Proprietary data

This manual, and the software is describes, are both copyrighted, with all rights reserved. Under the copyright laws, neither this manual nor the software may be copied, in whole or part, without written consent of MTS Systems Corporation, except in the normal use of the software or to make a backup copy of the software. The same proprietary and copyright notices must be affixed to any permitted copies as are made for others, whether or not sold, but all of the material purchased (with all backup copies) may be sold, given, or loaned to another person. Under the law, copying includes translating into another language or format. This software may be used on any computer, but extra copies cannot be made for that purpose.

Copyright information

© 1999-2003 MTS Systems Corporation. All rights reserved.

Trademark information

MTS, the MTS logo, and TestWare are registered trademarks of the MTS Systems Corporation.

FlexTest, Station Builder, Station Manager, TestStar, and TestWare are trademarks of the MTS Systems Corporation.

Microsoft, Windows NT, and Visual Basic, and Visual C++ are registered trademarks of the Microsoft Corporation.

Publication Information

<table>
<thead>
<tr>
<th>Software Version No.</th>
<th>Document No.</th>
<th>Release Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4</td>
<td>100-003-353</td>
<td>February 1999</td>
</tr>
<tr>
<td>3.0</td>
<td>100-037-822</td>
<td>June 2000</td>
</tr>
<tr>
<td>3.1</td>
<td>—</td>
<td>September 2000</td>
</tr>
<tr>
<td>3.2</td>
<td>—</td>
<td>July 2001</td>
</tr>
<tr>
<td>3.3</td>
<td>—</td>
<td>November 2002</td>
</tr>
<tr>
<td>3.4</td>
<td>—</td>
<td>September 2003</td>
</tr>
</tbody>
</table>
# Table of Contents

**Chapter 1 Introduction**

Overview 4
Prerequisites 4
For more information… 4

**Chapter 2 Usage notes**

Configuring a Visual Basic project to use the 793.00 components 5
Classes and interfaces 6
Object lifetimes 7
Event handling 8
Collections 8
Component diagrams 9
Foundation components 9
Dimensions and units components 9
System and station components 10
Signal components 11
Control channel and mode components 11
Function generation components 12
Data acquisition components 14
Time history data acquisition components 16
Detector components 16
Station hydraulic and interlock components 17
Miscellaneous components 18

**Chapter 3 Sample programs**

LogMonitor 20
SetLevel 20
SimpleTest 20

**Chapter 4 Advanced topics**

Component Object Model (COM) 21
Object references 21
Vtable interfaces 22
Message filters 23

**Appendix 1 Using other development tools**

Vtable outgoing interfaces 24
Enumerators 24
Passing arrays to functions 24
Getting class declarations into C++ 24

**Appendix 2 Redistributing applications**

Files required 26
Chapter 1
Introduction

Overview

The 793.00 Visual Basic Programming Libraries support Visual Basic developers who want to create applications for controllers running MTS 793.00 System Software. This document provides an overview of the programming libraries, including the interface design and some usage tips.

The design of the Visual Basic programming libraries is based on the C++ real-time programming model for the 793.00 platform. The two programming interfaces are intentionally very similar.

Prerequisites

Using the 793.00 Visual Basic libraries requires familiarity with Visual Basic component-based development. This includes knowledge of classes, interfaces, and events. For some of the more advanced topics, you will also need some background knowledge about Microsoft’s Component Object Model (COM), which is the underlying “protocol” used by Visual Basic for creating and using components.

For more details about the functions in the Visual Basic programming interface, please refer to the 793.00 C++ Programming Libraries User Guide. That document provides detailed information about the various classes, objects, and functions in the system for developers using the C++ API. Because of the similarity with the Visual Basic API, that document also serves as the reference for VB developers. By referring to the C++ User Guide in conjunction with this document, you can create applications that use the full set of features of the MTS 793.00 System Software.

For more information about COM, there are several good reference books available. For detailed information about how COM works, read Kraig Brockschmidt’s Inside OLE, Second Edition (MS Press). Another good overview of COM is Essential COM by Don Box (Addison Wesley). Both of these books approach COM primarily from a C/C++ programmer’s perspective. For a less technical description of COM, see Understanding ActiveX and OLE by David Chappell (MS Press).

