Library item and check the box.

- If the weldMARK™ Type Library item is not in the Available References list, click on the Browse button. The Add Reference dialog appears.



- Locate the type library wmCOM.tlb included with the weldMARK™ 3 install. This file is usually located in *programme**raylase**weldmark**activex*. Select the file and click open.
- The weldMARK™ Type Library is added to the list of Available references.
- Check the box indicating you want to add the reference.
- Click OK to close the dialog box.

Now that the weldMARK™ Automation object is available to Visual Basic, add the following declaration at module global scope:

Dim Marker As New weldMARK.Automate

In this code, *weldMARK.Automate* is the ProgID of the CoClass, *New* creates an object, and the variable *Marker* is assigned to the default interface of the object, in this case IAutomate.

You can now call methods on the Marker object such as:
Marker.AttachToMarker

4.2.7 Example for VBScript

To write an example VBScript:

- Start Windows Notepad, or any simple text editor.
- Enter the following lines exactly as shown:

```
' weldMARK.vbs
' This is a demo program to show basic interaction with the COM Serv-
er

MsgBox "weldMARK COM Server Taskbar Icon Demo"+(Chr(13) &
Chr(10))+"Copyright © 2002 Alase Technologies"+(Chr(13) &
Chr(10))+(Chr(13) & Chr(10))+"Click OK to load the COM Server."

' Create the COM object and get a reference to it
Dim marker
Set marker= CreateObject("WeldMARK.Automate")

MsgBox "The weldMARK COM Server is loaded and the taskbar notifica-
tion icon is visible in the bottom right corner of the taskbar (blue
gears)."+(Chr(13) & Chr(10))+(Chr(13) & Chr(10))+"Place your mouse
over the icon and to see the ToolTip. The icon is protected by de-
fault; the right click menu is not available."+(Chr(13) &
Chr(10))+(Chr(13) & Chr(10))+"Click OK to continue"

marker.ShowTrayIcon 0,1
MsgBox "Taskbar notification icon is invisible."+(Chr(13) &
Chr(10))+(Chr(13) & Chr(10))+"Click OK to conti-nue"

marker.ShowTrayIcon 1,0
MsgBox "Taskbar notification icon is visible and un-
protected."+(Chr(13) & Chr(10))+(Chr(13) & Chr(10))+"Place your mouse
over the icon and right click with the mouse to see the context
menu."+(Chr(13) & Chr(10))+"Do not select Terminate Automation Server
at this time."+(Chr(13) & Chr(10))+(Chr(13) & Chr(10))+"Click OK to
continue"

MsgBox "End of demo."+(Chr(13) & Chr(10))+(Chr(13) & Chr(10))+"Click
OK to unload the weldMARK COM Server"
```

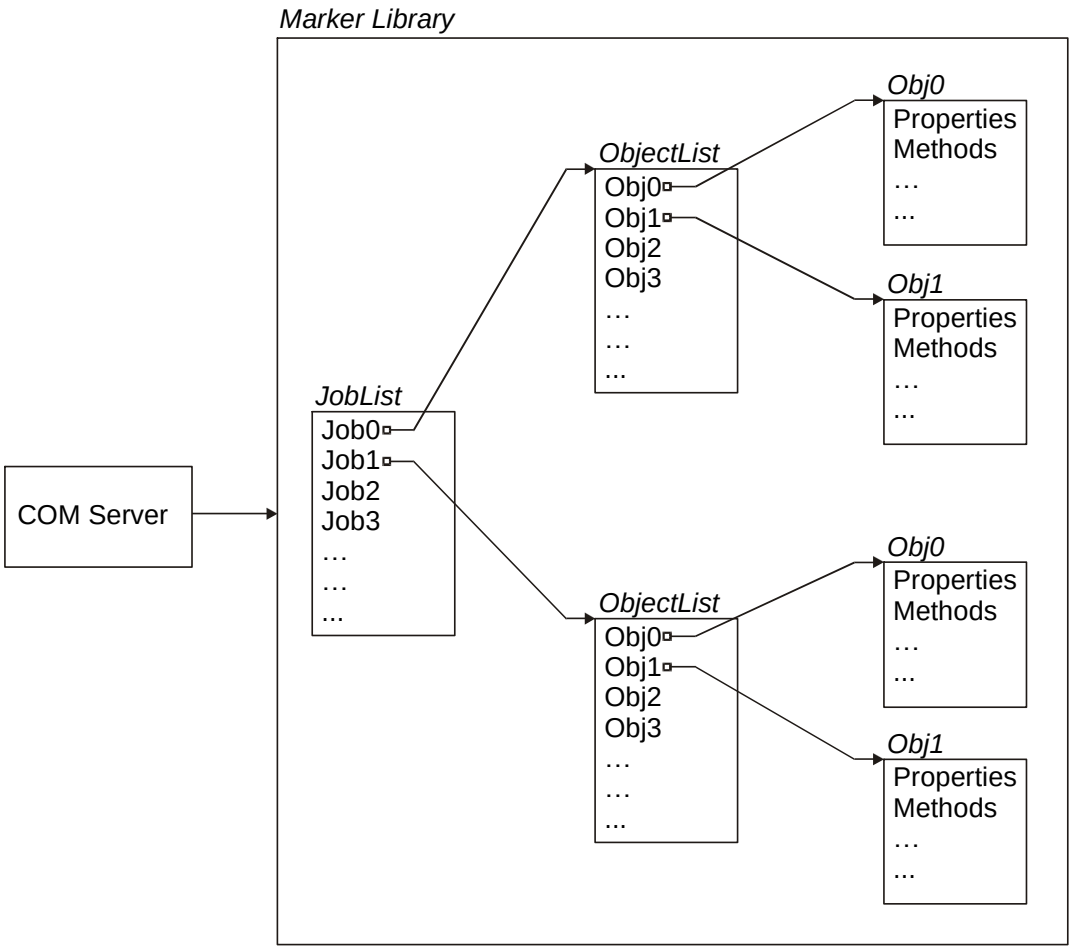
- When you are done editing the file, save it as *weldMARK.vbs*.
- Browse to the file in Windows Explorer, right click on the file with the mouse, and select **Open**.
- The weldMARK™ Automation object will load in memory, indicated by the icon in the system tray, and the message boxes will appear.

This simple example demonstrates the potential for controlling weldMARK™ from script files.

4.3 Using the Marker Library

4.3.1 Overview

The architecture of the underlying Marker Library is object oriented by design. JobObjects and MarkObjects are created in memory, and then maintained in their own lists. The following diagram illustrates the Marker Library structure:



Following the C/C++ convention, job- and objectlists are 0 based. So, for example, if the MarkObject list contains 5 (five) objects, the first has an Index value of 0 (zero), the next is 1 (one), and so forth.

4.3.2 Initializing the Marker Library

After an interface pointer has been obtained to Automate, the COM Object initializes all its resources and loads the Marker Library. After initialization, the client application must check to see if there are any scan head cards in the PC by calling *GetScanCardCount*. A result of 0 indicates no card present. The client application should gracefully exit at that time, showing an appropriate error message.

Because weldMARK™ (the GUI) and the weldMARK™ Automation object cannot be running at the same time, it may be necessary to detach from the Marker Library to allow the GUI to attach to it. To detach from the Marker Library, you call *ReleaseLibrary*. To reattach, you call *AttachToLibrary*. When your application first starts up, there is no need to call *AttachToLibrary*, as this is done automatically for you.

The client application may not want users to see or have access to the Notification area icon, which represents the weldMARK™ Automation object when it is running. To change the behavior of the Notification area icon, call *ShowTrayIcon*, passing the appropriate flags.

4.3.3 Working with JobObjects

You create JobObjects by calling *NewJob*, passing the name you want to give your new job. This call passes back the Index value of the new JobObject. The Index value also becomes the current ActiveJob. **All calls made to the Marker Library referencing a JobObject or MarkObject use the current ActiveJob index**. You can change the ActiveJob index by calling *SetActiveJob* and pass the new Index value. This Index value represents the position of the JobObject in its list. To find out how many JobObjects are in the list, call *GetJobCount*. The Marker Library supports a maximum of 10 JobObjects loaded simultaneously. You can delete a JobObject from memory by calling *CloseJob*.

4.3.4 Working with MarkObjects

MarkObjects are added to their corresponding JobObject parent, so set the ActiveJob to the desired Index. To add a MarkObject, you call one of the mark object creation functions such as NewBitmap, NewRect, etc. The Index of the new object is passed back when the call completes. MarkObjects are created in memory with a default location and size, and default Profile. MarkObject positions and sizes can be manipulated by calling functions such as CenterObj, SetObjToRect, etc. Calling the corresponding SetXXXAttributes sets specific properties of the objects themselves. Currently, there are 18 different MarkObject types. You can discover an object's type by calling *GetObjType* passing it's Index. A MarkObjects marking parameters (Profile) are set with a call to *SetObjProfile.* To find out how many MarkObjects are in the list, call *GetObjCount*. You can delete a MarkObject from memory by calling *DeleteObj*. The ScanCardNum is a property of the MarkObject, and when a MarkObject is first created, it's ScanCardNum is set to 0 (zero). If there are multiple scan cards in the PC, MarkObjects can be assigned to each card with a call to *SetScanCardNum.*

4.3.5 Working with the Standard I/O card

An interface to the Standard I/O card is provided by the COMServer to allow the programmer access to the various input and output ports on the card. These ports are directly controlled by the programmer, and are not set automatically by the software. For example, when using the weldMARK™ user interface, both the Mark In Progress and the Busy/Ready outputs toggle automatically at the appropriate times during the execution of a job. When using the COM-Server, however, these outputs are not set automatically, and must be set using the corresponding commands. In this way, the programmer has complete control over when these ports change states during the execution of their marking tasks.

4.4 Error Handling

Because the ability to get hold of rich error information from a COM object is important, Microsoft introduced *error objects*. Error objects are now the preferred method of receiving rich error information back from a COM object. The methods used to access the error object differ between programming languages.

4.4.1 Visual Basic Error Client

Because Visual Basic cannot use the HRESULT returned from a call to the COM server, another technique must be used to extract useful error information. Fortunately, Visual Basic supports the Err object, which can be used for this purpose. In order to detect all possible errors that are returned from the COM Server, any code that calls the server should be wrapped in the On Error GoTo/Error Handler routine, illustrated in the following code snippet:

```
Private Sub MarkButton_Click()

On Error GoTo ErrorHandler
Marker.MarkObj0,0
Other code…
…
…
ErrorHandler:
MsgBox Err.Description, vbOKOnly, "weldMARK.Automate error"

End Sub
```

The line OnError GoTo ErrorHandler catches any errors that are returned from the Marker.MarkObj(0,0) function call, and program execution is directed to ErrorHandler. ErrorHandler uses the Err object, and extracts its Description data member to obtain a description of the error.

4.4.2 C++ Error Client

In accordance with the COM specifications, each call to the COM server will return an HRESULT value. HRESULTs are used to return rich error information to a C++ client. Although most non-0 HRESULT values indicate an error condition, this is not always the case. Therefore, it is usually not acceptable to test for a non-0 condition to indicate an error. It is recommended to use the FAILED macro in the following way:

```
HRESULT hr=Marker->ShowTrayIcon(true, false);
if (FAILED(hr))
     {
     IErrorInfo* errorinfo;
     WideString errortext;
     ::GetErrorInfo(0.&errorinfo);
      //If NULL, no object available
     if (erorinfo==NULL)
          {
          Application->MessageBox("An unspecified COM Automation
          error has occurred.","", MB_OK);
          return;
          }
     else
          errorinfo->GetDescription(&errortext);
     Application->MessageBox(errortext,"", MB_OK);
     }
```

If you follow the method 4.2.3 and create a C++ MFC application, which is created by importing the type library wmCOM.tlb, an interface whose methods provide no return values, so no HRESULT. However, since MFC applications provide convenient exception handling, it is highly recommended to use the MFC class COleDispatchException, to commit the error messages just as easily. Assuming there is a job file called A1.wmj, this might look as follows:

```
void CWmCOMSampleCppDlg::OnPrepareJob()
{
   long nCards = 0L;
   long nObjectsInJob = 0L;

   try
   {
      m_pWMcom->LoadJobFromFile("A1.wmj", &m_iJobID);
      m_pWMcom->GetObjCount(&nObjectsInJob);
      m_pWMcom->SetObjPos(nObjectsInJob - 1L, 100L, 100L);
   }
   catch(COleDispatchException* e)
   {
      TCHAR szMsg[1024] = {0};
      e->GetErrorMessage(szMsg, 1024);
      CString sMsg;
      sMsg.Format("Prepare Job Error:\r\n\r\n%s", szMsg);
      AfxMessageBox(sMsg);
      e->Delete();
   }
}
```

The exception can be tested, by removing the parameter for # 1, when calling SetOb-JPOS (), which removes -1L. Thus, the error message is # 5 "ObjIndex out of bounds" solved.

4.5 Extended Error Handling

All error messages, in weldMARK™ COM server, are in English. In order to enable conversion into other languages, there is an option to prefix the error messages by a number within 7 characters. This 7 characters string part consists of a 6 digit error message-ID and one SPACE character, as delimiter. Reading the leading error-ID enables programmers to show their own, translated errortexts according to the list of messages below.
This option is enabled by setting the DWORD registry entry “ShowErrorIds” in:
[HKEY_CURRENT_USER\Software\RAYLASE\weldMARK\SysDefaults] to 1.
There is currently no possibility to activate this option through weldMARK™.
In order to maintain backward compatibility with previous versions of weldMARK™, default value is 0, which is set during weldMARK™ installation. The defaukt value is 0 (disabled) in case of a missing entry.
The following table gives Error messages with corresponding Error ID numbers.

ID	Error message
0.	Generic Automation Failure
1.	Marker Library not loaded
2.	A required dll file, wmml.DLL, did not load.
3.	No jobs currently in memory
4.	No objects currently loaded
5.	ObjIndex out of bounds
6.	ObjName cannot be empty
7.	JobIndex out of bounds
8.	CurrIndex value out of range
9.	NewIndex value out of range
10.	No scan head cards found
11.	No scan head cards installed
12.	System busy
13.	Hardware Key not found.
14.	Mark not allowed with current Hardware Key.
15.	Download to hardware not allowed with current Hardware Key.
16.	All objects in job must be assigned to the same scan head card.
17.	Not enough scan head card memory to store all objects in job
18.	HardwareStart out of bounds
19.	Repeat out of bounds
20.	Cannot download job to local card hardware
21.	Cannot copy object vector list
22.	CardNum value out of range
23.	HeadNum value out of range
24.	Command value out of range
25.	ProfileIndex value out of range
26.	ProfileIndex out of bounds
27.	Markspeed value out of range
28.	Jumpspeed value out of range
29.	Jumpdelay value out of range
30.	Markdelay value out of range

31.	Polygondelay value out of range
32.	Laseroffdelay value out of range
33.	Laserondelay value out of range
34.	Laserpower value out of range
35.	Frequency value out of range
36.	PulseWidth value out of range
37.	Taxis value out of range
38.	T1 value out of range
39.	T2 value out of range
40.	Varijumpdelay value out of range
41.	Varijumplength value out of range
42.	Wobblesize value out of range
43.	Wobblefrequency value out of range
44.	X value out of range
45.	Y value out of range
46.	No Standard I/O card detected.
47.	File not found:
48.	File not found
49.	File extension not supported
50.	Cannot set graphic file
51.	Wrong file extension. Extension must be 'wmj' or 'wlj'.
52.	Syntax error in job file
53.	Too many jobs loaded
54.	Cannot load and/or process file
55.	Cannot set laser config file
56.	Cannot set laser config file
57.	Operator cancelled operation
58.	Error reading LT position
59.	LT not motorized.
60.	LT motorised, but either LT not enabled, motor offline or axis-disabled
61.	Motor controller not available.
62.	Axis not homed
63.	Axes not homed. Please check and try again.
64.	No axes demanded to be homed
65.	Motor controller card not installed.
66.	Axis 1 not enabled.
67.	Axis 2 not enabled.
68.	Axis 3 not enabled.
69.	Axis 4 not enabled.
70.	LT Axis out of range.
71.	LT Axis not enabled.
72.	Error while positioning LT
73.	Could not adjust LT position
74.	Cannot create new barcode object

75.	CharString cannot be empty
76.	StartAngle value out of range
77.	EndAngle value out of range
78.	Sides value out of range
79.	FontName cannot be empty
80.	Orientation value out of range.
81.	Kerning value out of range.
82.	Leading value out of range.
83.	Styles value out of range.
84.	ParagraphStyle value out of range.
85.	PulseCount value out of range.
86.	WordValue out of range.
87.	Object does not contain any closed paths
88.	FillSpacing value out of range
89.	FillStyle value out of range
90.	Slope1 value out of range
91.	Slope2 value out of range
92.	Note cannot be empty
93.	MOTFFlag value out of range
94.	EncoderSimFlag value out of range
95.	EncoderCal value out of range
96.	MarkStartDelay value out of range
97.	MOTFAngle value out of range
98.	SetControllerConfiguration returned an error
99.	Mode value out of range
100.	PassCount value out of range
101.	Cannot use pens if ObjMarkMode is greater than 1
102.	Problems reading correction file installation folder.
103.	CodeType value out of range
104.	Wrong file extension. Extension must be bmp, jpg, gif, or pcx.
105.	Wrong file extension. Must be 'wmj'.
106.	Error creating new job
107.	A required graphics file referenced by the job could not be found at the specified path.
108.	A result of false was returned from the scancard
109.	ListNum value out of range
110.	WidthReduce value out of range
111.	NarrowToWide value out of range
112.	QuietZone value out of range
113.	Preferences value out of range
114.	DotMatrix value out of range
115.	Pixels value out of range
116.	PixelSep value out of range
117.	Contrast value out of range
118.	Brightness value out of range

119.	InvertPixels value out of range
120.	SkipBlack value out of range
121.	BlackCorners value out of range
122.	ErrorDiffusion value out of range
123.	Rows value out of range
124.	NumRows value out of range
125.	NumColumns value out of range
126.	NumPoints value out of range
127.	Duration value out of range
128.	Paragraph value out of range.
129.	Error while generating new text object
130.	Cannot import vector graphic
131.	OpCode value out of range
132.	Empty string
133.	No markable characters in string
134.	Cannot load bitmap
135.	Save Job failed:
136.	SpacingGrowth out of range
137.	Number of sides for circle approximation out of range
138.	Circle diameter out of range

5 FOCUS SHIFTER

COM server applications can use the full functionality of Focus Shifter feature. The following rules must be respected:

- Use a 3 axis Scan Head with f-Theta lens so that there is no need for Z-compensation through the correction file.
- Use the right Scan Head configuration file with Focus Shifter parameters.
- Update the SP-ICE or RLC PCI/USB control card software with support for Focus Shifter.
- Check so that the control card has DIRECT_Z mode enabled (required only for SP-ICE control card).

