G2 ActiveXLink

User's Guide

Version 2020

G2 ActiveXLink User's Guide, Version 2020

June 2020

The information in this publication is subject to change without notice and does not represent a commitment by Gensym Corporation.

Although this software has been extensively tested, Gensym cannot guarantee error-free performance in all applications. Accordingly, use of the software is at the customer's sole risk.

Ignite Technologies, Inc.
401 Congress Ave., Suite 2650
Austin, TX 78701 USA
Telephone: +1-800-248-0027
Email: success@ignitetech.com

Part Number: DOC034-1200

Contents

Preface ix

About this Guide ix

Version Information ix

Audience x

Conventions x

Related Documentation xii

Customer Support Services xiv

Chapter 1 Introduction 1

Introduction 1

What is an ActiveX Control? 1

What Does G2 ActiveXLink Do? 2

How Does G2 ActiveXLink Manage G2 Items? 3

Chapter 2 Creating a Link with G2 ActiveXLink 5

Introduction 5

Setting Up G2 for Authorization 7
- Running Your G2 Applications with G2 ActiveXLink 7
- Running the Example Programs 7

Using G2 ActiveXLink with Microsoft Visual Basic 7
- Adding the G2 ActiveXLink Control to the Toolbox 8
- Setting the Properties of the Control 11
- Building Your Connection Form 13
- Making a Connection to G2 14
- How to Communicate with G2 14

Using G2 ActiveXLink with Visual Basic .NET 20
- Visual Basic .NET Terminology 20
- Using ActiveXLink with Visual Basic .NET 21
- Programs that use AxG2Gateways 26
- The Program 28
- The PostMessage Demo 28
- The Call Demo 28

The Traffic Light Demo **29**

Using G2 ActiveXLink with Microsoft Excel **29**
Making a Connection to G2 **30**
Setting the Properties of the Control **31**
Calling a Procedure in G2 and Excel **33**

Using G2 ActiveXLink with Microsoft Internet Explorer **35**
Adding the G2 ActiveXLink Control to an HTML File **35**
Connecting with G2 on Startup **35**
Sending a Message to G2 from Internet Explorer **37**

Using G2 ActiveXLink with C++ **37**

Chapter 3 **Data Types** **43**

Introduction **43**

Mapping Data Types **44**

Overriding the Mapping of Simple Data Types **45**

Mapping Date and Time **46**

Mapping Currency **47**

Mapping Multidimensional Arrays **47**
Example: One-Dimensional Array **49**
Example: Two-Dimensional Array **49**
Example: Multidimensional Array **49**

Chapter 4 **Using G2Gateway** **51**

Introduction **52**

Properties **52**

Methods **56**
Call( ) **57**
Start( ) **59**
CallDeferred( ) **61**
Connect( ) **63**
Disconnect( ) **64**
PostMessage( ) **65**

Events **66**
g2com-call **67**
g2com-start **68**
g2com-start-over-interface **69**
RpcCalled( ) **71**
RpcStarted( ) **74**
G2Connected **76**

G2Disconnected **77**
RpcReturned( ) **78**
G2Paused **79**
G2Resumed **80**
G2Reset **81**
G2Started **82**
G2RunStateKnown **83**
AttributeModified() **84**
CustomEvent() **85**
IconColorChanged() **86**
ItemAdded() **87**
ItemDeleted() **88**
ItemRemoved() **89**
ItemSelected() **90**
ValueChanged() **91**
Error **92**

Chapter 5 **Custom Classes** **95**

Introduction **95**

Using G2Symbol **96**

Using G2Structure **97**
Creating a Variable to Represent a G2Structure **97**
Example: Reading the Value of a Structure Property **99**
Example: Setting the Value of a Structure Property **99**
Example: Determining the Number of Name/Value Pairs **99**
Example: Obtaining Lists of Property Names or Values **100**
Example: Removing a Name/Value Pair from a G2Structure **100**
Example: Iterating over Name/Value Pairs **100**

Using G2Item **101**
Specifying the G2 Class **101**
Setting the G2Item Name **102**
Determining the Number of User-Defined Attributes **102**
Getting and Setting Attribute Values **102**
Getting Attribute Names, Values, and Types **104**
Creating G2Items **105**
Creating Symbolic Attributes **106**
Removing Attributes **106**
Iterating Over the Attributes of a G2Item **107**
Using G2Item Value and Type **108**

Using G2List and G2Array **108**
Determining the Number of Elements **109**
Getting and Setting Element Values **109**
Determining the Type **109**
Inserting, Appending, and Adding Elements to the List or Array **109**

Removing Elements from a List or Array **110**
Iterating Over Elements of a List or Array **110**
Sending Lists and Arrays to G2 **110**

Using G2Workspace **112**
Subscribing to Workspace Events **112**

Using G2Window **113**
Getting the G2 User Mode of a Window **113**
Subscribing to Window Events **114**