One source of more information about Visual Basic and COM is Don Box’s series of columns in Microsoft Systems Journal. Although he doesn’t write exclusively about Visual Basic, his topics often deal with using COM from VB. There are also several journals available that focus on Visual Basic and, to various degrees, component-based development.
Chapter 2
Usage notes

Configuring a Visual Basic project to use the 793.00 components

Before you can use any of the MTS 793.00 Visual Basic libraries in a Visual Basic project, you must add references to the appropriate components. Open the Project/References… dialog box and click to add references to MTS 793.00 Box Detection Support, MTS 793.00 General Utilities, MTS 793.00 Real-time Programming Interface, and MTS 793.00 Units Support. The following figure shows the Visual Basic References window open with the appropriate entries checked:

Once you have added references to the MTS 793.00 libraries, you can write programs that use the components in the libraries. To browse the available methods and properties, use the Visual Basic Object Browser. The browser also includes a short description of each class and its members. For example, if you want more information on the RtSystem object, you would see:
The 793.00 Visual Basic Programming Libraries use a “multi-interface” design, where components often support more than one interface for clients to use. An application “asks” a component for one or more interfaces. In Visual Basic, the functionality available from a component depends on the type of the variable used to reference the component. This will be described in more detail in the following paragraphs.

Unfortunately, Visual Basic does not expose information about which interfaces are supported by each component through its Object Browser or its type system. In particular, Visual Basic often blurs the distinction between an object’s class and its default interface by interchanging their names in its type system. For example, given a component of type `RtSystem` with a default interface of `IRtSystem`, Visual Basic uses `RtSystem` as both the class name and as the name of the interface through which functions are invoked. Although Visual Basic allows the use of `IRtSystem`, it “hides” the interface in the browser and the Quick Info windows, preferring the use of `RtSystem` instead.

In C/C++ COM coding, the `QueryInterface()` function is used to request a specific interface from a component. In Visual Basic, the `Set` statement is used instead of `QueryInterface()`. If you want to see if a component supports a specific interface, use the `Set` statement to assign the object to a reference of another type. For example,

```vbnet
dim x as new RtSystem
dim y as INamed
x.Connect "", "AppName", "AppName"
set y = x
msgbox y.DisplayName
```

**Classes and interfaces**

The 793.00 Visual Basic Programming Libraries use a “multi-interface” design, where components often support more than one interface for clients to use. An application “asks” a component for one or more interfaces. In Visual Basic, the functionality available from a component depends on the type of the variable used to reference the component. This will be described in more detail in the following paragraphs.

Unfortunately, Visual Basic does not expose information about which interfaces are supported by each component through its Object Browser or its type system. In particular, Visual Basic often blurs the distinction between an object’s class and its default interface by interchanging their names in its type system. For example, given a component of type `RtSystem` with a default interface of `IRtSystem`, Visual Basic uses `RtSystem` as both the class name and as the name of the interface through which functions are invoked. Although Visual Basic allows the use of `IRtSystem`, it “hides” the interface in the browser and the Quick Info windows, preferring the use of `RtSystem` instead.

In C/C++ COM coding, the `QueryInterface()` function is used to request a specific interface from a component. In Visual Basic, the `Set` statement is used instead of `QueryInterface()`. If you want to see if a component supports a specific interface, use the `Set` statement to assign the object to a reference of another type. For example,
The Set statement will automatically call QueryInterface() to see if the RtSystem component supports the INamed interface. If the component does not support the requested interface, Visual Basic will generate a type mismatch error which will stop the application unless an error handler is installed. If the component does support the requested interface, Visual Basic will then have two references to the object, one through the variable x and another through the variable y. This technique is used to access the various functions supported by the components in the 793.00 Visual Basic libraries.

Interfaces group together related methods and properties. They are used to help organize the functionality in a logical way and also to provide polymorphism when appropriate. For example, objects that are named support the INamed interface. Any function requiring a named object can use this interface, regardless of the type of the component. Another example is the IRtTask interface; this interface is supported by all components in the system that have the behavior of a task, that is, can be started, stopped, etc. Any application code that operates on tasks can work with any object that supports IRtTask.

Object lifetimes

In certain cases, it is crucial for an application developer to understand when an object is created and when it will be destroyed. A good example of this is a read-write segment generator. When an application gets a reference to an RtReadWriteSg, it has effectively locked the segment generator for that channel so no other applications can use it. When the application is finished with the segment generator, it should release the reference to the RtReadWriteSg so it will again be available for other applications. All references to an object and its children must be released before the object will free its resources. If you allocate a read-write segment generator and create a ramp profile, just releasing the read-write segment generator will not unlock it. All profiles created through this segment generator must also be released.