There are various types of applications based on weldMARK™ COM server. Those, which only load jobs created by weldMARK™ GUI and mark them, do not require any additional changes regarding Focus Shifter.

- The Scan Head configuration file (selected in System > Preferences > Hardware) is loaded when the system is started. This will set mode to Direct Z and also position the lens so that it corresponds to Z=0, that is in the middle of the marking field.
- Z-height of each object is defined within the object profile which is send to the card before the object is marked. This will cause the Z height to adjust automatically to the parameters read in the Scan Head Configuration file.

5.1 Loading Scan Head Configuration file

At the moment it is not possible to load another Scan Head Configuration file from the COM server application at run time.

5.2 Creating Objects

If Objects have to be created from a COM server based application, then Z position for each object can be specified either by changing the default profile (SetDefaultProfile) which is attached to each new object, or by using SetObjProfile() command.

These commands specify Z Height in bits and the same calibration factor is used as for X and Y. If it would be more convenient to do it in mm or inches then it can be done by using the following relationship:

Z_ Height [bits] = Z_Height [mm] * calfactor [bits/mm]

The Z Volume or the available Z field size can be obtained by a COM server function GetLensCalFactorEx.

The Z field size is distributed from Z=0 to +Zmax in beam direction and Z=0 to –Zmin against the beam direction of laser.

GetLensCalFactorEx command can be used to read the values for ‘calfactor’ and Zmin and Zmax from the System. ‘Calfactor’ is returned as a floating point number so that higher precision can be achieved.

Zmin and Zmax are returned in bits. The corresponding height in mm can be calculated by dividing the values by calfactor.

5.3 Changed/New commands

There is a number of COM server commands which have changed to support the Focus Shifter feature. They are:

- GetLensFactor_Ex()

All the commands dealing with profiles: GetDefaultProfile(), SetDefaultProfile, GetObjProfile(),SetObjProfile().

6 PULSED IPG- AND SPI-LASER

Care must be taken so that signals, required for IPG and SPI laser, are managed correctly.

6.1 Initialization

When the COM server is started laser settings are read from the **laser_ipg.cfg** or **laser_spi.cfg** file, defined in the **controller.ini** file as the active configuration file.

IPG:

```
RAYLASE AG - weldMARK 3
Head Controller Initialization File
Original Filename = controller.ini
Copyright © 2010 RAYLASE AG

[CONTROLLER1]
corrfile1=AS-30-C_0250-1250bo_0400
corrfile2=Y330_10
laserfile=laser_ipg.cfg
…
```

SPI:

```
RAYLASE AG - weldMARK 3
Head Controller Initialization File
Original Filename = controller.ini
Copyright © 2010 RAYLASE AG

[CONTROLLER1]
corrfile1=AS-30-C_0250-1250bo_0400
corrfile2=Y330_10
laserfile=laser_spi.cfg
…
```

Laser Type in the laser configuration file must be set:

IPG:

```
Copyright © 2010 RAYLASE AG
Laser Calibration File
Modified 01/04/10

[LASER]
name=IPG Pulsed Laser
type=4
```

SPI:

```
Copyright © 2010 RAYLASE AG
Laser Calibration File
Modified 01/04/10

[LASER]
name=SPI Laser
type=5
```

On the controller card side, the laser mode is set to YAG1 (SPI Extended Interface) or YAG2 (IPG Interface, SPI Basic Interface), but within weldMARK™ it must be set to IPG or SPI Laser type!

Important:

```
If Set_Mode command is sent from the COM server through
“ScanCardCommand” then YAG1 mode (D5=1, D4=0) or YAG2 mode
(D5=0, D4=0) must be defined for the laser mode!
```

6.2 Mark_In_Progress signal

In order to set laser power correctly, **Mark_In_Progress** signal must be set 10ms before turning the laser on and actually marking.
This can be achieved in several ways:

1. The most convenient way of initializing and using the IPG and SPI Laser from the COM server is if following command sequence is used: **LoadJobFromFile; DownloadAllObj; ScanCardExecute**. In this case the COM server generates the **Mark_In_Progress** signal and a delay required for the IPG or SPI laser.

2. If objects are downloaded one by one with the **DownLoadObj** command, or if any of the Mark Object commands: **MarkAllObj**, **MarkObj**, or **MarkObjEx** is used then **Mark_In_Progress** signal must be explicitly set before marking is started.:
   - **SetMarkInProgressBit(1**)
     Sets the Mark_In_Progress signal**.**
   - **10ms** delay
   - **ScanCardExecute**
     …
     …
   - **SetMarkInProgressBit(0)**
     Resets Mark_In_Progress signal to 0, to enable the laser to be correctly turned on the next time.

3. If a job is created from the COM server via any of the New Object commands (NewText, NewRectangle, …) then **Mark_In_Progress signal** can be set at the beginning of the list in the following way::
   - **ScanCardCommand(Write_Port_list, port, portValue);** port=PortC(12dec, 0CH), and **Mark_In_Progress** (Bit 4) in PortValue set to TRUE.
   - **ScanCardCommand**(**Long_delay**, delay, 0); with delay set to 1000, 1000 x 10mikrosec -> 10msec
   - … various marking commands, and then at the end of the job =>
   - **ScanCardCommand(Write_Port_list, port, value);** port=PortC(12dec, 0CH), and **MarkInProgress (Bit 4**) set to FALSE

6.3 Adjusting laser parameters and setting the power

Laser parameters and power are sent to control card automatically before marking either a whole job or an object. No specific actions are required for this from the COM server.

6.4 Checking for Errors of pulsed IPG/SPI Lasers

SPI Laser Error can be checked through **GetScanCardInput** command:

- GetScanCardInput(CardNum, PortAOffset, *PortAValue),
  PortAOffset is set to 8dec (08H).

If the function returns **PortAValue**, with bit **D6=**FALSE, then there an Error occurred.

6.5 Resetting Errors of pulsed IPG/SPI Lasers

If an SPI Laser Error occurred, then it must be reset before the laser can be turned on again. Resetting SPI Laser Error can be done through "**SetScanCardOutput"** COM server command:

- **SetScanCardOutput** (CardNum, PortCOffset, PortCValue, *unused*),
  PortCOffest = 12dec (0CH) and
  bit D5(Remote_Execute_1) set to **TRUE** in the PortCValue.
- There should be a delay of minimum 1ms before sending the next command
- **SetScanCardOutput** (CardNum, PortCOffset, PortCValue, unused),
  PortCOffest = 12dec (0CH) and
  bit D5(Remote_Execute_1) set to **FALSE** in the PortCValue

7 MARKER LIBRARY FUNCTIONS

In this section, the command interface library is documented by function, and then alphabetically.

7.1 Function Overview

The following list organizes the available commands into functional groups.

Marker Library functions

AttachToMarker
ReleaseMarker
ShowTrayIcon

Scan Card functions

GetScanCardCount
GetScanCardCapacity
GetScanHeadCount
ScanCardCommand
ScanCardExecute
TerminateMark
GetBusyStatus
GetBusyStatusEx
SetMOTFConfig
GetMOTFConfig
SetScanCardOutput
GetScanCardInput
GoToXY

I/O Card functions

IsIOCardInstalled
GetStartProcessBit
SetBusyReadyBit
SetMarkInProgressBit
GetUserInWord
SetProcessEnabledWord
SetUserOutWord

Lens file functions

LoadLensCalFile
GetLensCalFile
GetLensCalFactor
GetLensCalFactorEx

Laser functions

LoadLaserConfigFile
GetLaserConfigFile
GetLaserPowerMinMax
GetLaserName
GetReadStatusWord
EnableLaser
TurnLaserOff
TurnLaserOn

Motor Controller functions

HomeAxes
HomeLTAxis

JobObject functions

GetJobCount
GetJobCorrFile
GetJobCorrFlag
NewJob
SetActiveJob
CloseJob
LoadJobFromFile
SaveJobToFile

MarkObject functions

NewBitmap
SetBitmapAttributes
SetBitmapGrayScaleMode
GetBitmapAttributes
GetBitmapGrayScaleMode
GetBmpEndOfLineDelay
GetBmpLineShiftCorrection
GetBmpSkippedPixelTreshold
SetBmpEndOfLineDelay
SetBmpLineShiftCorrection
SetBmpSkippedPixelTreshold
NewVectorGraphic
SetVectorGraphicAttributes
GetVectorGraphicAttributes
NewText
SetTextAttributes
GetTextAttributes
NewBarcode
SetBarcodeAttributes
SetBarcodeAttributesEx
GetBarcodeAttributes
GetBarcodeAttributesEx
NewLine
NewRect
NewPolygon
SetPolygonAttributes
GetPolygonAttributes
NewDrill
SetDrillAttributes
GetDrillAttributes
GetObjCount
GetObjMemSize
MoveObjInList
DeleteObj
DeleteAllObj
SetObjScanCardNum
GetObjScanCardNum
GetObjVectorList
GetObjFillList
SetObjMarkMode
GetObjMarkMode
SetObjNumPasses
SetObjUsePensFlag
GetObjUsePensFlag
GetObjPens
GetObjNumPasses
SetObjMarkFillFlag
GetObjMarkFillFlag
SetObjMarkOutlineFlag
GetObjMarkOutlineFlag
SetDefaultProfile
GetDefaultProfile
SetObjProfile
GetObjProfile
SetObjName
GetObjName
SetObjFill
SetObjFillEx
GetObjFill
GetObjFillEx
SetObjNote
GetObjNote
MarkObj
MarkObjEx
MarkAllObj
DownloadObject
DownloadAllObj
GetAllObjRect
GetObjRect
IsObjOutOfBounds
GetObjType
GetObjTypeString
CenterObj
OffsetObj
RotateObj
RotateObjEx
ScaleObj
SetObjPos
SetObjSize
SkewObj
SetObjToRect
SetObjCharString
GetObjCharString
SetObjGraphicFile
GetObjGraphicFile

7.2 Functions

The following list describes all automation functions in alphabetical order.

AttachToMarker

Purpose	Loads the Marker libraries and initializes the hardware.
Implementation	HRESULT *AttachToMarker* (void)
Comments	When first loading the COM server, it is not necessary to call this function. However, if you have called *ReleaseMarker*, then you must call this function to regain access to the scan card hardware. Returns *S_OK* if the function succeeds.
See Also	*ReleaseMarker*

CenterObj

Purpose	Positions the center of an object at the center of the marking field.	
Implementation	HRESULT *CenterObj* (int ObjIndex)	
Parameters	*ObjIndex*	Index of object in the ObjectList. Valid range: [0 to (number of objects-1)]
Comments	The marking field is described using a Cartesian coordinate system, with (0,0) at the center of the field, (-32768, -32768) at the bottom left corner, and (32767, 32767) at the top right corner. Returns *S_OK* if the function succeeds.	

CloseJob

Purpose	Closes a job and clear objects from memory.	
Implementation	HRESULT *CloseJob* (int JobIndex)	
Parameters	*JobIndex*	Index of Job in the JobList Valid range: [0 to (number of jobs-1)]
Comments	If there are still jobs in the JobList, the Active Job is set to 0; otherwise the Active Job is set to –1. Returns *S_OK* if the function succeeds.	

DeleteAllObj

Purpose	Deletes all objects currently loaded in the Active Job.
Implementation	HRESULT *DeleteAllObj* (void)
Comments	Returns *S_OK* if the function succeeds.

DeleteObj

Purpose	Deletes an object from the Active Job.
Implementation	HRESULT *DeleteObj* (int ObjIndex)
Parameters	ObjIndex: Index of object in the ObjectList
Comments	Returns *S_OK* if the function succeeds.

DownloadAllObj

Purpose	Copies the vector lists of all objects in the Active Job to the scan card hardware.	
Implementation	HRESULT *DownloadAllObj* (int Orientation, int HardwareStart, int Repeat)	
Parameters	*Orientation*	Rotates the marked image relative to screen Valid values: [0, 90, 180, 270]
	HardwareStart	Configures card to wait for external start signal on card. 1 = wait for external start. Valid values: [0, 1]
	Repeat	Configures card to continuously process list. 1 = repeat continuously Valid values: [0, 1]
Comments	After calling *DownloadAllObj*, you must call *ScanCardExecute* to start the processing of the list. If there is not enough memory to store all objects in the Active Job, the function will fail, and an error is returned. You can discover the capacity of the scan card memory by calling *GetScanCardCapacity*. Calling *GetObjMemSize* returns the memory size requirements of each object. Returns *S_OK* if the function succeeds.	
See Also	*DownloadObj, ScanCardExecute, GetScanCardCapacity, GetObjMemSize*	

DownloadObj

Purpose	Copies the vector list of an object in the Active Job to the scan card hardware.	
Implementation	HRESULT *DownloadObj* (int ObjIndex, int Orientation)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	Orientation	Rotates the marked image relative to screen Valid values: [0, 90, 180, 270]
Comments	Before calling *DownloadObj*, the scan card list must be opened with a call to *ScanCardCommand*, passing the StartList1 or StartList2 parameter. To execute objects downloaded to the scan card, you must first close the list with a call to *ScanCardCommand*, passing the SetEndOfList parameter. The list can then be executed with a call to *ScanCardExecute* to start the processing of the list. Returns *S_OK* if the function succeeds.	
See Also	*DownloadAllObj*, *ScanCardCommand*, *ScanCardExecute*, *GetScanCardCapacity*, *GetObjMemSize*	

EnableLaser

Purpose	Enables or disables the laser.	
Implementation	HRESULT *EnableLaser* (int Flag)	
Parameters	*Flag*	Valid values: [0, 1]
Comments	When the laser is disabled, calls to *MarkObj*, *MarkAllObj*, etc. will succeed, but the laser will not turn on. Returns *S_OK* if the function succeeds.	
See Also	*MarkObj*, *MarkAllObj*	

GetAllObjRect

Purpose	Retrieves the smallest rectangle that fits around all objects in the Active Job.	
Implementation	HRESULT *GetAllObjRect* (float* Left, float* Top, float* Right, float* Bottom)	
Returns	*Left*	The x-coordinate of the upper-left corner of the bounding rectangle.
	Top	The y-coordinate of the upper-left corner of the bounding rectangle
	Right	The x-coordinate of the lower-right corner of the bounding rectangle.
	Bottom	The y-coordinate of the lower-right corner of the bounding rectangle
Comments	The marking field is described using a Cartesian coordinate system, with (0,0) at the center of the field, (-32768, -32768) at the bottom left corner, and (32767, 32767) at the top right corner. Returns *S_OK* if the function succeeds.	