G2 Type Names **114**

Subscription Types **115**

Chapter 6 **Item References** **117**

Introduction **117**

Creating and Linking a G2Item **118**

Getting the Icon for a G2Item **121**

Deleting a G2Item **122**

Updating the Item in G2 **122**

Refreshing a G2Item **124**

Verifying Linked Items **124**

Unlinking a G2Item **124**

Getting G2Item Attribute Names, Values, and Types **125**

Using Linked Items as Parameters to RPCs **126**

Subscribing to Item Events **127**
Subscribing to Attribute Changes **128**
Subscribing to Item Deletions **129**
Subscribing to Icon Color Changes **129**
Subscribing to Variable and Parameter Value Changes **129**
Subscribing to Custom Events **130**
Unsubscribing from Attribute Changes **130**
Unsubscribing from Item Deletions **130**
Unsubscribing from Icon Color Changes **130**
Unsubscribing from Custom Events **130**
Unsubscribing from Variable and Parameter Value Changes **131**
Unsubscribing from All Event Notification **131**
Getting Information about Subscriptions **131**

Appendix A **Example Code** **135**

Introduction **135**

Using G2 ActiveXLink in Microsoft Visual Basic **135**
Using G2 ActiveXLink in Microsoft Excel **137**

Index 139

Preface

The preface describes this document and the conventions that it uses.

About this Guide **ix**

Version Information **ix**

Audience **x**

Conventions **x**

Related Documentation **xii**

Customer Support Services **xiv**

About this Guide

This guide introduces the G2 ActiveX Link control, its capabilities, and how you, the developer, can use it to create a link with G2 and pass data between G2 and a container application such as Microsoft Visual Basic.

Version Information

The G2 ActiveXLink control works on Windows 7, 8.1 or 10, Windows Server 2008/2008 R2 or 2012 R2.

G2 ActiveXLink operates in any Microsoft COM-compatible container or development environment that supports Microsoft COM, including:

- Microsoft Office, including Word, Excel, and PowerPoint
- Microsoft Internet Explorer

- Microsoft Visual Basic and Visual Basic .NET
- Microsoft Visual C++
- Active Server Page (ASP)

Audience

You should be familiar with ActiveX and how ActiveX controls are used in the Windows development environment. You should also be familiar with Microsoft Visual Basic or Microsoft Visual Basic for Applications in Microsoft Office.

This guide assumes that you are already an experienced user of G2.

Conventions

This guide uses the following typographic conventions and conventions for defining system procedures.

Typographic

Convention Examples	Description
g2-window, g2-window-1, ws-top-level, sys-mod	User-defined and system-defined G2 class names, instance names, workspace names, and module names
history-keeping-spec, temperature	User-defined and system-defined G2 attribute names
true, 1.234, ok, “Burlington, MA”	G2 attribute values and values specified or viewed through dialogs
Main Menu > Start KB Workspace > New Object create subworkspace Start Procedure	G2 menu choices and button labels
conclude that the x of y ...	Text of G2 procedures, methods, functions, formulas, and expressions

Convention Examples	Description
new-argument	User-specified values in syntax descriptions
text-string	Return values of G2 procedures and methods in syntax descriptions
File Name, OK, Apply, Cancel, General, Edit Scroll Area	GUIDE and native dialog fields, button labels, tabs, and titles
File > Save Properties	GMS and native menu choices
workspace	Glossary terms
`c:\Program Files\Gensym\`	Windows pathnames
`/usr/gensym/g2/kbs`	UNIX pathnames
`spreadsh.kb`	File names
`g2 -kb top.kb`	Operating system commands
`public void main()` `gsi_start`	Java, C and all other external code

Note Syntax conventions are fully described in the *G2 Reference Manual*.

Procedure Signatures

A procedure signature is a complete syntactic summary of a procedure or method. A procedure signature shows values supplied by the user in *italics*, and the value (if any) returned by the procedure *underlined*. Each value is followed by its type:

g2-clone-and-transfer-objects
(*list*: class item-list, *to-workspace*: class kb-workspace,
delta-x: integer, *delta-y*: integer)
–> *transferred-items*: g2-list

Related Documentation

G2 Core Technology

- *G2 Bundle Release Notes*
- *Getting Started with G2 Tutorials*
- *G2 Reference Manual*
- *G2 Language Reference Card*
- *G2 Developer's Guide*
- *G2 System Procedures Reference Manual*
- *G2 System Procedures Reference Card*
- *G2 Class Reference Manual*
- *Telewindows User's Guide*
- *G2 Gateway Bridge Developer's Guide*

G2 Utilities

- *G2 ProTools User's Guide*
- *G2 Foundation Resources User's Guide*
- *G2 Menu System User's Guide*
- *G2 XL Spreadsheet User's Guide*
- *G2 Dynamic Displays User's Guide*
- *G2 Developer's Interface User's Guide*
- *G2 OnLine Documentation Developer's Guide*
- *G2 OnLine Documentation User's Guide*
- *G2 GUIDE User's Guide*
- *G2 GUIDE/UIL Procedures Reference Manual*