In Visual Basic, references to objects are released in one of two ways. First, whenever a variable goes out of scope, it releases its reference. Second, an application can explicitly release a reference by assigning Nothing to an object. For example, consider the following code excerpt:

```vbs
Private rwsg As RtReadWriteSg
Private ramp As RtSegment

Public Sub StartFG(chan As RtChannel)
    Dim task As IRtTask
    Set rwsg = chan.AllocSegGen
    Set ramp = rwsg.CreateProfile(oProfSegment, ...)  
    ... configure the ramp and queue it ...
    Set task = rwsg
    task.Start
End Sub

Public Sub StopFG()
    Dim task As IRtTask
    Set task = rwsg
    task.Stop
    Set task = Nothing
    Set ramp = Nothing
    Set rwsg = Nothing
End Sub
```
In the `StopFG` subroutine, if only the `rwsg` variable were set to `Nothing`, the read-write segment generator would not be unlocked since the `ramp` variable would still be referencing a child profile.

Similar rules apply if an application wants to free other resources. If an application is collecting data, it must release all references to acquisition-related objects before the resources are freed on the 793.00 controller. If an application wishes to disconnect from a station, it cannot hold any references to station objects such as channels or signals.

The topic of object references and lifetimes is discussed in more general terms in Chapter 4 in the section *Object references*.

**Event handling**

The 793.00 Visual Basic libraries present events to client applications using COM connection points. Connection points are a standard, general-purpose COM protocol for delivering events to clients. In Visual Basic, connection points are accessed using the `WithEvents` keyword.

There are two steps to handling an event in Visual Basic. First, an object must be declared using `WithEvents`:

```vbnet
Dim WithEvents stream As RtLogStream
```

Next, the appropriate event handler functions must be written:

```vbnet
Private Sub stream_OnNewMessages()
End Sub
```

When the server object “fires” the events, the event handler will be called.

**Collections**

The 793.00 Visual Basic libraries return collections as an `IObjectCollection` reference. Visual Basic clients see this as an object of type `ObjectCollection`. Whether your application iterates through the collection using `For…Each` or by explicitly calling the `Item` method, the application gets back an `IUnknown` reference. This type is even more limited than the `Object` type in VB. It must be `Set` to the desired interface before use. Here is a code sample that shows how to use an `ObjectCollection`:

```vbnet
' Get the list of stations and display them in the combo box
Dim names As ObjectCollection
Dim s As Variant
Dim n As INamed
Set names = sys.StationNames
For Each s In names
    Set n = s
    comboStations.AddItem n.DisplayName
Next s
```

If you prefer using the `Item` method, the code would become:

```vbnet
' Get the list of stations and display them in the combo box
Dim names As ObjectCollection
Dim n As INamed
Dim i As Integer
Set names = sys.StationNames
For i = 1 To names.Count
    Set n = names.Item(i)
    comboStations.AddItem n.DisplayName
Next i
```
**Component diagrams**

The following diagrams show the components provided by the 793.00 Visual Basic Programming Libraries and the interfaces supported by each component. This reference can be used along with the Visual Basic Object Browser to see what capabilities are provided by the various components and interfaces in the system. The diagrams show both the incoming and outgoing (event) interfaces. The event interfaces are accessed using the ` WithEvents ` keyword in Visual Basic. For easier event implementations in non-VB tools, both a dispatch event interface and a vtable event interface is supported by each component that has events. Other than this difference, the two event interfaces support an identical set of events.

**Foundation components**

**Dimensions and units components**
System and station components
Signal components

Control channel and mode components
Function generation components

- **RtReadWriteSg**
  - IUnknown
  - IRtReadWriteSg
  - IRtObject
  - IRtTask
  - DIRtProfileEvents
  - IRtProfileEvents

- **RtStep**
  - IUnknown
  - IRtStep
  - IRtProfile
  - IRtObject
  - DIRtProfileEvents
  - IRtProfileEvents

- **RtSegment**
  - IUnknown
  - IRtSegment
  - IRtProfile
  - IRtObject
  - DIRtProfileEvents
  - IRtProfileEvents

- **RtCyclic**
  - IUnknown
  - IRtCyclic
  - IRtProfile
  - IRtObject
  - DIRtProfileEvents
  - IRtProfileEvents

- **RtProfile**
  - IUnknown
  - IRtProfile
  - IRtObject
  - DIRtProfileEvents
  - IRtProfileEvents
Data acquisition components
Time history data acquisition components

Detector components
Station hydraulic and interlock components
RtInterlock

IUnknown

IRtInterlock
IRtInterlockCommands
INamed
IRtObject
DIRtInterlockEvents
IRtInterlockEvents

RtTimer

IUnknown

IRtTimer
IRtObject
DIRtTimerEvents
IRtTimerEvents

RtLog

IUnknown

IRtLog
IRtObject
IRtLogCommands

RtLogStream

IUnknown

IRtLogStream
IRtObject
DIRtLogStreamEvents
IRtLogStreamEvents

RtSpanGrp

IUnknown

IRtSpanGrp
IRtObject
INamed
DIRtSpanGrpEvents
IRtSpanGrpEvents

RtSequence

IUnknown

IRtSequence
IRtObject
INamed

RtSyncGroup

IUnknown

IRtSyncGroup
IRtTask
IRtObject

Miscellaneous components
Chapter 3
Sample programs

LogMonitor
This sample application demonstrates how to monitor and add to the station log. It also shows how to connect to different stations.