GetBarcodeAttributes
GetBarcodeAttributesEx

Purpose	Gets the attributes of a barcode object.	
Implementation	HRESULT *GetBarcodeAttributesEx*(int ObjIndex, int* WidthReduce, int* NarrowToWide, int* QuietZone, int* Preferences, int* DotMatrix, int* Pixels, int* PulseCount, int* SpaceGrowth, int* CircleNumSides, int* CircleDiameter)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*WidthReduce*	The amount of reduction in the width of all bars. Valid for 1D codes only. Units: % of bar width
	NarrowToWide	The change in width ratio of the narrow and wide bars from default. Only valid for 1D codes. For PDF417, it represents the aspect ratio of the height to width of the entire barcode. Units: % of bar width
	QuietZone	When inverting a barcode, the amount of quiet space surrounding the code. DataMatrix, QR code, PDF417 = number of cells Units: 1D = % of code width.
	Preferences	See *SetBarcodeAttributes* for Preferences details.
	DotMatrix	The dot matrix flag. Set to 1 (one) to enable dot matrix mode.
	Pixels	For dot matrix mode, depends on barcode type: 1D codes: The spacing between adjacent pixels Units: bits 2D codes: The number of rows and columns in the pixel array in each cell.
	PulseCount	The number of laser pulses fired at each dot using the current laser frequency and pulse width settings.
1)	*SpaceGrowth*	Filling a 2D barcode with circle dots. Increases or reduces the distance between circle dots. Units: ± 100% of the circle diameter
1)	*CircleNumSides*	Filling a 2D barcode with circle dots. Number of segments to describe a circle.
1)	*CircleDiameter*	Filling a 2D barcode with circle dots. Definition of the circle diameter in Unit bits
Comments	Returns *S_OK* if the function succeeds.	
See Also	*SetBarcodeAttributes, SetBarcodeAttributesEx*	

1) GetBarcodeAttributesEx only

GetBitmapAttributes

Purpose	Retrieves the attributes of a Bitmap object.	
Implementation	HRESULT *GetBitmapAttributes* (int ObjIndex, int* PixelSep, int* Contrast, int* Brightness, int* InvertPixels, int* SkipBlack, int* BlackCorners, int* ErrorDiffusion)	
Parameters	*ObjIndex*	Index of object in the ObjectList. Valid range: [0 to (number of objects-1)]
Returns	*PixelSep*	Distance between adjacent pixels. Units: bits
	Contrast	A relative value affecting the difference between the darkest and lightest pixels.
	Brightness	A relative value affecting the overall brightness of all pixels.
	InvertPixels	Flag indicating whether the pixels are inverted (black to white). 0 = not inverted 1 = inverted.
	SkipBlack	Flag indicating whether black pixels are jumped over when marking the bitmap. 0 = Do not skip 1 = Skip black pixels
	BlackCorners	Flag indicating what color to make pixels in the corners if the bitmap has been rotated to an angle other that 90, 180 or 270. 0 = white 1 = black
	ErrorDiffusion	Flag indicating whether the Error Diffusion algorithm has been applied to the bitmap. 0 = no error diffusion 1 = error diffusion applied.
Comments	Returns *S_OK* if function succeeds.	
See Also	*SetBitmapAttributes*	

GetBitmapGrayScaleMode

Purpose	Retrieves the mode defining the way in which the laser power is controlled when marking grayscale bitmaps.	
Implementation	HRESULT *GetBitmapGrayScaleMode* (int ObjIndex, int* Mode)	
Parameters	*ObjIndex*	Index of object in the ObjectList.
Returns	*Mode*	Value of the used bitmap algorithm that was attached to the object. 0 = POINT_AND_SHOOT_ALG 4 = ANALOG_POWERSET_ALG 5 = DIGITAL_POWERSET_ALG 9 = PWM_ALG
Comments	For ErrorDiffusion mode use function *GetBitmapAttributes*. Returns *S_OK* if function succeeds.	
See Also	*SetBitmapGrayScaleMode, GetBitmapAttributes, SetBitmapAttributes*	

GetBmpEndOfLineDelay

Purpose	Returns the actual value for delaying the mark process at the end of each line and after jumping to the start of the next line.	
Implementation	HRESULT *GetBmpEndOfLineDelay* (int ObjIndex, int Delay)	
Parameters	*ObjIndex*	Index of object in the ObjectList. Valid range: [0 to (number of objects-1)]
	Delay	Valid range: [0 to 20000] µSec
Comments	Returns *S_OK* if function succeeds.	
See Also	*SetBmpEndOfLineDelay*	

GetBmpLineShiftCorrection

Purpose	Returns the correction value, currently programmed to compensate the shift errors while marking bitmap object in bidirectional mode.	
Implementation	HRESULT *GetBmpLineShiftCorrection* (int ObjIndex, int* Correction)	
Parameters	*ObjIndex*	Index of object in the ObjectList. Valid range: [0 to (number of objects-1)]
	Correction	Valid range: [0 to 65500] bits
Comments	Returns *S_OK* if function succeeds.	
See Also	*SetBmpLineShiftCorrection*	

GetBmpSkippedPixelTreshold

Purpose	Returns the currently programmed threshold value, which was programmed to skip low gray values at the marking of bitmap objects.		
Implementation	HRESULT *GetBmpSkippedPixelTreshold* (int ObjIndex, int MinPixel)		
Parameters	*ObjIndex*	Index of object in the ObjectList. Valid range: [0 to (number of objects-1)]	
	MinPixel	Value ≤ 1	No skip of white pixels
		Value ≥ 2	Skips processing of pixel with greyscale value < the defined treshold value
Comments	Returns *S_OK* if function succeeds.		
See Also	*SetBmpSkippedPixelTreshold*		

GetBusyStatus

Purpose	Retrieves the system busy status.	
Implementation	HRESULT *GetBusyStatus* (int CardNum, int* BusyFlag)	
Parameters	*CardNum*	Index of scan head card Valid range: [0 to (number of cards-1)]
Returns	*BusyFlag*	0 = ready 1 = busy
Comments	The range of scan head card index values can be determined by calling *GetScanCardCount*. Whenever the system is currently executing an object, the Busy status is 1 (one). An application must call this function before making any call to modify or execute an object. Returns *S_OK* if function succeeds.	
See Also	*GetBusyStatusEx*	

GetBusyStatusEx

Purpose	Retrieves the system busy status, external start count, and hardware stop status.	
Implementation	HRESULT *GetBusyStatus* (int CardNum, int* ListLoading, int* CardBusy, int*ExtStarts, int* HardwareStop)	
Parameters	*CardNum*	Index of scan head card Valid range: [0 to (number of cards-1)]
Returns	*ListLoading*	0 = idle 1 = list actively loading vectors
	CardBusy	0 = idle 1 = card actively executing vector list
	ExtStarts	The number of External Starts received on the card hardware since the last card reset.
	HardwareStop	0 = no hardware stops 1 = a hardware stop has been received since the last card reset
Comments	The range of scan head card index values can be determined by calling *GetScanCardCount*. An application must call this function before making any call to modify or execute an object. Returns *S_OK* if function succeeds.	
See Also	*GetBusyStatus*	

GetDefaultProfile

Purpose	Retrieves the default Profile settings applied to all new objects.	
Implementation	HRESULT *GetDefaultProfile* (int ProfileIndex, int* Mode, int* PassCount, double* Markspeed, double* Jumpspeed, int* Jumpdelay, int* Markdelay, int* Polygondelay, float* Laserpower, int* Laseroffdelay, int* Laserondelay, int* TAxis, double* T1, int* T2, int* Unused, int* Varijumpdelay, int* Varijumplength, int* Wobblesize, double* Wobblefrequency, int* Varipolydelay)	
Parameters	*ProfileIndex*	Index of the Profile. Valid range: [0 to 7]
Returns	*Mode*	See *SetObjMarkMode* for a description of this parameter.
	PassCount	See *SetObjNumPasses* for a description of this parameter.
	MarkSpeed	Defines the speed of the laser spot while marking. Units: bits/mm
	Jumpspeed	Defines the speed at which the mirrors jump to the next marking vector. Units: bits/mm
	Jumpdelay	Defines the delay after a jump and before the next marking vector starts. Units: µSec
	Markdelay	Defines the delay between a marking vector and a jump vector. Units: µSec
	Polygondelay	Defines the delay between contiguous marking vectors. Units: µSec
	Laserpower	Defines the laser power for non CO_2-type lasers. Valid range: [0 to 100] % (percent) Note: Laserpower for CO_2- type lasers is defined as: Duty Cycle (percent) = 0.1 × T2 [µs] × T1 [kHz]
	Laseroffdelay	Defines the delay after the last marking vector finishes and the laser is turned off. Units: µSec

	Laserondelay	Defines the delay after a marking vector starts and the laser is turned on. Units: µSec
	TAxis	Defines the Z position of the object. +Z is toward the scan head and –Z away from the scan head. Position is defined in bits and the same calibration factor is used as for x and y. Z field size is limited by the available Linear Translator movement. Values for Zmin and Zmax are defined in the scan head configuration file and can be read with *GetLensCalFactorEx* command. Units: bits Valid range: [Zmin to Zmax]
	T1	Defines the frequency of the laser modulation signal. Units: kHz
	T2	Defines the pulse width of the laser modulation signal. Units: µSec
	Unused	Reserved for future use.
	Varijumpdelay	Defines the delay after a jump and before the next marking vector starts if variable jump delay is in effect. Units: µSec
	Varijumplength	Defines the length of a vector, at which any vector that is longer will use the Varijumpdelay parameter, and any vector that is shorter will use the Jumpdelay parameter. Units: bits
	Wobblesize	The diameter of the circle created when the spot is dithered. Units: Bits
	Wobblefrequency	The frequency of the laser spot as it dithers around the circle defined in Wobblesize. Units: Hz (cycles per second)
	Unused	Reserved
	Varipolydelay	Reserved
Comments	The Mode and PassCount parameters are global to all four individual Profiles. Returns *S_OK* if the function succeeds.	
See Also	*SetObjProfile*, *GetObjProfile*, *SetDefaultProfile*, *SetObjMarkMode*, *SetObjNumPasses*	

GetDrillAttributes

Purpose	Returns the attributes of a drill object.	
Implementation	HRESULT *GetDrillAttributes* (int ObjIndex, int* Rows, int* Columns, int* NumPoints, int* Duration)	
Parameters	*ObjIndex:*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*Rows*	The number of rows in the point array
	Columns	The number of columns in the point array
	NumPoints	The total number of points in the point array
	Duration	Number of pulses the laser will fire at each point.
Comments	Returns *S_OK* if function succeeds.	
See Also	*SetDrillAttributes*	

GetJobCount

Purpose	Returns the number of jobs currently in memory.	
Implementation	HRESULT *GetJobCount* (int* JobCount)	
Returns	*JobCount*	The job count.
Comments	Returns *S_OK* if function succeeds.	
See Also	*SetActiveJob*, *NewJob*, *CloseJob*	

GetJobCorrFile

Purpose	Gets the full pathname of the lens correction file used in the active job	
Implementation	HRESULT *GetJobCorrFile* (BSTR* CalFile)	
Returns	*CalFile*	Path and name of the correction file associated with the job.
Comments	Returns *S_OK* if function succeeds.	
See Also	*LoadJobFromFile, LoadLensCalFile*	

GetJobCorrFlag

Purpose	Retrieves the ‘Use job correction file’ flag from the active job.	
Implementation	HRESULT *GetCorrFlag* (int* JobCorrFlag)	
Returns	*JobCorrFlag*	0: do not use job correction file 1: use job correction file
Comments	Returns *S_OK* if function succeeds.	
See Also	*LoadJobFromFile, LoadLensCalFile*	

GetLaserConfigFile

Purpose	Returns the fully qualified filename of the laser config file.	
Implementation	HRESULT *GetLaserConfigFile* (int CardNum, BSTR* ConfigFile)	
Parameters	*CardNum*	Index of scan head card Valid range: [0 to (number of cards-1)]
Returns	ConfigFile: The fully qualified filename of the laser config file.	
Comments	Returns *S_OK* if function succeeds.	
See Also	*LoadLaserConfigFile*	

GetLaserName

Purpose	Returns the name of the laser from the currently loaded laser config file.	
Implementation	HRESULT *GetLaserName* (int CardNum, BSTR* LaserName)	
Parameters	*CardNum*	Index of scan head card Valid range: [0 to (number of cards-1)]
Returns	*LaserName*	The name of the laser in the laser config file
Comments	Returns *S_OK* if function succeeds.	

GetLaserPowerMinMax

Purpose	Gets the minimum and maximum allowable laser power settings.	
Implementation	HRESULT *GetLaserPowerMinMax* (int CardNum, int* Min, int* Max)	
Parameters	*CardNum*	Index of scan head card Valid range: [0 to (number of cards-1)]
Returns	*Min*	The minimum valid laser power value. Units: % (percent)
	Max	The maximum valid laser power value. Units: % (percent)
Comments	Use *GetLaserPowerMinMax* to discover the upper and lower range for the laser power settings. These values must be used to range check when calling *SetObjProfile* or *SetDefaultProfile*. Returns *S_OK* if function succeeds.	
See Also	*SetObjProfile*, *SetDefaultProfile*	

GetLensCalFactor

Purpose	Gets the calibration factor in bits/mm of the specific scan head lens file.	
Implementation	HRESULT *GetLensCalFactor* (int CardNum, int HeadNum, int* CalFactor)	
Parameters	*CardNum*	Index of scan head card. Valid range: [0 to (number of cards-1)]
	HeadNum	Index of scan head. Valid range: [0 to (number of heads-1)]
Returns	*CalFactor*	The calibration factor. Units:bits/mm
Comments	Use the scan head calibration factor to convert dimensional data from bits into real world dimensions. Each SP-ICE scan card can have up to four scan heads attached (master/slave), the RLC scan card can have just one scan head attached. Use *GetScanHeadCount* to discover the number of heads attached to a specific card. Returns *S_OK* if function succeeds.	
See Also	*LoadLensCalFile*, *GetScanHeadCount, GetLensCalFactorEx*	

GetLensCalFactorEx

Purpose	Gets the calibration factor in bits/mm of the specific scan head lens file.	
Implementation	HRESULT *GetLensCalFactorEx* (int CardNum, int HeadNum, int* CalFactor, int* Zmin, int* Zmax)	
Parameters	*CardNum*	Index of scan head card. Valid range: [0 to (number of cards-1)]
	HeadNum	Index of scan head. Valid range: [0 to (number of heads-1)] for SP-ICE, RLC-USB, RLC-PCI control cards always set to 0
Returns	*CalFactor*	Calibration factor Units: bits/mm
	Zmin	Minimum allowed Z position Units: bits/mm Range: +/-32767
	Zmax	Maximum allowed Z position Units: bits/mm Range: +/-32767
Comments	Use the scan head calibration factor to convert dimensional data from bits into real world dimensions. This command is similar to *GetLensCalFactor,* only it returns a float value for CalFactor and not an integer. It also returns values for Zmin and Zmax which define the available Z volume. Intended use for Focus Shifter. Returns *S_OK* if function succeeds.	
See Also	*LoadLensCalFile*, *GetScanHeadCount*, *GetLensCalFactor*	

GetLensCalFile

Purpose	Gets the fully qualified filename of the lens correction file used by a specific scan head.	
Implementation	HRESULT *GetLensCalFile* (int CardNum, int HeadNum, BSTR* CalFile)	
Parameters	*CardNum*	Index of scan head card Valid range: [0 to (number of cards-1)]
	HeadNum	Index of scan head. Valid range: [0 to (number of heads-1)]
Returns	*CalFile*	The fully qualified lens correction file.
Comments	Returns *S_OK* if function succeeds.	
See Also	*LoadLensCalFile*	

GetMOTFConfig

Purpose	Gets the Mark on the Fly configuration parameters.	
Implementation	HRESULT *GetMOTFConfig* (int CardNum, int* MOTFFlag, int* EncoderSimFlag, double* EncoderCal, int* MarkStartDelay, double* MOTFAngle)	
Parameters	*CardNum*	Index of scan head card. Valid range: [0 to (number of cards-1)]
Returns	*MOTFFlag*	The Mark on the Fly flag. 0 = disable Mark on the Fly 1 = enable.
	EncoderSimFlag	The encoder simulation flag. 1 = simulate an encoder
	EncoderCal	The calibration factor of the encoder. Units: counts/mm
	MarkStartDelay	The number of encoder counts to wait before starting the mark. Units: counts
	MOTFAngle	The angular orientation of the moving part with respect to the x-axis. Units: degrees
Comments	For a part that is moving along the x-axis in the direction of increasing x, *MOTFAngle* is 0. For a part that is moving along the y-axis in the direction of increasing y, *MOTFAngle* is 90, etc. Returns *S_OK* if the function succeeds.	
See Also	*SetMOTFConfig*	

GetObjCharString

Purpose	Retrieve the Character String value of a Text object or Barcode object.	
Implementation	HRESULT *GetObjCharString* (int ObjIndex, BSTR* CharString)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*CharString*	The Character String value
Comments	Returns *S_OK* if function succeeds	

GetObjCount

Purpose	Gets the number of objects in the Active Job.	
Implementation	HRESULT *GetObjCount* (int* ObjCount)	
Returns	*ObjCount*	The number of objects in the Active Job
Comments	The ObjectList is 0 based, so if *GetObjCount* returns 10, the loaded objects are numbered 0-9. Returns *S_OK* if function succeeds.	