G2 Developers' Utilities

- *Business Process Management System Users' Guide*
- *Business Rules Management System User's Guide*
- *G2 Reporting Engine User's Guide*
- *G2 Web User's Guide*
- *G2 Event and Data Processing User's Guide*

- *G2 Run-Time Library User's Guide*
- *G2 Event Manager User's Guide*
- *G2 Dialog Utility User's Guide*
- *G2 Data Source Manager User's Guide*
- *G2 Data Point Manager User's Guide*
- *G2 Engineering Unit Conversion User's Guide*
- *G2 Error Handling Foundation User's Guide*
- *G2 Relation Browser User's Guide*

Bridges and External Systems

- *G2 ActiveXLink User's Guide*
- *G2 CORBALink User's Guide*
- *G2 Database Bridge User's Guide*
- *G2-ODBC Bridge Release Notes*
- *G2-Oracle Bridge Release Notes*
- *G2-Sybase Bridge Release Notes*
- *G2 JMail Bridge User's Guide*
- *G2 Java Socket Manager User's Guide*
- *G2 JMSLink User's Guide*
- *G2 OPCLink User's Guide*
- *G2 PI Bridge User's Guide*
- *G2-SNMP Bridge User's Guide*
- *G2 CORBALink User's Guide*
- *G2 WebLink User's Guide*

G2 JavaLink

- *G2 JavaLink User's Guide*
- *G2 DownloadInterfaces User's Guide*
- *G2 Bean Builder User's Guide*

G2 Diagnostic Assistant

- *GDA User's Guide*
- *GDA Reference Manual*
- *GDA API Reference*

Customer Support Services

You can obtain help with this or any Gensym product from Gensym Customer Support. Help is available online, by telephone and by email.

To obtain customer support online:

➔ Access Ignite Support Portal at `https://support.ignitetech.com`.

You will be asked to log in to an existing account or create a new account if necessary. Ignite Support Portal allows you to:

- Register your question with Customer Support by creating an Issue.
- Query, link to, and review existing issues.
- Share issues with other users in your group.
- Query for Bugs, Suggestions, and Resolutions.

To obtain customer support by telephone or email:

➔ Use the following numbers and addresses:

United States Toll-Free	+1-855-453-8174
United States Toll	+1-512-861-2859
Email	support@ignitetech.com

1

Introduction

Introduces G2 ActiveXLink and the containers in which it operates.

Introduction 1

What is an ActiveX Control? 1

What Does G2 ActiveXLink Do? 2

How Does G2 ActiveXLink Manage G2 Items? 3

Introduction

G2 ActiveXLink enables you to establish communications between G2 and a COM-compliant application running under Windows 7, 8.1 or 10, Windows Server 2008/2008 R2 or 2012 R2. This chapter discusses ActiveX controls in general and the G2 ActiveXLink control in particular.

What is an ActiveX Control?

ActiveX controls provide object-oriented programming and reusable software components that conform to the Component Object Model (COM). They can interact with your application to perform a set of functionality. You can use an ActiveX control as if it were part of your application.

ActiveX is an extension of Object Linking and Embedding (OLE). OLE enables programs to share data. Both ActiveX and OLE are layered on top of COM, an industry standard object model that defines the rules by which objects are structured and the rules by which objects communicate and expose their functionality.

Because ActiveX controls use a standard interface specification, your application can access the features of an ActiveX control with a few lines of code. Your application becomes the container for the control. Any application or language that supports Microsoft COM can use G2 ActiveXLink.

ActiveX controls define a set of properties and communicate with the container application by using methods and events.

- Properties – An ActiveX control provides a set of values or characteristics, such as network address of the G2 server, that can be set and read, or only read.
- Methods – The container application can use the features of an ActiveX control by invoking its methods, such as a method that invokes a procedure in G2.
- Events – To the container application, an ActiveX control raises events, such as notification that a connection has been established with G2, to which the container application responds.

What Does G2 ActiveXLink Do?

G2 ActiveXLink enables container applications and languages that support Microsoft COM, such as Microsoft Office, Microsoft Visual Basic, Visual C++, Microsoft Internet Explorer, and Active Server Page (ASP) to communicate with G2. G2 ActiveXLink provides the `G2Gateway` control, which does the following:

- Enables users to invoke procedures in a G2 server, passing any number of arguments and returning any number of arguments with as little as a single line of code.
- Automatically maps data types.
- Supports both synchronous (blocking) and nonblocking calls.
- Can be used safely in multi-threaded applications because G2 ActiveXLink is thread-safe.
- Creates connections to multiple G2 servers at the same time.
- Automatically manages connections to the G2 server.
- Stores configuration information, such as the G2 server location as a visually configurable property.

Additionally, the G2 server can invoke logic in the COM-compliant container application with or without return arguments. Clients, the container applications, can post messages on the G2 Message Board.