SetLevel
This application demonstrates a variety of ways to manipulate a command signal on a control channel and also how to monitor the feedback signal. Some other interesting points include manual mode switches, using the RtTimer object, how to tell if a setpoint adjustment is complete, and how to use an event to signal the completion of a ramp.

SimpleTest
This sample application is a simple cyclic test that generates a sine wave and collects data on one or more input signals, writing the data out to a file. Some highlights of this code include buffered data acquisition, controlling the run/stop/hold state on the pod, using unit conversions for data input and display, and responding to interlocks while a test is running.
Chapter 4
Advanced topics

Component Object Model (COM)

The Visual Basic libraries are implemented as a group of components exposing their capabilities through a set of COM interfaces. COM is a binary standard established by Microsoft for component- and language-interoperability. It has sometimes been known by other acronyms such as OLE and ActiveX. For more information about COM, review the references listed in the first chapter of this document.

COM components are “black boxes” that expose their functions to external clients using interfaces, which can be thought of as contracts to provide a set of services. In typical Visual Basic development, components have a single interface that is associated with the object. However, COM supports any number of interfaces on a single component, as does Visual Basic, albeit somewhat indirectly.

Object references

Since the 793.00 Visual Basic libraries are created as COM components, they must follow the rules of COM reference counting. This can create some interesting situations.

Consider the following code sequence:

1. Create an RtSystem object and connect to the system
2. Get an RtStation object by connecting to a station on the system.
3. Get the list of RtFloatSigs in the station and store one signal in a variable for later use.
4. Release all references to the RtSystem, RtStation, and list of RtFloatSigs. Hang on to the single signal.

Consider what should happen after the fourth step. In the native C++ API, if an application deleted the station and system objects, all connections would be lost and any pointers the application held to individual signals would now be invalid. But in COM, all references held by an application must remain valid until the application releases them. The remaining reference to the signal is still valid and will still work as expected. This means the station and system are “alive” until the last reference to a related object is released. In a sense, there is a network of objects and any reference to a single object in that network keeps the entire network alive. The following diagram shows the runtime object relationships between some of the real-time objects provided by the VB API:
As long as the application keeps a reference to any one of the objects in the above diagram, the entire network stays active. Applications can use the IRtObject interface to traverse the runtime object hierarchy. In IDL, this interface is described as:

```c
interface IRtObject : IUnknown
{
[propget] HRESULT Parent([out, retval] IRtObject** parent);
[propget] HRESULT Station([out, retval] IRtStation** station);
[propget] HRESULT System([out, retval] IRtSystem** system);
HRESULT GetPropertyValue([in] BSTR propNmKey,
                           [out, retval] INamed** propVal);
};
```

Thus, given any real-time object, an application can get the associated station and the parent of that object.

**Vtable interfaces**

The components provided in the 793.00 Visual Basic libraries do not use dispatch interfaces (often called Automation interfaces). Instead, they provide vtable interfaces (sometimes called custom interfaces). This means that objects from the MTS libraries cannot be assigned to the Visual Basic Object type because any objects assigned to the Object type must have a dispatch interface. Objects should also be created with the new keyword. The following code will not work with the 793.00 libraries:

```vbnet
Dim s As Object
Set s = CreateObject("MTS793Rt.RtSystem")
```

Instead, the following code is used:

```vbnet
Dim s As RtSystem
Set s = New RtSystem
```

The use of vtable interfaces also enforces stricter type checking at compile time. Runtime binding is not supported (this is a feature supported explicitly by dispatch interfaces). Because runtime binding is not supported, there are certain development environments such as VBScript that cannot access the 793.00 libraries.

The use of vtable interfaces affects how the collections returned by the VB API are used. The main impact is that the returned object reference must be Set to a reference of the
desired type before it can be used. See the earlier section on using 793.00 object collections for more information.

**Message filters**

If an application includes code inside VB timer events or paint handlers, you need to ensure that you do not make a call into the MTS 793.00 Visual Basic libraries while you are inside another call in the libraries. Visual Basic will report an error if you try to do this. The error is coming from the COM runtime and is error number 0x80010005, RPC_E_CANTCALLOUT_INEXTERNALCALL. This error seems to happen more often in compiled applications than in the VB environment.