GetObjFill
GetObjFillEx

Purpose	Gets fill parameters of an object.	
Implementation	HRESULT *GetObjFill* (int ObjIndex, int* FillSpacing, int* FillOffset, int* Slope1, int* Slope2, int* FillStyle)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*FillSpacing*	The distance between adjacent fill lines. Units: bits
1)	*FillOffset*	The distance between any endpoint of filling hatchlines and the outlines of the object. Units: bits
	Slope1	The angle with respect to the x-axis of the first set of fill lines.
	Slope2	The angle with respect to the x-axis of the second set of fill lines. (for crosshatch)
	FillStyle	The fill style. 0 = parallel lines 1 = crosshatch 2 = bidirectional 3 = bidirectional and crosshatch 6 = bidirectional using meanderfill 7 = bidirectional+crosshatch using meanderfill
Comments	Only objects with closed paths can be filled. Returns *S_OK* if the function succeeds.	
See Also	*SetObjFill*	

1) FillOffsetEx only

GetObjFillList

Purpose	Returns a SAFEARRAY containing a list of vector commands that describe an objects fill image.	
Implementation	HRESULT *GetObjFillList* (int ObjIndex, SAFEARRAY(int)* ListArray)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*ListArray*	Pointer to the SAFEARRAY containing the vector commands.
Comments	The values in the safe array are listed in sets of three: *Parameter1* *Parameter2* *Parameter3* If using C/C++, the programmer is responsible for releasing the SAFEARRAY memory when done with the array with calls to: SafeArrayUnaccessData(SAFEARRAY*); SafeArrayDestroy(SAFEARRAY*); The above calls are not necessary when using VisualBasic, as VB does the memory management automatically. See the sample source code files included with the weldMARK™ 3 installation package for details on how to implement calls using SAFEARRAYS. Returns *S_OK* if the function succeeds.	Drawing command. 0 = MoveTo 1 = LineTo X coordinate Y coordinate

GetObjGraphicFile

Purpose	Gets the fully qualified filename containing the source vector graphic data.	
Implementation	HRESULT *GetObjGraphicFile* (int ObjIndex, BSTR* GraphicFile)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*GraphicFile*	The fully qualified filename containing the source vector graphic data.
Comments	Returns *S_OK* if function succeeds.	
See Also	*SetObjGraphicFile*	

GetObjMarkFillFlag

Purpose	Gets the MarkFill flag of an object.	
Implementation	HRESULT *GetObjMarkFillFlag* (int ObjIndex, int* MarkFillFlag)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*MarkFillFlag*	The MarkFill flag. 0 = no mark 1 = mark
Comments	If the flag is set to 1 (one), the objects fill will mark. Returns *S_OK* if function succeeds.	
See Also	*GetObjMarkOutlineFlag*	

GetObjMarkOutlineFlag

Purpose	Gets the MarkOutline flag of an object.	
Implementation	HRESULT *GetObjMarkOutlineFlag* (int ObjIndex, int* MarkOutlineFlag)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*MarkOutlineFlag*	The MarkOutline flag. 0 = no mark 1 = mark
Comments	If the flag is set to 1 (one), the objects outline will mark. Returns *S_OK* if function succeeds.	
See Also	*GetObjMarkFillFlag*	

GetObjMemSize

Purpose	Gets the scan card list memory size requirements of an object.	
Implementation	HRESULT *GetObjMemSize* (int ObjIndex, int* Size)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*Size*	The number of memory locations required to store object in the scan card list.
Comments	Use *GetObjMemSize* to determine how many objects can fit into the scan card list memory. You can discover the size of the scan card list by calling *GetScanCardCapacity*. This function is used when intending to call *DownloadObj* and *DownloadAllObj*. Returns *S_OK* if function succeeds.	
See Also	*GetScanCardCapacity*, *DownloadObj*, *DownloadAllObj*	

GetObjMarkMode

Purpose	Gets the current MarkMode of an object.
Implementation	HRESULT *GetObjMarkMode* (int ObjIndex, int* Mode)
Parameters	*ObjIndex* Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*Mode* The current MarkMode, which can have the following values: 0 = Mark object once. NumPasses is ignored. 1 = Mark object using the value of NumPasses. 2 = Mark object with two passes, where: Pass1 uses Profile0 Pass2 uses Profile1 3 = Mark object with three passes, where: Pass1 uses Profile0 Pass2 uses Profile1 Pass3 uses Profile2 4 = Mark object with four passes, where: Pass1 uses Profile0 Pass2 uses Profile1 Pass3 uses Profile2 Pass4 uses Profile3
Comments	Use *SetObjNumPasses* to set the NumPasses value of an object. Use *SetObjProfile* to change the profile settings of an object. Returns *S_OK* if function succeeds.
See Also	*SetObjNumPasses*, *SetObjProfile*

GetObjName

Purpose	Retrieves the object name.
Implementation	HRESULT *GetObjName* (int ObjIndex, BSTR* ObjName)
Parameters	*ObjIndex* Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*ObjName* The object name.
Comments	Returns *S_OK* if the function succeeds.
See Also	*SetObjName*

GetObjNote

Purpose	Gets the note stored in the object.	
Implementation	HRESULT *GetObjNote* (int ObjIndex, BSTR* Note)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*Note*	The note stored in the object.
Comments	Returns *S_OK* if function succeeds.	
See Also	*SetObjNote*	

GetObjNumPasses

Purpose	Gets the NumPasses value of an object.	
Implementation	HRESULT *GetObjNumPasses* (int ObjIndex, int* PassCount)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*PassCount*	The number of times to mark the object.
Comments	The use of NumPasses depends on the objects MarkMode setting. Use *GetObjMarkMode* to discover the current setting, and *SetObjMarkMode* to change it. Returns *S_OK* if function succeeds.	
See Also	*SetObjNumPasses*, *GetObjMarkMode*, *SetObjMarkMode*	

GetObjPens

Purpose	Gets the pens contained in the object.	
Implementation	HRESULT *GetObjPens* (int ObjIndex, int* Pen1, int* Pen2, int* Pen3, int* Pen4, int* Pen5, int* Pen6, int* Pen7, int* Pen8)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*Pen1*	The pen flag. 0 = Pen not found, 1 = Contains Pen1.
	Pen2	The pen flag. 0 = Pen not found, 1 = Contains Pen2.
	Pen3	The pen flag. 0 = Pen not found, 1 = Contains Pen3.
	Pen4	The pen flag. 0 = Pen not found, 1 = Contains Pen4.
	Pen5	The pen flag. 0 = Pen not found, 1 = Contains Pen5.
	Pen6	The pen flag. 0 = Pen not found, 1 = Contains Pen6.
	Pen7	The pen flag. 0 = Pen not found, 1 = Contains Pen7.
	Pen8	The pen flag. 0 = Pen not found, 1 = Contains Pen8.
Comments	If the object contains pen information (usually in *.plt files), the Profile used to mark the object is dynamically selected during the marking of the object by the current pen using the following mapping: Pen Number — Uses Profile 1 — 0 2 — 1 3 — 2 4 — 3 5 — 4 6 — 5 7 — 6 8 — 7 You can turn the pen function on or off by calling *SetObjUsePensFlag*. Returns *S_OK* if function succeeds.	
See Also	*SetObjUsePensFlag*, *GetObjUsePensFlag*	

GetObjProfile

Purpose	Retrieve the Profile settings for a mark object.	
Implementation	HRESULT *GetObjProfile* (int ObjIndex, int ProfileIndex, double* Markspeed, double* Jumpspeed, int* Jumpdelay, int* Markdelay, int* Polygondelay, float* Laserpower, int* Laseroffdelay, int* Laserondelay, int* TAxis, double* T1, int* T2, int* Unused, int* Varijumpdelay, int* Varijumplength, int* Wobblesize, double* Wobblefrequency, int* Autosegmentation, int* Varipolydelay)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	ProfileIndex	Index of Profile Valid range: [0 to 7]
Returns	*MarkSpeed*	Defines the speed of the laser spot while marking. Units: bits/mm
	Jumpspeed	Defines the speed at which the mirrors jump to the next marking vector. Units: bits/mm
	Jumpdelay	Defines the delay after a jump and before the next marking vector starts. Units: µSec
	Markdelay	Defines the delay between a marking vector and a jump vector. Units: µSec
	Polygondelay	Defines the delay between contiguous marking vectors. Units: µSec
	Laserpower	Defines the laser power for non CO_2-type lasers. Valid range: [0 to 100] % (percent) Note: Laserpower for CO_2- type lasers is defined as: Duty Cycle (percent) = 0.1 × T2 [µs] × T1 [kHz]
	Laseroffdelay	Defines the delay after the last marking vector finishes and the laser is turned off. Units: µSec
	Laserondelay	Defines the delay after a marking vector starts and the laser is turned on. Units: µSec

	TAxis	Defines the Z position of the object. +Z is toward the scan head and –Z away from the scan head. Position is defined in bits and the same calibration factor is used as for x and y. Z field size is limited by the available Linear Translator movement. Values for Zmin and Zmax are defined in the scan head configuration file and can be read with *GetLensCalFactorEx* command. Units: bits Valid range: [Zmin to Zmax]
	T1	Defines the frequency of the laser modulation signal. Units: kHz
	T2	Defines the pulse width of the laser modulation signal. Units: µSec
	Unused	Reserved for future use.
	Varijumpdelay	Defines the delay after a jump and before the next marking vector starts if variable jump delay is in effect. Units: µSec
	Varijumplength	Defines the length of a vector, at which any vector that is longer will use the Varijumpdelay parameter, and any vector that is shorter will use the Jumpdelay parameter. Units: bits
	Wobblesize	The diameter of the circle created when the spot is dithered. Units: bits
	Wobblefrequency	The frequency of the laser spot as it dithers around the circle defined in Wobblesize. Units: Hz (cycles per second)
	Unused	Reserved
	Varipolydelay	Reserved
Comments	An object has eight profiles available, Profile0 to Profile7. Returns *S_OK* if the function succeeds.	
See Also	*SetObjProfile*, *SetDefaultProfile*, *GetDefaultProfile*	

GetObjRect

Purpose	Retrieve the position and size of an object.	
Implementation	HRESULT *GetObjRect* (int ObjIndex, float* Left, float* Top, float* Right, float* Bottom)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*Left*	The x-coordinate of the upper-left corner of the bounding rectangle.
	Top	The y-coordinate of the upper-left corner of the bounding rectangle
	Right	The x-coordinate of the lower-right corner of the bounding rectangle.
	Bottom	The y-coordinate of the lower-right corner of the bounding rectangle
Comments	The marking field is described using a Cartesian coordinate system, with (0,0) at the center of the field, (-32768, -32768) at the bottom left corner, and (32767, 32767) at the top right corner. Every MarkObject has a bounding rectangle, which describes the smallest rectangle that will fit around the object. Returns *S_OK* if the function succeeds.	
See Also	*GetAllObjRect*	

GetObjScanCardNum

Purpose	Get the scan card index number of an object.	
Implementation	HRESULT *GetObjScanCardNum* (int ObjIndex, int* CardNum)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*CardNum*	Index of scan head card.
Comments	An objects scan head card index controls which card is used when marking the object. When an object is initially created, it has a CardNum of 0. If there is only one scan card in use, there is no need to call this function. Returns *S_OK* if the function succeeds.	
See Also	SetScanCardNum	

GetObjType

Purpose	Gets the object type of an object.	
Implementation	HRESULT *GetObjType* (int ObjIndex, int* ObjType)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*ObjType*	The numerical object type, which can be one of the following values:

CAD Graphic	1
Polyline	2
Bezier	3
PDF	4/7
Barcode	5
Excellon2 Graphic	6
MCL Graphic	7
EPS Graphic	8
DXF Graphic	9
System Line	10
System Rectangle	11
System Polygon	12
PLT Graphic	13
EMF Graphic	14
WLO Graphic	15
Text	16
System Drill	17
Barcode 39	18
Barcode CodaBar	19
Barcode 93	20
Barcode 128	21
Barcode 2 of 5	22
Barcode PostNET	23
Barcode UPC	24
Barcode EAN	25
DataMatrix	26
QRCode	27
Bitmap Graphic	28
WaitOnPortState I/O	29
SetPort I/O	30
TimeDelay	31
InfoMsgBox	32
Generic Motor Controller	33
XY Table Controller	34
Rotary Indexer Controller	35
Custom Axis Controller	36

Comments	Returns *S_OK* if function succeeds.

GetObjTypeString

Purpose	Gets a character string description of the object type.	
Implementation	HRESULT *GetObjTypeString* (int ObjIndex, BSTR* TypeString)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*TypeString*	The string representation of the object type.
Comments	Returns *S_OK* if function succeeds	

GetObjUsePensFlag

Purpose	Gets the pens flag from an object.	
Implementation	HRESULT *GetObjUsePensFlag* (int ObjIndex, int* Flag)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*Flag*	The pens flag. 0 = Do not use pens 1 = Use pens.
Comments	If the object contains pen information (usually in *.plt files), the Profile used to mark the object is selected by the current pen. Returns *S_OK* if function succeeds.	

GetObjVectorList

Purpose	Returns a SAFEARRAY containing a list of vector commands that describe an objects image.	
Implementation	HRESULT *GetObjVectorList* (int ObjIndex, SAFEARRAY(int)* ListArray)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*ListArray*	Pointer to the SAFEARRAY containing the vector commands.
Comments	The values in the safe array are listed in sets of three: *Parameter1*: Drawing command. 0 = MoveTo, 1 = LineTo, 2=SetPen, … *Parameter2*: X coordinate or Pen number *Parameter3*: Y coordinate If using C/C++, the programmer is responsible for releasing the SAFEARRAY memory when done with the array with calls to: SafeArrayUnaccessData(SAFEARRAY*); SafeArrayDestroy(SAFEARRAY*); The above calls are not necessary when using VisualBasic, as VB does the memory management automatically. See the sample source code files included with the weldMARK™ 1.0 installation package for details on how to implement calls using SAFEARRAYS. Returns *S_OK* if the function succeeds.	

GetPolygonAttributes

Purpose	Retrieve the attributes of a polygon object.	
Implementation	HRESULT *GetPolygonAttributes* (int ObjIndex, int* StartAngle, int* EndAngle, int* Sides)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*StartAngle*	The starting angular direction of the polygon. 0 (zero) degrees corresponds to the 12:00 position. Units: degrees
	EndAngle	The ending angular direction of the polygon. 360 degrees corresponds to the 12:00 position. Units: degrees
	NumSides	The number of straight line segments in the polygon.
Comments	Returns *S_OK* if the function succeeds.	
See Also	*SetPolygonAttributes*	

GetReadStatusWord

Purpose	Reads the status word from SP-ICE, RLC-USB, RLC-PCI.		
Implementation	HRESULT *GetReadStatusWord* (int CardNum, int Status)		
Parameters	*CardNum*	Index of scan head card Valid range: [0 to (number of cards-1)]	
Returns	*Status*	Bit 0 Load1	Indicates that list 1 is open for data input and all following list commands will be stored in it.
		Bit 1 Load2	Indicates that list 2 is open for data input and all following list commands will be stored in it.
		Bit 2 Ready1	Indicates that list 1 has been filled and closed.
		Bit 3 Ready2	Indicates that list 2 has been filled and closed.
		Bit 4 Busy1	Indicates that list 1 is being executed.
		Bit 5 Busy2	Indicates that list 2 is being executed.
		Bit 6 Busy	Indicates that one of the lists is being executed.
		Bit 7 LaserOn	Indicates that laser is on.
		Bit 8 Scan Complete	Indicates that scanning was finished either regularly at the end of the list or interrupted during execution.
		Bit 9	Previously used for indication that manual operation is switched on.
		Bit 10	Previously used for manual movement indicating that scanners are moved with control command.
		Bit 11 Marking Busy	Indicates that marking is not yet finished – this occurs when there are still commands in the output buffer to be processed even though all list commands have been interpreted.
		Bit 12	not used
		Bit 13	not used
		Bit 15 STOP Marking	The hardware signal STOP_MARK was received (through port C). Laser will be switched off list execution stopped.

GetScanCardCapacity

Purpose	Get the size of the scan cards memory buffer.
Implementation	HRESULT *GetScanCardCapacity* (int CardNum, int* Capacity)
Parameters	*CardNum* Index of scan head card Valid range: [0 to (number of cards-1)]
Returns	*Capacity* The number of available storage locations in the scan card memory buffer.
Comments	Use this command to determine how much vector list memory is available in the scan card. When using *MarkObj* or *MarkAllObj*, this command is not necessary, as the COM Server manages the moving of objects into the memory. When downloading objects for storage in the scan card with calls to *DownloadObj*, etc., there must be sufficient space to store all the objects to be downloaded. Calling *GetObjMemSize* can discover the memory space required by an object. Returns *S_OK* if the function succeeds.
See Also	*GetObjMemSize*, *DownloadObj*, *DownloadAllObj*

GetScanCardCount

Purpose	Get the number of scan cards installed and detected.
Implementation	HRESULT *GetScanCardCount* (int CardCount)
Returns	*CardCount* The number of installed and detected scan cards.
Comments	Use this command to determine how many scan cards are installed in the computer. An application should call this function when first initializing the COM Server, and exit if no cards are detected. Returns *S_OK* if the function succeeds.