The following Visual Basic code fragment shows how compact and powerful calls to G2Gateway can be:

```
Private Sub Form_Load()
   Call G2Gateway1.PostMessage("Hello from Visual Basic!")
   Call G2Gateway1.Call("My-Procedure",1,123,3.1415,True)
End Sub
```

The Form_Load() function automatically:

- Creates a connection to a G2 server.
- Posts a message to the G2 Message Board.
- Calls the G2 procedure my-procedure with four arguments.

The G2 server resides at the TCP/IP address specified in the G2Location property of the G2Gateway1 object inserted in the Visual Basic form. The G2Gateway1 object is an instance of the G2Gateway class in G2 ActiveXLink.

For details, see:

- Chapter 2, Creating a Link with G2 ActiveXLink on page 5.
- Chapter 3, Data Types on page 43.
- Chapter 4, Using G2Gateway on page 51.

How Does G2 ActiveXLink Manage G2 Items?

G2Gateway is a control that you normally place on a form at design time, although it is not visible at run time. G2 ActiveXLink also defines a number of classes, which are not controls so they are not visible and are, therefore, only available at run time. These classes include:

- G2Symbol
- G2Structure
- G2Item
- G2List and G2Array
- G2Workspace
- G2Window

You use these classes to represent G2 items in you COM application. By default, a G2Item is a static copy of the item in G2. You can also create a G2Item so that is linked to the item in G2, which means the item updates automatically in both directions when changes occurs.

G2Item defines a number of methods for subscribing to various events on the item. These events occur on the G2Gateway to which the G2Item is linked.

`G2Gateway` provides notification for these events: attribute changes, item deletions, icon color changes, variable or parameter value changes, and custom events.

For details, see:

- Chapter 5, Custom Classes on page 95.
- Chapter 6, Item References on page 117.

2

Creating a Link with G2 ActiveXLink

This chapter describes the steps for creating and sending information across the link.

Introduction **5**

Setting Up G2 for Authorization **7**

Using G2 ActiveXLink with Microsoft Visual Basic **7**

Using G2 ActiveXLink with Visual Basic .NET **20**

Using G2 ActiveXLink with Microsoft Excel **29**

Using G2 ActiveXLink with Microsoft Internet Explorer **35**

Using G2 ActiveXLink with C++ **37**

Introduction

This chapter describes how to use the G2 ActiveXLink control with the popular ActiveX control containers Microsoft Visual Basic, Microsoft Visual Basic .NET, Microsoft Excel, and Microsoft Internet Explorer. You can also use G2 ActiveXLink with any application that supports ActiveX controls, such as Active Server Page (ASP).

The chapter explains the steps for establishing a connection with G2 using G2 ActiveXLink. Over this connection, data is transmitted to G2 and data is returned from G2 to the container application. The container application can start a procedure in G2, and G2 can start a procedure in the container application.

Note G2 ActiveXLink is a COM component. The descriptions in this guide are generally for non-.NET products. Users of .NET need to wrap the ActiveXLink control and modify the instructions in a manner similar to that described in Using G2 ActiveXLink with Visual Basic .NET on page 20.

The example programs used in this chapter show you how the properties, methods, and events in the G2 ActiveXLink control are used to establish a connection with G2 and to send and receive data. Use these example programs as guides for creating your own programs.

In most applications, one or more G2Gateways will be added to the program when it is being designed. Rarely, an application's code will create a G2Gateway at runtime. In this case, the G2Gateway is said to have been created dynamically.

When you create a G2Gateway dynamically, you must call the `OnEndPage` method before deleting the G2Gateway to cause it to shut down its event thread. You should also call the `OnEndPage` method in the error event handler when an attempt to connect with a new, dynamically created G2Gateway fails.

For most applications, G2 ActiveXLink provides excellent throughput. However in extreme cases, you might wish to increase the number of calls from G2 it can process. This is possible by enabling the high-throughput option. Typically G2 ActiveXLink will be able to process more than 50 times as many calls from G2 per second as would be possible with the high-throughput option disabled.

To do this, you create a `G2ComConfigurator` object and set its `HighThroughput` property to `True`. You can then delete the `G2ComConfigurator` object. The following Visual Basic 6 code accomplishes this:

```
Dim axlCfg as New G2ComConfigurator
axlCfg.HighThroughput = True
Set axlCfg = Nothing
```

Enabling the `HighThroughput` option causes G2 ActiveXLink to take 100% of the available processor time. It will appear that the processor is overloaded. However, as long as your application does not push the processor to its limit, the responsiveness of the computer will still be good; generally, G2 ActiveXLink will immediately release the processor if any other process needs it.

You should use the `g2com.kb` and `gsi.dll` that come with the `g2com.dll` (ActiveXLink) that you are using. In general, you should use compatible versions of G2 and G2 ActiveXLink (`g2com.kb`). However, if no substantive changes have been made to `g2com.kb` and you do not wish to upgrade versions of G2, you can sometimes use newer versions of G2 ActiveXLink (`g2com.kb`) with older versions of G2. Refer to the G2 Bundle Release Notes for version compatibility.