What happens is that while VB is waiting for a call to the MTS libraries to return, VB receives a timer or paint event and allows the application to handle it. At this point, COM will not allow another call into the MTS libraries to prevent certain deadlock situations. There are several workarounds for this:

- Use **RtTimer** objects rather than Visual Basic timers.
- Do not call any MTS library routines from timer or paint events.
- Replace the Visual Basic COM message filter, which is the code that decides whether events should be allowed through to the application while waiting for a remote server to respond.

The message filter is a standard COM facility, but we recommend caution when overriding Visual Basic’s built-in message filter. Also, providing your own message filter usually requires C or C++ coding. A simple implementation of the replacement message filter will have `IMessageFilter::MessagePending()` return `PENDINGMSG_WAITNOPROCESS` for all messages. This will prevent VB timers from firing events while in a call to the MTS libraries, but it may also make your application appear less responsive if it spends a lot of time in the MTS libraries.

The same problem can occur in applications developed with tools other than VB.
Appendix 1
Using other development tools

Vtable outgoing interfaces

When a component supports events, the default outgoing event interface is a dispatch interface suitable for Visual Basic clients. However, developers using other tools such as C++ may find it more convenient to use a vtable interface instead. Each component with events supports an alternate vtable event interface to make coding in such environments more convenient.

Use a type library browser such as the Microsoft OLE-COM Object Viewer to see the details about the supported event interfaces. The vtable interface is always named with the component class as the root, prefixed by I and suffixed by Events. For example, the Range component class has a vtable event interface called IRangeEvents.

Events interfaces are configured using connection points. Consult one of the available COM references for information on connection points.

Enumerators

Collections are objects that follow a COM protocol known by Visual Basic so applications can enumerate the objects contained in the collection. Non-VB clients may find it more convenient to use traditional COM enumerators. Whenever there is a method returning an IObjectCollection*, we have tried to provide a corresponding method returning an IEnumUnknown*.

Passing arrays to functions

Most array parameters are passed from Visual Basic clients to the MTS libraries as SAFEARRAYs. SAFEARRAYs are used by Visual Basic internally for array storage. Non-VB clients may find it more natural to work with “raw” memory arrays. Whenever there is a method using a SAFEARRAY parameter, we have tried to provide a corresponding method that uses a pointer to the array base type and a count indicating the length of the memory array. These functions have a V suffix to distinguish them from the SAFEARRAY versions.

Getting class declarations into C++

If you are developing a client using Visual C++, there are several ways to access the information required to use the 793.00 COM interface. The CD-ROM contains a folder with C and C++ header files that you can include directly into your source files. These header files were generated by the Microsoft IDL compiler. The required header files are:

ocom_util.h
ocom_units.h
ocom_rt.h
ocom_boxes.h

You will also need to include some C source files in your project to provide definitions of the various UUIDs provided by the 793.00 COM interface. You may need to include one or more of these files in your project:
ocom_util_i.c
ocom_units_i.c
ocom_rt_i.c
ocom_boxes_i.c

All of these files are located on the *MTS 793.00 Programming Libraries* CD-ROM in the folder called `COM-CPP-HDRS`.

Alternatively, you can use the `#import` directive in Visual C++ to access the information directly from the type libraries included in the COM servers. Please refer to the Visual C++ documentation for more information on how to use `#import`. 
Appendix 2
Redistributing applications

Files required

If you wish to distribute an application using the 793.00 Visual Basic libraries to other users who do not have development licenses, you will need to include several runtime components with your application. The runtime files are installed into the ntbin folder, which is in the main installation folder for the 793.00 System Software. The files required are:

```plaintext
...\ntbin\ocom_util.dll
...\ntbin\ocom_utilps.dll
...\ntbin\ocom_units.dll
...\ntbin\ocom_unitsps.dll
...\ntbin\ocom_rt.exe
...\ntbin\ocom_rtps.dll
...\ntbin\ocom_boxes.exe
...\ntbin\ocom_boxesps.dll
```

Your installation program should not overwrite newer versions of these files if they are present on the user’s system.

These files must be registered on the user’s system as COM servers in the order listed. The DLLs can be registered using `regsvr32.exe`, a Microsoft-provided redistributable file:

```plaintext
Regsvr32.exe ocom_xxx.dll
```

The EXE files can be registered by running them with the appropriate command line option:

```plaintext
ocom_rt.exe /RegServer
ocom_boxes.exe /RegServer
```

Most installation development tools will provide ways to automatically register COM servers.