GetScanCardInput

Purpose	Read a specific port on the SP-ICE, SP-ICE-2 or RLC card.	
Implementation	HRESULT *GetScanCardInput* (int CardNum, int* Offset, int* Word)	
Parameters	*CardNum*	Index of scan head card Valid range: [0 to (number of cards-1)]
	Offset	Address of the specific port address to read. See the SP-ICE card manual for more details. Valid range: [see SP-ICE card manual]
Returns	*Word*	The 16 bit data read in from the specified port is placed in Word. Only the lower 16-bits are valid, and the upper 16 bits should be ignored.
Comments	This command is valid only for the SP-ICE scan head cards. Returns *S_OK* if the function succeeds.	
See Also	*SetScanCardOutput*	

GetScanHeadCount

Purpose	Get the number of scan heads connected to a scan card.	
Implementation	HRESULT *GetScanHeadCount* (int CardNum, int* Count)	
Parameters	*CardNum*	Index of scan head card Valid range: [0 to (number of cards-1)]
Returns	*Count*	The number of scan heads detected.
Comments	The scan head must be connected to the scan card and have power applied to it for the scan card to detect it. Returns *S_OK* if the function succeeds.	
See Also	*GetScanCardCount*	

GetStartProcessBit

Purpose	Returns state of the StartProcess port on the Standard I/O card.	
Implementation	HRESULT *GetStartProcessBit* (int* Bit)	
Returns	*Bit*	The state of the StartProcessPort.
Comments	The Standard I/O card uses reverse logic, so a Bit value of 1 (one) indicates the port is at ground (true). Returns *S_OK* if the function succeeds.	
See Also	*SetIOSource*	

GetTextAttributes

Purpose	Get the attributes of a text object.	
Implementation	HRESULT *GetTextAttributes* (int ObjIndex, BSTR* FontName, int* FontType, int* Orientation, int* Kerning, int* Leading, int* Styles, int* ParagraphStyle, int* PulseCount)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*FontName*	The font name (e.g. Arial or Times New Roman).
	FontType	Font type flag. 0 = TrueType 1 = LaserFont
	Orientation	An integer value representing the physical orientation of singleline text objects. Orientation can contain one of the following values: 1 = Horizontal 2 = Vertical 3 = Radial
	Kerning	The added spacing between each character. Units: % (percent) of character width.
	Leading	The added spacing between each line in paragraph text. Units: % (percent) of character height.
	Styles	The font style (only for TT-Fonts). Styles can contain a combination of the following values: 0 = Normal text 1 = Bold 2 = Italics
	ParagraphStyle	The paragraph justification for multiline text objects. ParagraphStyle can be one of the following values: 0 = LeftJustify 1 = RightJustify 2 = CenterJustify
	PulseCount	The number of laser pulses fired at each dot using the current laser frequency and pulse width settings.
Comments	Only for TT fonts. Returns *S_OK* if the function succeeds.	
See Also	*SetTextAttributes*	

GetUserInWord

Purpose	Get the status of the USERIN ports on the Standard I/O card.
Implementation	HRESULT *GetUserInWord* (int* WordValue)
Returns	*WordValue*: Value of the word represented by the USERIN ports.
Comments	On the Standard I/O card, there are six bits that make up the USERIN ports. WordValue is a bitwise description of all six ports. For example, a WordValue of 0 indicates all the ports are set to false. AWordValue of 3 indicates that port 1 and port 2 are true, and the rest are false. A WordValue of 63 indicates all ports are set to true. The Standard I/O card uses reverse logic, so a true indicates the port is at ground. There must be a Standard I/O card installed for this function to succeed. Returns *S_OK* if the function succeeds.
See Also	GeScanCardInput

GetVectorGraphicAttributes

Purpose	Get the attributes of a vector graphic object.	
Implementation	HRESULT *GetVectorGraphicAttributes* (int ObjIndex, int* PulseCount)	
Parameters	*ObjIndex*	ObjIndex: Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*PulseCount*	The number of laser pulses fired at each dot using the current laser frequency and pulse width settings.
Comments	Not all vector graphic file formats support a dot entity. If the vector graphic file contains dot entities, the PulseCount parameter returns the value all dot entities have within the vector graphic. Returns *S_OK* if the function succeeds.	
See Also	*SetVectorGraphicAttributes*	

GoToXY

Purpose	Commands the mirrors to an X,Y coordinate.	
Implementation	HRESULT *GoToXY* (int CardNum, int X, int Y, double Jumpspeed)	
Parameters	*CardNum*	Index of scan head card Valid range: [0 to (number of cards-1)]
	X	The x-coordinate location to move the mirrors to. Valid range: [-32768 to 32767] bits
	Y	The y-coordinate location to move the mirrors to. Valid range: [-32768 to 32767] bits
	Jumpspeed	The speed at which the mirrors will jump to the X,Y coordinate.
Comments	You must call *GetBusyStatus* to determine if the COM Server is ready to execute this command before calling *GoToXY*. If the system is currently busy, the function will fail. The marking field is described using a Cartesian coordinate system, with (0,0) at the center of the field, (-32768, -32768) at the bottom left corner, and (32767, 32767) at the top right corner. Returns *S_OK* if the function succeeds.	
See Also	*GetBusyStatus*	

HomeAxes

Purpose	Homes the specified axes which HomeAxis flag is set to 1.	
Implementation	HRESULT *HomeAxes*(int HomeAxis1, int HomeAxis2, int HomeAxis3, int HomeAxis4)	
Parameters	*HomeAxis1…4*	Corresponds to OMS motor controller axis and represents a request to home it. 0 = home not required 1 = home required
Returns	Returns *S_OK* if the function succeeds.	
See Also	*HomeLTAxis*	

HomeLTAxis

Purpose	Homes the axis which controls the LT for specified scan head.	
Implementation	HRESULT *HomeLTAxis*(int ACard, int AHead)	
Parameters	*CardNum*	Index of scan head card Valid range: [0 to (number of cards -1)] bits
	HeadNum	Index of scan head Valid range: [0 to (number of cards -1)] bits
Returns	Returns *S_OK* if the function succeeds.	
See Also	*HomeAxes*	

IsIOCardInstalled

Purpose	Detects the PCI-DIO24H Standard I/O card.	
Implementation	HRESULT *IsIOCardInstalled* (int Flag)	
Returns	*Flag*	Flag indicating the presence of the I/O card. 0 = not installed 1 = installed
Comments	A proper connector with the CardID connected to ground and attached to the I/O card is required for the software to detect the I/O card. Refer to the weldMARK™ Reference Manual for details on installing and configuring the Standard I/O Card. Returns *S_OK* if the function succeeds.	

IsObjOutOfBounds

Purpose	Queries an object to determine if the whole object is within the markable boundaries of the marking field.	
Implementation	HRESULT *IsObjOutOfBounds* (int ObjIndex, int* OutOfBoundsFlag)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*OutOfBoundsFlag*	The OutOfBoundsFlag. 0 = in bounds 1 = out of bounds.
Comments	If OutOfBoundsFlag is 1 (one), the object, or part of the object is outside the marking field, and will not mark. The marking field is described using a Cartesian coordinate system, with (0,0) at the center of the field, (-32768, -32768) at the bottom left corner, and (32767, 32767) at the top right corner. Returns *S_OK* if the function succeeds.	

LoadJobFromFile

Purpose	Loads a compatible job file.	
Implementation	HRESULT *LoadJobFromFile* (BSTR FileName, int* JobIndex)	
Parameters	*FileName*	Full pathname to a valid job file. Valid types: [*.wmj]
Returns	*JobIndex*	The index of the newly loaded job.
Comments	If FileName is not found, has the wrong extension, or is corrupted, function will return an error. Returns *S_OK* if the function succeeds.	
See Also	*SaveJobToFile*	

LoadLaserConfigFile

Purpose	Loads a compatible laser config file.	
Implementation	HRESULT *LoadLaserConfigFile* (int CardNum, BSTR ConfigFile)	
Parameters	*CardNum*	Index of scan head card Valid range: [0 to (number of cards-1)]
	ConfigFile	Fully qualified path to a valid laser config file. Valid types: [*.cfg]
Comments	Laser config files are used to set up the scan card hardware for proper control of the laser output signals. If weldMARK™ 3 has been run on the target computer, and a laser type selected, you do not need to call this function. However, this function is available if you want to change laser types from the COM Server interface. Returns *S_OK* if the function succeeds.	
See Also	*GetLaserConfigFile*	

LoadLensCalFile

Purpose	Loads and binds a compatible lens correction file to a specific scan head.	
Implementation	HRESULT *LoadLensCalFile* (int CardNum, int HeadNum, BSTR CalFile)	
Parameters	*CardNum*	Index of scan head card. Valid range: [0 to (number of cards-1)]
	HeadNum	Index of scan head. Valid range: [0 to (number of heads-1)]
	CalFile	Full pathname of the lens correction file. Valid type: [*.gcd]
Comments	Lens correction files are used to correct for geometric distortions introduced by the XY scan head. If weldMARK™ 3 has been run on the computer, and a lens type selected, you do not need to call this function. However, this function is available if you want to change lens correction files from the COM Server interface. If a folder other than the standard lens correction file install folder is used, ensure the lens correction file is already present before restarting weldMARK™. Returns *S_OK* if the function succeeds.	
See Also	*GetLensCalFile*	

MarkAllObj

Purpose	Marks all objects in the Active Job.
Implementation	HRESULT *MarkAllObj* (int Orientation)
Parameters	*Orientation* Rotates the marked image relative to screen. Valid values: [0, 90,180, 270]
Comments	Each object will be marked using that objects settings for MarkMode and NumPasses. Use *SetObjMarkMode* and *SetObjNumPasses* to change these settings. You must call *GetBusyStatus* to determine if the COM Server is ready to mark before calling *MarkAllObj*. If the system is currently executing an object, the function will fail. All objects will be marked using the ProfileIndex of 0 (zero), unless the object supports pens, and the pens have been enabled. If pens are enabled, the Profile used will depend on the pens contained in the object. This function will not return until all objects have been marked. If you want control returned to your application immediately, use *MarkObj* instead. Returns *S_OK* if the function succeeds.
See Also	*TerminateMark*, *GetBusyStatus*, *MarkObj*, *MarkObjEx*, *SetObjProfile*, *GetObjProfile*, *SetObjMarkMode*, *SetObjNumPasses*, *SetObjUsePensFlag*

MarkObj

Purpose	Marks an object in the Active Job.	
Implementation	HRESULT *MarkObj* (int ObjIndex, int Orientation)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	Orientation	Rotates the marked image relative to screen. Valid values: [0, 90, 180, 270]
Comments	The object will be marked using the current object settings for MarkMode and NumPasses. Use *SetObjMarkMode* and *SetObjNumPasses* to change these settings. You must call *GetBusyStatus* to determine if the COM Server is ready to mark before calling *MarkObjEx*. If the system is currently executing an object, the function will fail. The object will be marked using the ProfileIndex of 0 (zero), unless the object supports pens, and the pens have been enabled. If pens are enabled, the Profile used will depend on the pens contained in the object.The active mark can be terminated at any time by calling *TerminateMark*. This function will return immediately. Returns *S_OK* if the function succeeds.	
See Also	*TerminateMark*, *GetBusyStatus*, *MarkObjEx*, *MarkAllObj*, *SetObjProfile*, *GetObjProfile*, *SetObjMarkMode*, *SetObjNumPasses*, *SetObjUsePensFlag*	

MarkObjEx

Purpose	Marks an object in the Active Job.	
Implementation	HRESULT *MarkObjEx* (int ObjIndex, int ProfileIndex, int Orientation)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	ProfileIndex	Index of Profile to use when marking. Valid range: [0 to 7]
	Orientation	Rotates the marked image relative to screen. Valid values: [0, 90, 180, 270]
Comments	The objects NumPasses and MarkMode settings are ignored when using this function, and the object will only mark once for each call to the function. You must call *GetBusyStatus* to determine if the COM Server is ready to mark before calling *MarkObj*. If the system is currently executing an object, the function will fail. The object will be marked using ProfileIndex, even if the object contains pens and the pens have been enabled. The active mark can be terminated at any time by calling *TerminateMark*. This function will return immediately. Returns *S_OK* if the function succeeds.	
See Also	*TerminateMark*, *GetBusyStatus*, *MarkObj*, *MarkAllObj*, *SetObjProfile*, *GetObjProfile*	

MoveObjInList

Purpose	Moves an object to another position within the ObjectList.	
Implementation	HRESULT *MoveObjInList* (int CurrIndex, int NewIndex)	
Parameters	*CurrIndex*	Current index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	NewIndex	New index of object in the ObjectList. Valid range: [0 to (number of objects-1)]
Comments	The object order in the ObjectList determines the order in which the objects will mark when calling *MarkAllObj* or *DownloadAllObj*. Returns *S_OK* if the function succeeds.	
See Also	*MarkAllObj*, *DownloadAllObj*	

NewBarcode

Purpose	Add a new barcode object to the current ObjectList in the Active Job.	
Implementation	HRESULT *NewBarcode* (int* NewObjIndex, BSTR ObjName, int CodeType, BSTR CharString)	
Parameters	*ObjName*	A name for the object. Valid length: [1 to 256 characters]
	CodeType	An integer indicating the type of barcode to add. Valid values: 0 = Code 39, Extended Code 39 1 = CodaBar 2 = Code 93 3 = Code 128 EAN/UCC 128 4 = Interleaved 2 of 5 (ITF) 5 = POSTNET (Zip+4, Zip+6) 6 = UPCA, UPCE 7 = EAN 8 EAN 13, BookLan 8 = DataMatrix (ECC200) 9 = Denso QR code 10= PDF417
	CharString	Valid string to represent with the barcode. [depends on barcode specification] Valid length: [depends on barcode specification]
Returns	*NewObjIndex*	The index of the new object in the ObjectList.
Comments	When the object is created, the DefaultProfile is applied to all Profiles in the object. Depending on the barcode type, different rules apply in specifying a valid value for CharString. Consult the specific barcode specifications for rules regarding string validity. Returns *S_OK* if the function succeeds.	

NewBitmap

Purpose	Add a new bitmap object to the current ObjectList in the Active Job.	
Implementation	HRESULT *NewBitmap* (int* NewObjIndex, BSTR ObjName, BSTR FileName)	
Parameters	*ObjName*	A name for the object. Valid length: [1 to 256 characters]
	FileName	A fully qualified path to a bitmap file. Valid types: [*.bmp, *.jpg, *.gif, *.pcx]
Returns	*NewObjIndex*	The index of the new object in the ObjectList.
Comments	When the object is created, the DefaultProfile is applied to all Profiles in the object. Returns *S_OK* if the function succeeds.	
See Also	*SetBitmapAttributes*	

NewDrill

Purpose	Add a new drill object to the current ObjectList in the Active Job.	
Implementation	HRESULT *NewDrill* (int* NewObjIndex, BSTR ObjName, int NumRows, int NumColumns, int NumPoints, int Duration)	
Parameters	*ObjName*	A name for the object. Valid length: [1 to 256 characters]
	NumRows	Numbers of rows in the point array. Valid range: [1 to 100]
	NumColumns	Numbers of columns in the point array. Valid range: [1 to 100]
	NumPoints	Total number of points in the point array. Valid range: [1 to 10000]
	Duration	The length of time the laser is turned on at each point. Valid range: [10 to 120,000,000] µSec
Returns	*NewObjIndex*	The index of the new object in the ObjectList.
Comments	When the object is created, the DefaultProfile is applied to all Profiles in the object. Returns *S_OK* if the function succeeds.	

NewJob

Purpose	Add a new job to the JobList.	
Implementation	HRESULT *NewJob* (int* NewJobIndex, BSTR FileName)	
Parameters	*FileName*	A fully qualified filename for the new job. Valid length: [1 to 511 characters]
Returns	*NewJobIndex*	The index of the new job in the JobList.
Comments	The new job is created empty (containing no objects). FileName is the filename used in calls to *SaveJobToFile*. It must have a .wmj extension. Returns *S_OK* if the function succeeds.	

NewLine

Purpose	Add a new line object to the current ObjectList in the Active Job.	
Implementation	HRESULT *NewLine* (int* NewObjIndex, BSTR ObjName)	
Parameters	*ObjName*	A name for the object. Valid length: [1 to 256 characters]
Returns	*NewObjIndex*	The index of the new object in the ObjectList.
Comments	When the object is created, the DefaultProfile is applied to all Profiles in the object. Returns *S_OK* if the function succeeds.	

NewPolygon

Purpose	Add a new polygon object to the current ObjectList in the Active Job.	
Implementation	HRESULT *NewPolygon* (int* NewObjIndex, BSTR ObjName, int NumSides, int StartAngle, int EndAngle)	
Parameters	*ObjName*	A name for the object. Valid length: [1 to 256 characters]
	NumSides	The number of straight line segments in the polygon. Valid range: [3 to 10000]
	StartAngle	The start direction when drawing the polygon. Valid range: [0 to 360]
	EndAngle	The end direction when drawing the polygon. Valid range: [0 to 360]
Returns	*NewObjIndex*	The index of the new object in the ObjectList.
Comments	When the object is created, the DefaultProfile is applied to all Profiles in the object. Returns *S_OK* if the function succeeds.	