Setting Up G2 for Authorization

Before you can communicate with G2, merge the `g2com.kb` file into your G2 KB application. The `g2com.kb` contains declarations that are required to enable G2 to use G2 ActiveXLink. The default location is `\g2\kbs\utils\g2com.kb` in your G2 Bundle installation directory.

The `g2com.kb` file also contains authorization information that enables G2 ActiveXLink to run.

Running Your G2 Applications with G2 ActiveXLink

To run your G2 application KB with G2 ActiveXLink, load your KB into G2 and merge `g2com.kb` into it.

To set up G2 to communicate with G2 ActiveXLink:

1. Load your G2 application into G2.
2. Merge `g2com.kb` into your application.
3. Start G2.

Running the Example Programs

To run the example programs, load the demonstration KB `axldemo.kb` into G2. The `g2com.kb` is already merged into the demonstration KB. G2 starts automatically.

The default location is `\g2\kbs\demos\axldemo.kb`.

To set up G2 to communicate with the example programs:

➔ Load `axldemo.kb` into G2.

Using G2 ActiveXLink with Microsoft Visual Basic

The following examples of using G2 ActiveXLink with Visual Basic walk you through placing the G2 ActiveXLink control on a form, calling and starting procedures in G2, and raising an RPC event from G2.

The examples in this section are drawn from the demonstration Visual Basic project shipped with G2 ActiveXLink. The default location is `\activexlink\demos\vbdemo\VBDemo.vbp`.

You can also run this example from the Start menu by choosing:

Start > Programs > Gensym G2 2011 > Examples > ActiveXLink > VB Demo Project and VB Demo

The following instructions are for non-.NET versions of Visual Basic. Users of VB .NET should refer to Using G2 ActiveXLink with Visual Basic .NET on page 20.

Adding the G2 ActiveXLink Control to the Toolbox

To give your Visual Basic form the ability to connect to G2, you must place the G2 ActiveXLink control on a form. Before you can place the control, you must add the control to the Visual Basic toolbox.

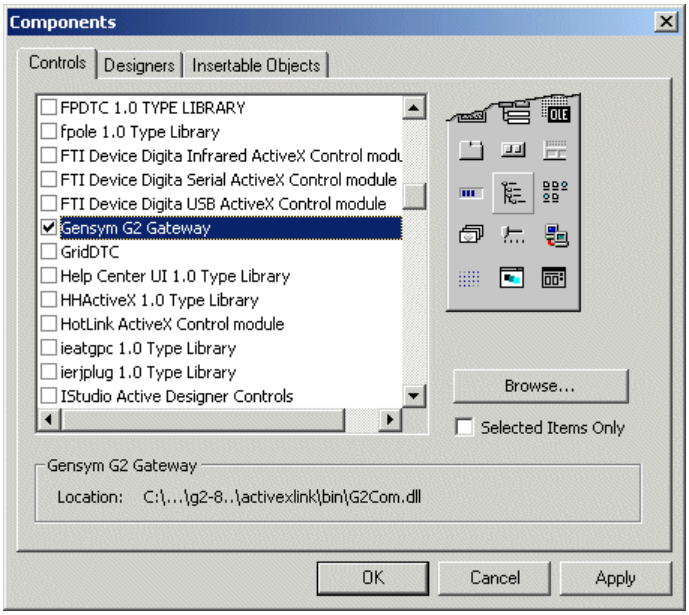
To add the G2 ActiveXLink control to the toolbox:

1 Create or open a project in Microsoft Visual Basic.

Tip You can open the demonstration Visual Basic project and observe how the G2 ActiveXLink is used as you follow these examples.

---

2 Choose the Components option in the Project menu in Visual Basic to display the Components dialog, as shown in the following figure:



3 Locate the "Gensym G2 Gateway" in the list on the Controls page and click the box to select the control, as shown.

4 Click OK on the Components dialog.

In the toolbox, you see a new icon for the G2 ActiveXLink control. The name of the class is "G2Gateway," as shown in the following figure:

Using the Control in Your Form

You can place the control directly on your form or you can create an instance of the control programmatically in Visual Basic code.

When placed on a form, an instance of the G2Gateway class appears in the Properties window as "G2Gateway1", as shown in the following figure:

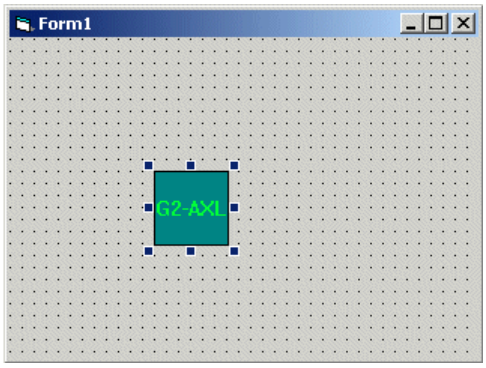
If you have more than one G2 ActiveXLink G2Gateway object in your project, the number for each of the subsequent controls automatically increments; for example, G2Gateway2, G2Gateway3, G2Gateway*n*. You can, however, rename the instances to whatever you want.

Each G2Gateway object can connect to the same G2 server or to a different G2 server, if it is authorized to run G2 ActiveXLink. For example, you could connect to more than one G2 server by using a different instance of the G2Gateway class for different G2 servers.

To place the G2 ActiveXLink directly on your form:

1. Click the G2 Gateway icon in the Visual Basic toolbox.
2. Place the cursor over the form in the form designer.
3. Press and hold the mouse button and drag the plus cursor down and to the right over the area of the form where you want to place the control.

4 Release the mouse button and the control appears with the words “G2-AXL:”



When you run your form, the control is invisible and it may be located under other controls on the form.

Note To place the control directly on the form, double click on the icon. Visual Basic places it in the middle of the form. For more information on building your form, see Building Your Connection Form on page 13.

You can complete the form as you would any other Visual Basic form.

Creating the Control Programmatically

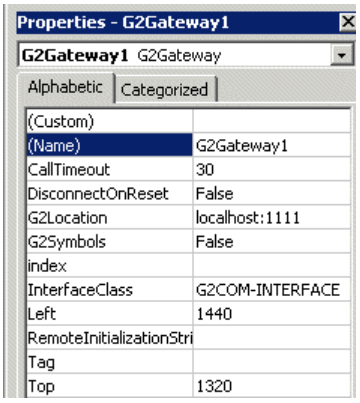
You can use the G2 ActiveXLink control programmatically by referencing it in your code. The following code fragment shows how you can create and define a variable for the G2 ActiveXLink control, create an instance of the control, and use it to start a G2 procedure in a running G2.

```
Dim myCTL As G2Gateway
Set myCTL = New G2Gateway
myCTL.Start "G2PROC-ONE", StartItem.Text
```

Note Make sure that `Gensym G2 Gateway` is referenced by Visual Basic. To bring up the Available References dialog, click References in the Project menu.

Setting the Properties of the Control

Like other controls in Visual Basic, the G2 ActiveXLink control has properties you can set. You can set them by clicking on the control in the form to display the Properties window, as shown in the following figure:



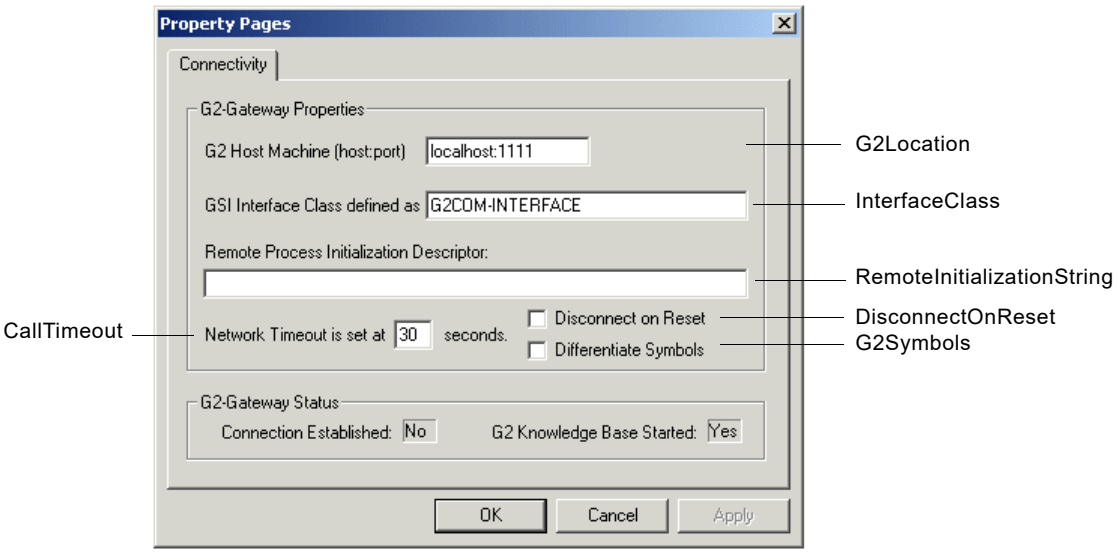
You can set properties by typing values in the text boxes in the Properties window.

Setting the Basic Properties

The basic properties for the G2 ActiveXLink control set values for the location of the G2 process, the interface class for the connection, the name you give the connection, and the timeout for a response from G2.