NewRect

Purpose	Add a new rectangle object to the current ObjectList in the Active Job.	
Implementation	HRESULT *NewRect* (int* NewObjIndex, BSTR ObjName)	
Parameters	*ObjName*	A name for the object. Valid length: [1 to 256 characters]
Returns	*NewObjIndex*	The index of the new object in the ObjectList.
Comments	When the object is created, the DefaultProfile is applied to all Profiles in the object. Returns *S_OK* if the function succeeds.	

NewText

Purpose	Add a new text object to the current ObjectList in the Active Job.	
Implementation	HRESULT *NewText* (int* NewObjIndex, BSTR ObjName, BSTR FontName, BSTR CharString, int Paragraph)	
Parameters	*ObjName*	A name for the object. Valid length: [1 to 256 characters]
	FontName	The text objects font. Valid length: [1 to 256 characters]
	CharString	The string the text object represents. Valid length: [1 to 511 characters]
	Paragraph	A flag to indicate paragraph text. Valid values: 0 = singleline object 1 = multiline object
Returns	*NewObjIndex*	The index of the new object in the ObjectList.
Comments	When the object is created, the DefaultProfile is applied to all Profiles in the object. FontName must be a LaserFont installed on the machine, or a TrueType font installed in Windows or the Arial font is substituted. A value of 0 in Paragraph instructs the input parser to ignore embedded carriage return/line feed pairs. Returns *S_OK* if the function succeeds.	

NewVectorGraphic

Purpose	Add a new vector graphic object to the current ObjectList in the Active Job.	
Implementation	HRESULT *NewVectorGraphic* (int* NewObjIndex, BSTR ObjName, BSTR FileName)	
Parameters	*ObjName*	A name for the object. Valid length: [1 to 256 characters]
	FileName	A fully qualified filename pointing to the vector graphics file to import. Valid types: [*.wlo, *.plt, *.emf [1)], *.wmf [1)], *.dxf, *.ex2]
Returns	*NewObjIndex*	The index of the new object in the ObjectList.
Comments	When the object is created, the DefaultProfile is applied to all Profiles in the object. Returns *S_OK* if the function succeeds.	

1) This file formats can contain not only vector graphics but also bitmap images. If the bitmap images should be imported, in the registry the following flag must be set:
HKLM\Software\RAYLASE\weldMARK\ObjDefaults\vectorGraphic\ImportExtractedBitmap
If the flag is not present, 0 (false) is assumed.

OffsetObj

Purpose	Move an object within the marking field.	
Implementation	HRESULT *OffsetObj* (int ObjIndex, int XOffset, int YOffset)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	XOffset	The amount to move the object along the x-axis. Valid range: unlimited integer value. Keep within [-31768 to 32767]
	YOffset	The amount to move the object along the y-axis. Valid range: unlimited integer value. Keep within [-31768 to 32767]
Comments	The marking field is described using a Cartesian coordinate system, with (0,0) at the center of the field, (-32768, -32768) at the bottom left corner, and (32767, 32767) at the top right corner. Returns *S_OK* if the function succeeds.	

ReleaseMarker

Purpose	Detaches from and closes the marker libraries.
Implementation	HRESULT *ReleaseMarker* (void)
Comments	weldMARK™ and the COM Server cannot access the scan card hardware at the same time. If you have loaded the COM Server and want to access the scan card from weldMARK™, call *ReleaseMarker*. When you want to gain access to the scan card with the COM Server again, call *AttachToMarker*. Returns *S_OK* if the function succeeds.
See Also	*AttachToMarker*

RotateObj

Purpose	Rotates an object about its center.	
Implementation	HRESULT *RotateObj* (int ObjIndex, float Angle)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	Angle	The relative amount to rotate the object. Valid range: [-360.00 to 360.000] degrees
Comments	Positive values of Angle rotate the object clockwise. Returns *S_OK* if the function succeeds.	
See Also	*RotateObjEx*	

RotateObjEx

Purpose	Rotates an object about a coordinate center.	
Implementation	HRESULT *RotateObjEx* (int ObjIndex, float Angle, int XCenter, int YCenter)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	Angle	The relative amount to rotate the object. Valid range: [-360.00 to 360.000] degrees
	XCenter	The center of rotation in the x-axis. Valid range: [-2,147,483,648 to 2,147,483,647]
	YCenter	The center of rotation in the y-axis. Valid range: [-2,147,483,648 to 2,147,483,647]
Comments	Positive values of Angle rotate the object clockwise. Returns *S_OK* if the function succeeds.	
See Also	*RotateObj*	

SaveJobToFile

Purpose	Saves the Active Job to a file.	
Implementation	HRESULT *SaveJobToFile* (BSTR FileName, BSTR AppVersion, BSTR TodaysDate, BSTR AppName, BSTR CompanyName)	
Parameters	*FileName*	Fully qualified path to the file to save. Valid length: [1 to 256 characters]
	AppVersion	A string for the version number. Valid length: [1 to 256 characters]
	TodaysDate	A string with the date. Valid length: [1 to 256 characters]
	AppName	A string for the name of your Application. Valid length: [1 to 256 characters]
	CompanyName	A string for your company name. Valid length: [1 to 256 characters]
Comments	Any changes made to objects or the job after it was loaded will be saved. FileName must be a fully qualified path to the job file. Use AppVersion to save the client applications version number to the file. Use TodaysDate to save a string of the current date (in any format) to the file. Use AppName to save the client applications name to the file. Use CompanyName to save the client applications company name to the file. Returns *S_OK* if the function succeeds.	
See Also	*LoadJobFromFile*	

ScaleObj

Purpose	Scales an object from its center.	
Implementation	HRESULT *ScaleObj* (int ObjIndex, float XScale, float YScale, int XMirror, int YMirror)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	XScale	Amount to scale the x-axis. Valid range: [>0]
	YScale	Amount to scale the y-axis. Valid range: [>0]
	XMirror	Setting to 1 (one) will cause the object to mirror itself in x-axis. Valid values: [0, 1]
	YMirror	Setting to 1 (one) will cause the object to mirror itself in y-axis. Valid values: [0, 1]
Comments	Returns *S_OK* if the function succeeds.	

ScanCardCommand

Purpose

Add a list command to the scan card list buffer.

Implementation

HRESULT *ScanCardCommand* (int CardNum, int OpCode, int AParam, int BParam, BSTR Buffer)

Parameters

CardNum — Index of scan head card.

Valid range: [0 to (number of cards-1)]

OpCode — An operation code used to indicate which list command to send to the scan card. Can be one of the following values:

Function	OpCode	Aparam	BParam	Buffer	Typ
Jump_abs	0	Xcoordinate	Ycoordinate	NULL	LC
Laser_on	1	Duration	0	NULL	LC
Long_delay	2	Duration	0	NULL	LC
Mark_abs	3	Xcoordinate	Ycoordinate	NULL	LC
Pola_abs	4	Xcoordinate	Ycoordinate	NULL	LC
Polb_abs	5	Xcoordinate	Ycoordinate	NULL	LC
Polc_abs	6	Xcoordinate	Ycoordinate	NULL	LC
Set_Mark_Parameters_List	10	StepPeriod	StepSize	NULL	LC
Write_Da_List	11	PortNumber	Value	NULL	LC
Write_Port_List	12	PortNumber	Value	NULL	LC
Output_To_File	101	0	0	FileName	CC
Copy_File_To_Target_Disk	102	0	0	FileName	CC
Output_To_File	103	0	0	NULL	CC
Set_Start_List_1	104	0	0	NULL	CC
Set_Start_List_2	105	0	0	NULL	CC
Wait_For_External_Start	106	0	0	NULL	LC
Loop_To_Start_List	107	ListNum[1)]	0	NULL	LC
Set_End_Of_List	108	0	0	NULL	LC
Execute_List_1	109	0	0	NULL	CC
Execute_List_2	110	0	0	NULL	CC
Set_Active_Card	112	CardNum+1	0	NULL	CC
LoadCorrFileFromTargetDsk	114	0	0	FileName	CC
Set_Mode	116	Mode	0	NULL	CC
Delete_File_On_Target_Disk	117	0	0	NULL	CC
Wait_For_Counter_Value_Ex	118	Value	0	NULL	LC
Reset_Jump_List	119	Xcoordinate	Ycoordinate	NULL	LC
Mark_Immediately	120	0	0	NULL	LC
Modify_Gain	121	XGain	YGain	NULL	CC
Modify_Offset	122	XOffset	YOffset	NULL	CC
Clear_Scan_Complete	123	0	0	NULL	CC

AParam — The first parameter associated with OpCode. See above. Check the scan card documentation for valid value ranges.

BParam — The second parameter associated with OpCode. See above. Check the scan card documentation for valid value ranges.

	Buffer	The character string parameter associated with OpCode. See above. Check the scan card documentation for valid value ranges.
Comments	Use this command to gain direct control over the list commands sent to the scan card. Normally, scan card list commands are interspersed with Object vector list commands, using *ScanCardCommand* and *DownloadObj* or *DownloadAllObj* to create a customized job loaded into the scan card hardware. After the list has been set up, *ScanCardExecute* is called. Refer to the scan card documentation for a complete description of each command. Returns *S_OK* if the function succeeds.	
	Modify_Gain and *Modify_Offset* commands affect only the first active controller card and not any connected slave controller cards.	
	XGain and *YGain* parameters in the *Modify_Gain* command are set as integer values in [1/1000 of a %] units. The current gain values are modified by the specified percentage. Example: A gain change of +1% is set with a value of 1000 and a gain change of -1% is set with a value of -1000.	
	XOffset and *YOffset* parameters in the *Modify_Offset* command are positive or negative integers in bits, that are added to the original scan head offsets. Please see the *OffsetObj* command for valid ranges.	
See Also	*DownloadObj*, *DownloadAllObj*, *ScanCardExecute*	

1) ListNum: Valid values: [1 or 2]

LC = List Command, CC = Control Command

ScanCardExecute

Purpose	Execute the indicated list buffer in the scan card.	
Implementation	HRESULT *ScanCardExecute* (int CardNum, int ListNum)	
Parameters	*CardNum*	Index of scan head card. Valid range: [0 to (number of cards-1)]
	ListNum	The list buffer to execute. Valid values: [1 or 2]
Comments	Use this command to execute a previously built set of list commands. Normally, *ScanCardCommand* is called, interspersed with Object vector lists, using *DownloadObj* or *DownloadAllObj* to create a customized job loaded into the scan card hardware. After the list has been set up, *ScanCardExecute* is called. There are two lists available in the scan card buffer. Returns *S_OK* if the function succeeds.	
See Also	*DownloadObj*, *DownloadAllObj*, *ScanCardCommand*	

SetActiveJob

Purpose	Set a job within the JobList to the Active Job.	
Implementation	HRESULT *SetActiveJob* (int JobIndex)	
Parameters	*JobIndex*	Index of job to make the Active Job. Valid range: [0 to (number of jobs-1)]
Comments	Use *GetJobCount* to discover the total number of jobs currently loaded in memory. All functions that reference and use Objects work with the Active Job. Returns *S_OK* if the function succeeds.	
See Also	*GetJobCount*	

SetBarcodeAttributes
SetBarcodeAttributesEx

Purpose	Set the attributes of a barcode object.	
Implementation	HRESULT *SetBarcodeAttributes* (int ObjIndex, int WidthReduce, int NarrowToWide, int QuietZone, int Preferences, int DotMatrix, int Pixels, int PulseCount)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	WidthReduce	The amount of reduction in the width of all bars. Valid for 1D codes only. Valid range: [-99 to 99] % of bar width
	NarrowToWide	The change in width ratio of the narrow and wide bars from default. Valid for 1D codes only. For PDF417, represents the aspect ratio of the height and width of the entire barcode. Valid range: [-20 to 30] % of bar width
	QuietZone	When inverting a barcode, the amount of quiet space to surround the code. Set to 0 to disable inverting. Valid range: 1D = [0 to 50] % of code width. DataMatrix, QR code, PDF417 = [0 to 50] # of cells

Preferences: Use to set options depending on the symbology type.

To use one or more of the Preferences options, perform a logical OR using the desired preferences constants and pass the result in the Preferences parameter.

Constant	Symbology	Function
1	QR code	Use Model1
2	QR code	Use Model2
4	QR code	Use MicroQR
8	QR code	Use Error correction level L (7%)
16	QR code	Use Error correction level M (15%)
32	QR code	Use Error correction level Q (25%)
64	QR code	Use Error correction level H (30%)
2	PDF417	Use Security Level 0
4	PDF417	Use Security Level 1
8	PDF417	Use Security Level 2
32	PDF417	Use TruncatedPDF
64	PDF417	Use Security Level 3
128	PDF417	Use Security Level 4
256	PDF417	Use Security Level 5
512	PDF417	Use Security Level 6
1024	PDF417	Use Security Level 7
2048	PDF417	Use Security Level 8
4096	PDF417	Use Security Level 9 (Automatic)
8192	PDF417	Use Compaction Mode 0
16384	PDF417	Use Compaction Mode 1
32768	PDF417	Use Compaction Mode 2
65536	PDF417	Use Compaction Mode 3
131072	PDF417	Use Compaction Mode 4
16	DataMatrix	Use Standard ASCII
131072	DataMatrix	Use 8 X 18 Format
262144	DataMatrix	Use 8 X 32 Format
393216	DataMatrix	Use 12 X 26 Format
524288	DataMatrix	Use 12 X 36 Format
655360	DataMatrix	Use 16 X 36Format
786432	DataMatrix	Use 16 X 48 Format
1048576		Mark cells individually
2097152	2d Matrix	Request to automatically enlarge code if necessary
16777216	DataMatrix Code	Generate fixed 10 x 10
33554432	DataMatrix Code	Generate fixed 12 x 12
50331648	DataMatrix Code	Generate fixed 14 x 14
67108864	DataMatrix Code	Generate fixed 16 x 16
83886080	DataMatrix Code	Generate fixed 18 x 18
100663296	DataMatrix Code	Generate fixed 20 x 20
117440512	DataMatrix Code	Generate fixed 22 x 22
134217728	DataMatrix Code	Generate fixed 24 x 24
150994944	DataMatrix Code	Generate fixed 26 x 26
167772160	DataMatrix Code	Generate fixed 32 x 32

	184549376	DataMatrix Code	Generate fixed 36 x 36
	201326592	DataMatrix Code	Generate fixed 40 x 40
	218103808	DataMatrix Code	Generate fixed 44 x 44
	234881024	DataMatrix Code	Generate fixed 48 x 48
	251658240	DataMatrix Code	Generate fixed 52 x 52
	268435456	DataMatrix Code	Generate fixed 64 x 64
	285212672	DataMatrix Code	Generate fixed 72 x 72
	301989888	DataMatrix Code	Generate fixed 80 x 80
	318767104	DataMatrix Code	Generate fixed 88 x 88
	335544320	DataMatrix Code	Generate fixed 96 x 96
	352321536	DataMatrix Code	Generate fixed 104 x 104
	369098752	DataMatrix Code	Generate fixed 120 x 120
	385875968	DataMatrix Code	Generate fixed 132 x 132
	402653184	DataMatrix Code	Generate fixed 144 x 144
	32	Code 39	Use Full ASCII
	64	Code 39	Use HIBC
	64	Code 128	Use EAN/UCC128
	64	EAN/BookLan	Use BookLan
	128	Code 39	Use Check digit Modulo 43
		Interleaved 2 of 5	Use Check digit Modulo 10
		CodaBar	Use Check digit Modulo 16
		EAN/UCC 128	Use Check digit Modulo 10
	DotMatrix	The dot matrix flag. Set to 1 (one) to enable dot matrix mode, set to 2 (two) to enable circle dot mode. If parameter is set to 1 or 2, please be sure that filling is disactivated (*SetObjMarkFillFlag* = 0). Valid values: [1 or 2]	
	Pixels	For dot matrix mode, depends on barcode type: 1D codes: The spacing between adjacent pixels. Valid range: [1 to 32767] bits. 2D codes: The number of rows and columns in the pixel array in each cell. Valid range: [1 to 100]	
	PulseCount	The number of laser pulses fired at each dot using the current laser frequency and pulse width settings. Valid range: [1 to 10000]	
Comments	Returns *S_OK* if the function succeeds.		
See Also	*GetBarcodeAttributes, GetBarcodeAttributesEx*		

SetBitmapAttributes

Purpose	Sets the attributes of a bitmap object.[1)]	
Implementation	HRESULT *SetBitmapAttributes* (int ObjIndex, BSTR FileName, int PixelSep, int Contrast, int Brightness, int InvertPixels, int SkipBlack, int BlackCorners, int ErrorDiffusion)	
Parameters	*ObjIndex*	Index of object in the ObjectList. Valid range: [0 to (number of objects-1)]
	FileName	Fully qualified filename to use as objects graphics source. Valid types: [*.bmp, *.jpg, *.pcx, *.gif]
	PixelSep	Distance between adjacent pixels. Valid range: [1 to 10000] bits
	Contrast	A relative value affecting the range between the darkest and lightest pixels. Valid range: [-100 to 500]
	Brightness	A relative value affecting the overall brightness of all pixels. Valid range: [-100 to 500]
	InvertPixels	Flag indicating whether the pixels are inverted (black to white). 0 = not inverted, 1 = inverted Valid values: [0, 1]
	SkipBlack	Flag indicating whether black pixels are jumped over when marking the bitmap. 0 = Do not skip, 1 = Skip black pixels Valid values: [0, 1]
	BlackCorners	Flag indicating what color to make pixels in the corners if the pixel has been rotated to an angle other that 90, 180 or 270. 0 = white, 1 = black Valid values: [0, 1]
	ErrorDiffusion	Flag indicating whether the Error Diffusion algorithm has been applied to the bitmap. 0 = no error diffusion, 1 = error diffusion applied. Valid values: [0, 1]
Comments	Returns *S_OK* if function succeeds.	
See Also	*GetBitmapAttributes*	

1) Since weldMARK™ v2.0.0.98a there is a registry entry to set the raster mode to single or bidirectional.
HKCU\Software\RAYLASE\weldMARK\objDefaults\Bitmap raster REG_DWORD
0 -> bidirectional; 1 -> single raster mode
Since the value is read every time before a command is executed, it is possible to set or change the the raster mode befor calling NewBitmap command.