For the four basic properties, a button with three dots appears when you click in its text box. Click the button shown in the following figure to display the Property Pages dialog for the control:

The following figure shows the Property Pages dialog with the properties you can set for the G2 ActiveXLink control:



The basic properties are:

- `G2Location` – The host machine name and port number for the machine running G2. The name and port number are separated by a colon. The default is `localhost:1111` and the Visual Basic data type is `String`. In G2, the data type is text.
- `InterfaceClass` – The G2 class that defines the connection in G2. The default is g2com-interface (vbdemo-interface in this example), and the Visual Basic data type is `String`. In G2, the data type is symbol.
- `RemoteInitializationString` – A descriptive identifier for the connection that G2 uses. There is no default. The Visual Basic data type is `String`. In G2, the data type is text.
- `DisconnectOnReset` — Allows you to choose the reset behavior that your application requires. When set to `False`, the default, the connection between G2 ActiveXLink and G2 is maintained when G2 is reset. When set to `True`, the connection between G2 ActiveXLink and G2 is broken when G2 is reset
- `G2Symbols` — Allows you to choose the behavior when sending symbols from G2. When set to `True`, the default, simple symbols are stored as instances of `G2Symbol`. When set to `False`, symbols are stored as `String` types.
- `CallTimeout` – The maximum amount of time for the application containing the G2 ActiveXLink control to wait for G2 to respond. The default is 30 seconds and the Visual Basic data type is `Long`. In G2, the data type is integer.

For more information on these and other G2 ActiveXLink properties, see Properties on page 52.

Setting Properties Programmatically

Instead of using a form, you can set the value of a property programmatically in your Visual Basic code.

To set a property programmatically

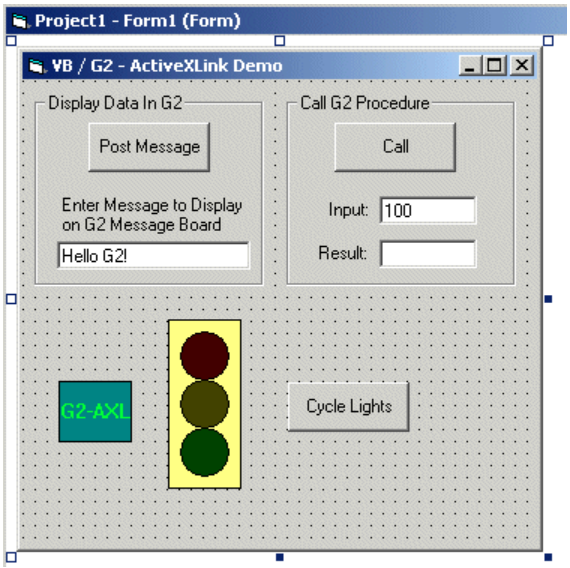
➔ Type the name of a property and the value you want to give it, as shown:

```
G2Gateway1.G2Location = "pc1:1122"
```

For example, you could programmatically set the `G2Location` property by prompting the user for a value in a text box. With the ability to set the `G2Location` property in a text box, you could link to a different G2.

Building Your Connection Form

You can build a form in Visual Basic quickly. Just as you did with the G2 ActiveXLink control, you can create instances of buttons, text boxes, and other controls on the form. The following figure shows the working version of the form from the demonstration project shipped with G2 ActiveXLink. The G2 ActiveXLink control is selected.



The next sections on connecting and transmitting data use this example to illustrate how you can work with the G2 ActiveXLink control. The form calls procedures in G2 and displays values returned by G2.

For more information on building a form in Visual Basic, refer to the documentation on Visual Basic or one of the numerous books on learning and using Visual Basic.

Making a Connection to G2

Making a connection to G2 (or "G2 server") is automatic when you open a Visual Basic program and transmit data to G2 by using the `Start()`, `Call()`, or `CallDeferred()` methods. You can connect to any G2 server running anywhere in your TCP/IP network.

You can also explicitly call the `Connect()` method to establish a connection.

If it is not already connected, G2 by itself cannot connect with a container application by using G2 ActiveXLink.

When G2 responds to the G2 ActiveXLink connection request, the control raises the `G2Connected` event to Visual Basic, which you can program to respond appropriately.

Connecting with G2 on Startup

You can connect to G2 as soon as you run the Visual Basic form. This way of connecting saves time when you click a button to make your first remote procedure call.

You can program an application to establish a connection on startup by placing the `Connect()` method in the application's startup procedure. For example, the following procedure in Visual Basic attempts to make a connection to G2 when the form loads:

```
Private Sub Form_Load()
   Call G2Gateway1.Connect(false)
End Sub
```

When you load the Visual Basic form, the `Form_Load()` function executes:

- The `Connect(false)` call creates a connection to G2, but the system does not wait for the connection to complete before displaying the Visual Basic form.
- The `Connect(true)` call also creates a connection to G2, but the system waits for the connection to complete before displaying the form.

For a description of the `Connect()` method, see Connect() on page 63.

Note To explicitly disconnect from G2 when you close a Visual Basic form, you can place the `Disconnect()` call in the `Form_Unload()` function.