SetBitmapGrayScaleMode

Purpose	Sets the mode defining the way in which the laser power is controlled when marking a grayscale bitmap. The result achieves with different modes depend on the type of laser. There are three selectable modes (see below).	
Implementation	HRESULT *SetBitmapGrayScaleMode* (int ObjIndex, int Mode)	
Parameters	*ObjIndex*	Index of object in the ObjectList. Valid range: [0 to (number of objects-1)]
	Mode	Value of the bitmap algorithm to be used for the-object..
Comments	The selectable modes are:	

ID	Mode (Bitmap Algorihm)	Usable for Laser types
0	POINT_AND_SHOOT_ALG	All (usually CO_2): Grayscale value of pixels set by laser-on time.
4	ANALOG_POWER_ALG	Nd:YAG: Grayscale value of pixel set by analog laser power interface.
5	DIGITAL_POWERSET_ALG	Nd:YAG: Grayscale value of pixel set by digital laser power interface.
9	PWM_ALG	For CO2 and lasers that can be modulated on higher frequencies

Do not use any Mode IDs other than 0, 4, 5 or 9.

The ErrorDiffusion mode can be set using the function *SetBitmapAttributes*.

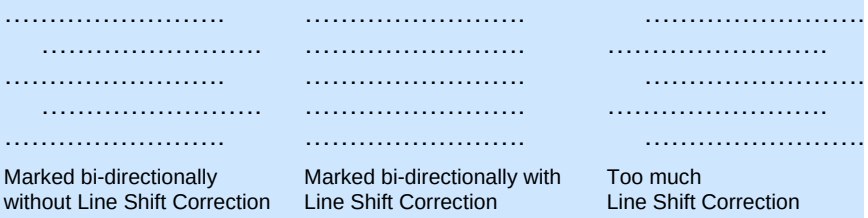
Returns *S_OK* if function succeeds.

See Also	*GetBitmapGrayScaleMode, GetBitmapAttributes, SetBitmapAttributes*	

SetBmpEndOfLineDelay

Purpose	Insert a delay at the end of each pixel line after jumping to the next lines start position. This is one possibility to improve marking quality.	
Implementation	HRESULT *SetBmpEndOfLineDelay* (int ObjIndex, int Delay)	
Parameters	*ObjIndex*	Index of object in the ObjectList. Valid range: [0 to (number of objects-1)]
	Delay	Valid range: [0 to 20000] µSec
Comments	Returns *S_OK* if function succeeds.	
See Also	*GetBmpEndOfLineDelay*	

SetBmpLineShiftCorrection

Purpose	Bitmap graphics are marked line by line. In bi-directional mode the marking speed can be increased considerably. Due to mechanical inertia and laser specific delay, line offset can occur which can be corrected via parameter *Set_BmpLineShiftCorrection.* Example: 
Implementation	HRESULT *SetBmpLineShiftCorrection* (int ObjIndex, int* Correction)
Parameters	*ObjIndex* Index of object in the ObjectList. Valid range: [0 to (number of objects-1)] *Correction* Valid range: [0 to 65500] bits
Comments	Returns *S_OK* if function succeeds.
See Also	*GetBmpLineShiftCorrection*

SetBmpSkippedPixelTreshold

Purpose	The “Pixel Treshold” value enables to specify a threshold value that SP_ICE card uses to encounter bitmap areas that are not marked but skipped. This value depends on the originally imported bitmap and the cost of quality that the user is willing to accept to enhance the marking speed. Above the bitmap dependent threshold, pixels with lower values are not marked but skipped. This however is only done if at least 3 pixels in a row are below pixel threshold value. “Pixel Treshold” values 0 and 1 means that there will be no skipping of unmarked areas at all, so that every single pixel is marked by the laser.
Implementation	HRESULT *SetBmpSkippedPixelTreshold* (int ObjIndex, int MinPixel)
Parameters	*ObjIndex* Index of object in the ObjectList. Valid range: [0 to (number of objects-1)] *MinPixel* Value ≤ 1 No skip of white pixels Value ≥ 2 Skips processing of pixel wit grey-scale value < the defined threshold value
Comments	Returns *S_OK* if function succeeds.
See Also	*GetBmpSkippedPixelTreshold*

SetBusyReadyBit

Purpose	Set the Busy/Ready port on the Standard I/O card.	
Implementation	HRESULT *SetBusyReadyBit* (int Bit)	
Parameters	*Bit*	The Busy/Ready flag. Valid values: [0 or 1]
Comments	When using the COMServer, the Busy/Ready port on the Standard I/O card does not change automatically. The programmer must use this command to change the state of the port. The Standard I/O card uses reverse logic, so a Bit value of 1 (one) will set the port to ground (true). Returns *S_OK* if the function succeeds.	

SetDefaultProfile

Purpose	Change the Default Profile settings applied to all new objects.	
Implementation	HRESULT *SetDefaultProfile* (int ProfileIndex, int Mode, int PassCount, double Markspeed, double Jumpspeed, int Jumpdelay, int Markdelay, int Polygondelay, float Laserpower, int Laseroffdelay, int Laserondelay, int TAxis, double T1, int T2, int Unused, int Varijumpdelay, int Varijumplength, int Wobblesize, double Wobblefrequency, int Autosegmentation, int Varipolydelay)	
Parameters	*ProfileIndex*	Index of the Profile. Valid range: [0 to 7]
	Mode	See *SetObjMarkMode* for a description of this parameter. Valid range: [0 to 4]
	PassCount	See *SetObjNumPasses* for a description of this parameter. Valid range: [>0]
	MarkSpeed	Defines the speed of the laser spot while marking. Valid range: [0 to 30000] bits/mm
	Jumpspeed	Defines the speed at which the mirrors jump to the next marking vector. Valid range: [50 to 30000] bits/mm
	Jumpdelay	Defines the delay after a jump and before the next marking vector starts. Valid range: [0 to 65500] µSec
	Markdelay	Defines the delay between a marking vector and a jump vector. Valid range: [30 to 20000] µSec
	Polygondelay	Defines the delay between contiguous marking vectors. Valid range: [0 to 20000] µSec
	Laserpower	Defines the programmed laser power for non CO_2-type lasers Valid range: [0 to 100] % (percent) Laserpower for CO_2- type lasers is defined as: Duty Cycle (%) = 0.1 × T2 [µs] × T1 [kHz]
	Laseroffdelay	Defines the delay after the last marking vector finishes and the laser is turned off. Valid range: [0-65535] µSec
	Laserondelay	Defines the delay after a marking vector starts and the laser is turned on. Valid range: [0-65535] µSec

	TAxis	Defines the Z position of the object. +Z is toward the scan head and –Z away from the scan head. Position is defined in bits and the same calibration factor is used as for x and y. Z field size is limited by the available Linear Translator movement. Values for Zmin and Zmax are defined in the scan head configuration file and can be read with *GetLensCalFactorEx* command. Units: bits Valid range: [Zmin to Zmax]
	T1	Defines the frequency of the laser modulation signal. Valid range: [0 to 250] kHz
	T2	Defines the pulse width of the laser modulation signal. Valid range: [1 to 65535] µSec
	Unused	Set to 0.
	Varijumpdelay	Defines the delay after a jump and before the next marking vector starts if variable jump delay is in effect. Set to 0 (zero) to disable variable jump delay. Valid range: 0 = not active, [1-30000] µSec
	Varijumplength	Defines the length of a vector, at which any vector that is longer will use the Varijumpdelay parameter, and any vector that is shorter will use the Jumpdelay parameter. Valid range: 0 = not active, [1-30000] bits
	Wobblesize	The diameter of the circle created when the spot is dithered. Set to 0 (zero) to disable Valid range: 0 = not active, [1-5000] bits
	Wobblefrequency	The frequency of the laser spot as it dithers around the circle defined in Wobblesize. Active only when wobble size > 0. Valid range: [0-6000] Hz (cycles per second)
	Unused	Reserved. Set to 0.
	Varipolydelay	Reserved. Set to 1.
Comments	The default Profile can store eight individual profiles. Objects also have eight profiles stored. The Mode and PassCount parameters are global to all eight individual profiles. Returns *S_OK* if the function succeeds.	
See Also	*GetObjProfile*, *SetDefaultProfile*, *GetDefaultProfile*, *SetObjMarkMode*, *SetObjNumPasses*	

SetDrillAttributes

Purpose	Sets the attributes of a drill object.	
Implementation	HRESULT *SetDrillAttributes* (int ObjIndex, int Rows, int Columns, int NumPoints, int Duration)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	Rows	The number of rows in the point array. Valid range: [1 to 100]
	Columns	The number of columns in the point array. Valid range: [1 to 100]
	NumPoints	The total number of points in the point array. Valid range: [1 to 10000]
	Duration	Number of pulses the laser will fire at each point. Valid range: [1 to 10000]
Comments	Returns *S_OK* if function succeeds.	
See Also	*GetDrillAttributes*	

SetMarkInProgressBit

Purpose	Set the Mark in Progress port on the Standard I/O card, and the scan head card.	
Implementation	HRESULT *SetMarkInProgressBit* (int Bit)	
Parameters	*Bit*	The Mark in Progress flag. Valid values: [0 or 1]
Comments	When using the COMServer, the Mark In Progress port does not change automatically; the programmer must use this command to change the state of the port. The scan card hardware must support user I/O for the MarkInProgress signal to be available on the scan head card. Returns *S_OK* if the function succeeds.	

SetMOTFConfig

Purpose	Sets the Mark on the Fly configuration parameters.	
Implementation	HRESULT *SetMOTFConfig* (int CardNum, int MOTFFlag, int EncoderSimFlag, double EncoderCal, int MarkStartDelay, double MOTFAngle)	
Parameters	*CardNum*	Index of scan head card. Valid range: [0 to (number of cards-1)]
	MOTFFlag	The Mark on the Fly flag. Set to 0 (zero) to disable Mark on the Fly, or set to 1 (one) to enable. Valid values: [0 or 1]
	EncoderSimFlag	The encoder simulation flag. To simulate an encoder, set to 1 (one). Valid values: [0 or 1]
	EncoderCal	The calibration factor of the encoder. Valid range: [0 to 65000] counts/mm
	MarkStartDelay	The number of encoder counts to wait before starting the mark. Valid range: [0 to 1500] counts
	MOTFAngle	The angular orientation of the moving part with respect to the x-axis. Valid range: [0 to 360] degrees
Comments	For a part that is moving along the x-axis in the direction of increasing x, MOTFAngle is 0. For a part that is moving along the y-axis in the direction of increasing y, MOTFAngle is 90, etc. Returns *S_OK* if the function succeeds.	
See Also	*GetMOTFConfig*	

SetObjCharString

Purpose	Set the String value of a text or barcode object.	
Implementation	HRESULT *SetObjCharString* (int ObjIndex, BSTR CharString)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	CharString	The new character string Valid length: [1 to 256 characters]
Comments	After setting a new string, the size and position of the object may change. Use *GetObjRect* to discover the objects current position and size. Returns *S_OK* if the function succeeds.	
See Also	*GetObjCharString*, *GetObjRect*	

SetObjFill
SetObjFillEx

Purpose	Sets fill parameters of an object.	
Implementation	HRESULT *SetObjFill* (int ObjIndex, int FillSpacing, int FillOffset, int Slope1, int Slope2, int FillStyle)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	FillSpacing	The distance between adjacent fill lines. Valid range: [1 to 32767] bits
1)	*FillOffset*	The distance between any endpoint of filling hatchlines and the outlines of the object. Valid range: [1 to 32767] bits
	Slope1	The angle with respect to the x-axis of the first set of fill lines. Valid range: [-90 to 90] degrees
	Slope2	The angle with respect to the x-axis of the second set of fill lines (for crosshatch). Applicable only if *FillingStyle* is set to 1. Valid range: [-90 to 90] degrees
	FillStyle	The fill style. 0 = parallel lines 1 = crosshatch 2 = bidirectional 3 = bidirectional and crosshatch 6 = bidirectional using meanderfill 7 = bidirectional+crosshatch using meanderfill Valid values:: [0, 1, 2, 3, 6, 7]
Comments	Only objects with closed paths can be filled. Returns *S_OK* if the function succeeds.	
See Also	*GetObjFill*	

1) SetObjFillEx only

SetObjGraphicFile

Purpose	Sets a new graphics source file for an object.	
Implementation	HRESULT *SetObjGraphicFile* (int ObjIndex, BSTR GraphicFile)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	GraphicFile	A fully qualified filename pointing to the vector graphics file. Valid types: [*.wlo, *.plt, *.emf, *.wmf, *.dxf, *.ex2]
Comments	After setting a new source file, the size and position of the object may change. Use *GetObjRect* to discover the objects current position and size. Returns *S_OK* if the function succeeds.	
See Also	*GetObjGraphicFile*, *GetObjRect*	

SetObjMarkFillFlag

Purpose	Sets the MarkFill flag of an object.	
Implementation	HRESULT *SetObjMarkFillFlag* (int ObjIndex, int MarkFillFlag)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	MarkFillFlag	The fill flag. Set to 1 (one) to enable fill marking. Valid values: [0, 1]
Comments	If the flag is set, the objects system generated fill will mark. If the object has no fill, this function has no effect. Returns *S_OK* if the function succeeds.	

SetObjMarkMode

Purpose	Sets the current MarkMode of an object.	
Implementation	HRESULT *SetObjMarkMode* (int ObjIndex, int Mode)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	Mode	The current MarkMode, which can have the following values: 0 = Mark object once. NumPasses is ignored. 1 = Mark object using the value of NumPasses. 2 = Mark object with two passes, where: Pass1 uses Profile0 Pass2 uses Profile1 3 = Mark object with three passes, where: Pass1 uses Profile0 Pass2 uses Profile1 Pass3 uses Profile2 4 = Mark object with four passes, where: Pass1 uses Profile0 Pass2 uses Profile1 Pass3 uses Profile2 Pass4 uses Profile3 Valid values: [0 ,1, 2,3,4]
Comments	If the Mode is set to 2, 3 or 4, the use of pens is automatically disabled. Use *SetObjNumPasses* to set the NumPasses value of an object. Use *SetObjProfile* to change the profile settings of an object. Returns *S_OK* if function succeeds.	
See Also	*SetObjNumPasses*, *SetObjProfile*	

SetObjMarkOutlineFlag

Purpose	Sets the MarkOutline flag of an object.	
Implementation	HRESULT *SetObjMarkOutlineFlag* (int ObjIndex, int MarkOutlineFlag)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	MarkOutlineFlag	The outline flag. Set to 1 (one) to enable outline marking. Valid values: [0, 1]
Comments	If the flag is set, the objects outline will mark. Returns *S_OK* if the function succeeds.	

SetObjName

Purpose	Set the name of an object.	
Implementation	HRESULT *SetObjName* (int ObjIndex, BSTR ObjName)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	ObjName	A name for the object. Valid length: [1 to 256 characters]
Comments	Returns *S_OK* if the function succeeds.	
See Also	*GetObjName*	

SetObjNote

Purpose	Sets the note stored in the object.	
Implementation	HRESULT *SetObjNote* (int ObjIndex, BSTR Note)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	Note	The note to store in the object. Valid length: [1 to 256 characters]
Comments	Returns *S_OK* if function succeeds.	
See Also	*GetObjNote*	

SetObjNumPasses

Purpose	Sets the NumPasses value of an object.	
Implementation	HRESULT *SetObjNumPasses* (int ObjIndex, int PassCount)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	PassCount	The number of times to mark the object. Valid range: [1 or greater]
Comments	The use of NumPasses depends on the objects MarkMode setting. Use *GetObjMarkMode* to discover the current setting, and *SetObjMarkMode* to change it. Returns *S_OK* if function succeeds.	
See Also	*GetObjNumPasses*, *GetObjMarkMode*, *SetObjMarkMode*	

SetObjPos

Purpose	Set the position of a mark object.	
Implementation	HRESULT *SetObjPos* (int ObjIndex, int HPosition, int VPosition)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	HPosition	The x-coordinate of the lower-left corner of the bounding rectangle. Valid range: [-2,147,483,648 to 2,147,483,647]
	VPosition	The y-coordinate of the lower-left corner of the bounding rectangle Valid range: [-2,147,483,648 to 2,147,483,647]
Comments	The marking field is described using a Cartesian coordinate system, with (0,0) at the center of the field, (-32768, -32768) at the bottom left corner, and (32767, 32767) at the top right corner. Every MarkObject has a bounding rectangle, which describes the smallest rectangle that will fit around the object. Returns *S_OK* if the function succeeds.	

SetObjProfile

Purpose	Change the Profile settings for a mark object.	
Implementation	HRESULT *SetObjProfile* (int ObjIndex, int ProfileIndex, double Markspeed, double Jumpspeed, int Jumpdelay, int Markdelay, int Polygondelay, float Laserpower, int Laseroffdelay, int Laserondelay, int TAxis, double T1, int T2, int Unused, int Varijumpdelay, int Varijumplength, int Wobblesize, double Wobblefrequency, int Autosegmentation, int Varipolydelay)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	ProfileIndex	Index of Profile Valid range: [0 to 7]
	MarkSpeed	Defines the speed of the laser spot while marking. Valid range: [0 to 30000] bits/mm
	Jumpspeed	Defines the speed at which the mirrors jump to the next marking vector. Valid range: [50 to 30000] bits/mm
	Jumpdelay	Defines the delay after a jump and before the next marking vector starts. Valid range: [0 to 65500] µSec
	Markdelay	Defines the delay between a marking vector and a jump vector. Valid range: [30 to 20000] µSec
	Polygondelay	Defines the delay between contiguous marking vectors. Valid range: [0 to 20000] µSec
	Laserpower	Defines the programmed laser power for non CO_2-type lasers Valid range: [0 to 100] % (percent) Laserpower for CO_2- type lasers is defined as: Duty Cycle (%) = 0.1 × T2 [µs] × T1 [kHz]
	Laseroffdelay	Defines the delay after the last marking vector finishes and the laser is turned off. Valid range: [0-65535] µSec
	Laserondelay	Defines the delay after a marking vector starts and the laser is turned on. Valid range: [0-65535] µSec

	TAxis	Defines the Z position of the object. +Z is toward the scan head and –Z away from the scan head. Position is defined in bits and the same calibration factor is used as for x and y. Z field size is limited by the available Linear Translater movement. Values for Zmin and Zmax are defined in the scan head configuration file and can be read with *GetLensCalFactorEx* command. Units: bits Valid range: [Zmin to Zmax]
	T1	Defines the frequency of the laser modulation signal. Valid range: [0-250] kHz
	T2	Defines the pulse width of the laser modulation signal. Valid range: [1-65535] µSec
	Unused	Set to 0.
	Varijumpdelay	Defines the delay after a jump and before the next marking vector starts if variable jump delay is in effect. Set to 0 (zero) to disable variable jump delay. Valid range: [0-30000] µSec
	Varijumplength	Defines the length of a vector, at which any vector that is longer will use the Varijumpdelay parameter, and any vector that is shorter will use the Jumpdelay parameter. Valid range: [0-30000] bits
	Wobblesize	The diameter of the circle created when the spot is dithered. Set to 0 (zero) to disable Valid range: [0-5000] bits
	Wobblefrequency	The frequency of the laser spot as it dithers around the circle defined in Wobblesize. Valid range: [0-6000] Hz (cycles per second)
	Unused	Reserved. Set to 0.
	Varipolydelay	Reserved. Set to 1.
Comments	An object has eight profiles available, Profile0 to Profile7. When saving a job, however, only VectorGraphic objects save all eight profiles. All other objects only save Profile0 to Profile3. Returns *S_OK* if the function succeeds.	
See Also	*GetObjProfile*, *SetDefaultProfile*, *GetDefaultProfile*	

SetObjScanCardNum

Purpose	Set the scan card index number of an object.	
Implementation	HRESULT *SetObjScanCardNum* (int ObjIndex, int CardNum)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	CardNum	Index of scan head card Valid range: [0 to (number of cards-1)]
Comments	When multiple scan cards are installed in the computer, an objects scan head card index controls which card is used when marking the object. When an object is initially created, it has a CardNum of 0. If there is only one scan card in use, there is no need to call this function. Returns *S_OK* if the function succeeds.	
See Also	*GetScanCardNum*	

SetObjSize

Purpose	Set the size of a mark object from its center.	
Implementation	HRESULT *SetObjSize* (int ObjIndex, int HSize, int VSize)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	HSize	The width of the objects bounding rectangle in the x-axis. Valid range: [0-65535]
	VSize	The width of the objects bounding rectangle in the y-axis. Valid range: [0-65535]
Comments	The marking field is described using a Cartesian coordinate system, with (0,0) at the center of the field, (-32768, -32768) at the bottom left corner, and (32767, 32767) at the top right corner. Every MarkObject has a bounding rectangle, which describes the smallest rectangle that will fit around the object. Returns *S_OK* if the function succeeds.	

SetObjToRect

Purpose	Set the position and size of a mark object.	
Implementation	HRESULT SetObjRect (int ObjIndex, float* Left, float* Top, float* Right, float* Bottom)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	Left	The x-coordinate of the upper-left corner of the bounding rectangle. Valid range: unlimited value. Keep within [0 to 65535]
	Top	The y-coordinate of the upper-left corner of the bounding rectangle Valid range: unlimited value. Keep within [0 to 65535]
	Right	The x-coordinate of the lower-right corner of the bounding rectangle. Valid range: unlimited value. Keep within [0 to 65535]
	Bottom	The y-coordinate of the lower-right corner of the bounding rectangle Valid range: unlimited value. Keep within [0 to 65535]
Comments	The marking field is described using a Cartesian coordinate system, with (0, 0) at the center of the field, (-32768, -32768) at the bottom left corner, and (32767, 32767) at the top right corner. Every MarkObject has a bounding rectangle, which describes the smallest rectangle that will fit around the object. Returns *S_OK* if the function succeeds.	
See Also	*GetAllObjRect*, *GetObjRect*	

SetObjUsePensFlag

Purpose	Sets the use pens flag of an object.	
Implementation	HRESULT *SetObjUsePensFlag* (int ObjIndex, int Flag)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	Flag	The pens flag. [Valid values:0 or 1]
Comments	If the object contains pen information (usually in *.plt files), the Profile used to mark the object is selected by the current pen. Returns *S_OK* if function succeeds.	

SetPolygonAttributes

Purpose	Set the attributes of a polygon object.	
Implementation	HRESULT *SetPolygonAttributes* (int ObjIndex, int StartAngle, int EndAngle, int Sides)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
Returns	*StartAngle*	The starting angular direction of the polygon. 0 (zero) degrees corresponds to the 12:00 position. Valid range: [0 to 360] degrees
	EndAngle	The ending angular direction of the polygon. 360 degrees corresponds to the 12:00 position Valid range: [0 to 360] degrees
	NumSides	The number of straight-line segments in the polygon. Valid range: [3 to 10000]
Comments	Returns *S_OK* if the function succeeds.	
See Also	*GetPolygonAttributes*	

SetProcessEnabledWord

Purpose	Set the PROCESSENABLED ports on the Standard I/O card.	
Implementation	HRESULT *SetProcessEnabledWord* (int WordValue)	
Parameters	*WordValue*	Word value to be used to set the ports. Valid range: [0 to 63]
Comments	On the Standard I/O card, there are six bits that make up the PROCESSENABLED ports, hence a range of 0-63. Use WordValue to set the corresponding bits. For example, setting WordValue to 0 will set all the ports to false. Setting WordValue to 2 will set port 1 and port 2 to true. Setting WordValue to 63 will set all ports to true. The Standard I/O card uses reverse logic, so a true will set the port to ground. There must be a Standard I/O card installed for this function to succeed. Returns *S_OK* if the function succeeds.	

SetScanCardOutput

Purpose	Set a 16-bit port value on the SP-ICE card.	
Implementation	HRESULT *SetScanCardOutput* (int CardNum, int Offset, int Word, int Unused)	
Parameters	*CardNum*	Index of scan head card. Valid range: [0 to (number of cards-1)]
	Offset	The valid port address to set. See the SP-ICE card manual for more details. Valid values: [see SP-ICE card manual]
	Word	The lower 16 bits of Word are used to set the specified port. The upper 16 bits are ignored. Valid values: [0-65535]
Comments	This command is valid only for the SP-ICE scan head cards. Returns *S_OK* if the function succeeds	
See Also	*GetScanCardInput*	

SetTextAttributes

Purpose	Set the attributes of a text object.	
Implementation	HRESULT *SetTextAttributes* (int ObjIndex, BSTR FontName, int Orientation, int Kerning, int Leading, int Styles, int ParagraphStyle, int PulseCount)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	FontName	The name of the font. Valid value: [Any Windows TrueType font i.e. Arial, or installed Laser Font]
	Orientation	An integer value representing the physical orientation of singleline text objects. Orientation can contain one of the following values: 1 = Horizontal 2 = Vertical 3 = Radial Valid values: [1,2,3]
	Kerning	The added spacing between each character. Valid range: [-2000 to 2000] % (percent) of character width.
	Leading	The added spacing between each line in paragraph text. Valid range: [-2000 to 2000] % (percent) of character width.
	Styles	The font style (only TT-Fonts). Styles can contain a combination of the following values: 0 = Normal text 1 = Bold 2 = Italics Valid values: [0 ,1, 2]
	ParagraphStyle	The paragraph justification. For multiline text objects. It can be one of the following values: 0 = LeftJustify 1 = RightJustify 2 = CenterJustify Valid values: [0 ,1, 2]
	PulseCount	The number of laser pulses fired at each dot using the current laser frequency and pulse width settings.(For Dot-Marking) Valid range: [1000 to 10000] (0 deactivates Dot-Marking)
Returns	Returns *S_OK* if the function succeeds.	
Comments	Only for TT-Fonts.	
See Also	*GetTextAttributes*	

SetVectorGraphicAttributes

Purpose	Set the attributes of a vector graphic object.	
Implementation	HRESULT *SetVectorGraphicAttributes* (int ObjIndex, int PulseCount)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	PulseCount	The number of laser pulses fired at each point using the current laser frequency and pulse width settings. Valid range: [1000 to 10000] (0 deactivates Dot-Marking)
Comments	Not all vector graphic file formats support a point entity. If the vector graphic file contains point entities, the PulseCount parameter sets all point entities within the vector graphic to the same value. Returns *S_OK* if the function succeeds.	
See Also	*SetVectorGraphicAttributes*	

SetUserOutWord

Purpose	Set the Word value of the USEROUT ports.	
Implementation	HRESULT *SetUserOutWord* (int WordValue)	
Parameters	*Word*	Value of the word to set the USEROUT ports. Valid range: [0 to 63]
Comments	On the Standard I/O card, there are six bits that make up the USEROUT ports, hence a range of 0-63. Use WordValue to set the corresponding bits. For example, setting WordValue to 0 will set all the ports to false. Setting WordValue to 3 will set port 1 and port 2 to true. Setting WordValue to 63 will set all ports to true. The Standard I/O card uses reverse logic, so a true will set the port to ground. There must be a Standard I/O card installed for this function to succeed. Returns *S_OK* if the function succeeds.	

ShowTrayIcon

Purpose	Show or hide the COMServer system notification icon.	
Implementation	HRESULT *ShowTrayIcon* (int Show, int Protect)	
Parameters	*Show*	The show icon flag. Set to 1 to show the icon, 0 to hide it. Valid values: [0 or 1]
	Protect	The protect flag. When set to 1, if the icon is visible, the context menu is not available. When set to 0, the context menu is available, and the COMServer can be terminated from the context menu. Valid values: [0 or 1]
Comments	Use this command when initially debugging your application to show the icon. When ready for release, the system notification icon should be protected or hidden so the user cannot terminate the COMServer manually. Returns *S_OK* if the function succeeds.	

SkewObj

Purpose	Add a skew (shear) to an object.	
Implementation	HRESULT *SkewObj* (int ObjIndex, float XSkew, float YSkew)	
Parameters	*ObjIndex*	Index of object in the ObjectList Valid range: [0 to (number of objects-1)]
	Xskew	The amount of Xskew. Valid range: [- 180 to 180] degrees
	Yskew	The amount of Yskew. Valid range: [- 180 to 180] degrees
Comments	If XSkew and/or YSkew are non-zero, the object will shear in that axis. Returns *S_OK* if the function succeeds.	

TerminateMark

Purpose	Immediately stop the marking process.
Implementation	HRESULT *TerminateMark* (void)
Comments	Returns *S_OK* if the function succeeds.

TurnLaserOff

Purpose	Immediately turn the laser off.	
Implementation	HRESULT *TurnLaserOff* (int CardNum)	
Parameters	*CardNum*	Index of scan head card Valid range: [0 to (number of cards-1)]
Comments	This command is usually preceded by the command *TurnLaserOn*. CardNum is the 0 based index of the scan head card to query. Returns *S_OK* if the function succeeds.	
See Also	*TurnLaserOn*	

TurnLaserOn

Purpose	Position the laser beam and turn the laser on indefinitely.	
Implementation	HRESULT *TurnLaserOn* (int CardNum, float LaserPower, float Frequency, int PulseWidth, int XPosition, int YPosition)	
Parameters	*CardNum*	Index of scan head card Valid range: [0 to (number of cards-1)]
	LaserPower	Defines the programmed laser power. Valid range: [1 to 100] % (percent)
	Frequency	Defines the frequency of the laser modulation signal. Valid range: [0.02 to 50.0] kHz
	PulseWidth	Defines the pulse width of the laser modulation signal. Valid range: [2 to 65535] µSec
	XPosition	The X coordinate position of the laser spot. Valid range: [-32768 to 32767] bits
	YPosition	The Y coordinate position of the laser spot. Valid range: [-32768 to 32767] bits
Comments	This command must be followed by a call to *TurnLaserOff*. Before the laser turns on, the spot is moved to XPosition, YPosition, using the current JumpSpeed and JumpDelay. These coordinate points are in bits. The marking field is described using a Cartesian coordinate system, with (0,0) at the center of the field, (-32768, -32768) at the bottom left corner, and (32767, 32767) at the top right corner. Returns *S_OK* if the function succeeds.	
See Also	*TurnLaserOff*	

8 EXAMPLE CODE

An example program is provided to illustrate how to initiate a session with the weldMARK™ Automate object, load a pre-defined job, and then mark all objects in the job to the default scan head card.

8.1 C++ Example

```
// Initialize Windows COM libraries

::CoInitialize(NULL);
// Create an interface pointer
IAutomate* pMarker=NULL;
::CoCreateInstance(CLSID_Automate,
     NULL,
     CLSCTX_LOCAL_SERVER,
     IID_IAutomate,
     reinterpret_cast<void**>(&pMarker));

// Return value variable
HRESULT hr;
//Make sure there is a scan head card installed in computer
int count;
pMarker ->GetScanCardCount (&count);
if (count == 0)
     return Error;
int newjobindex;
try
     {
     // Load a presaved job from disk
     BSTR filename = ”c:\\test.wlj”;
     hr = pMarker ->LoadJobFromFile (filename, &newjobindex);
     if (FAILED(hr))
          {
          Application->MessageBox("Error","",MB_OK);
          Application->Terminate();
          }
     // Find out how many marking objects are in the job
     int objcount;
     hr = pMarker ->GetObjCount (&objcount);
     if (FAILED(hr))
          {
          Application->MessageBox("Error","",MB_OK);
          Application->Terminate();
          }
     // Make sure application is not currently marking
     int busy=1;
     while (busy==1)
          {
          hr = pMarker ->GetBusyStatus (0,&busy);
          if (FAILED(hr))
               {
               Application->MessageBox("Error","",MB_OK);
               Application->Terminate();
               }
          }
     busy=1;
     // Mark all objects in job with 90 degree rotation
     for (int i=0;i<objcount;i++)
          {
```

```
          // Make sure application is not currently marking
          int busy=1;
          while (busy == 1)
               {
               hr = pMarker ->GetBusyStatus (cardnum ,&busy);
               if (FAILED(hr))
                   {
                   Application->MessageBox("Error","",MB_OK);
                   Application->Terminate();
                   }
               }
          busy=1;
          // Mark object with Profile0
          hr = pMarker ->MarkObj (cardnum,i,90.0);
          if (FAILED(hr))
               {
               Application->MessageBox("Error","",MB_OK);
               Application->Terminate();
               }
          }
     }
catch (Exception& E)
     {
     return Error;
```

INDEX

O

R

S

T