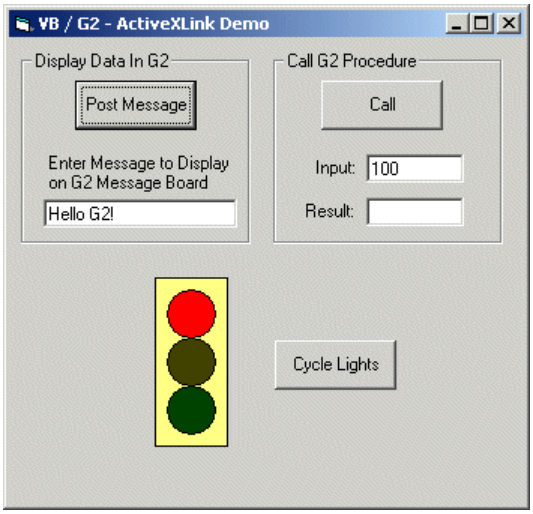
How to Communicate with G2

You can transmit data to G2, invoke a procedure in G2, and return data to Visual Basic. The following three sections reference the example form and describe each of the form's three buttons that you use to perform operations. Each section includes the relevant Visual Basic code and G2 procedures.

Note To display the Visual Basic example form, run `VBDemo.exe`. Its default location is `\activexlink\demos\vbdemo\VBDemo.exe`.

Posting a Message on the G2 Message Board

The button labeled "PostMessage" on the example form enables you to send any data to G2 for display on the Message Board.



For example, when you click the PostMessage button on the Visual Basic example form, the message "Hello G2!" appears on the G2 Message Board, as shown in the following figure:

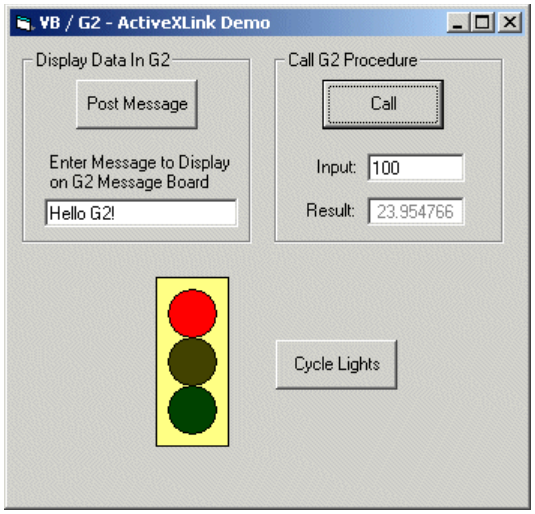
Clicking the PostMessage button invokes the `PostMessage` G2 ActiveXLink method, which passes the string in the text box `StartItem` to the G2 Message Board. The following Visual Basic code fragment specifies that the `PostMessage` method is invoked when you click the PostMessage button:

```
Private Sub StartRPC_Click()
   G2Gateway1.PostMessage StartItem.Text
End Sub
```

For more information on the `PostMessage` method, see PostMessage() on page 65.

Calling a Procedure in G2

The button labeled "Call" on the example form enables you to invoke a random number generator procedure in G2 and display the result on the Visual Basic form. The result appears in the "Result" text box, as shown in the following figure:



When you click the Call button on the Visual Basic example form, the `Call` method invokes a procedure named G2RandomGenerator in G2 and passes the string in the "Input" text box `CallItem`. The result of the call is assigned to the `rannum` variable, which is assigned to the "Result" text box `CallItemRetVal`. The following Visual Basic code fragment specifies that the `Call` method is invoked when you click the Call button:

```
Private Sub CallRPC_Click()
   rannum = G2Gateway1.Call("G2RANDOMGENERATOR",
      Val(CallItem.Text))
   CallItemRetVal = Str(rannum)
End Sub
```

For more information on the `Call` method, see Call() on page 57.

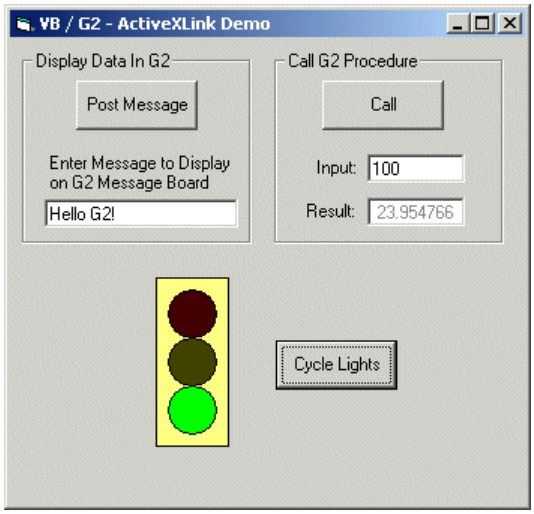
The G2RandomGenerator procedure uses the value received from Visual Basic as the value from which to generate a random number and returns the generated number to Visual Basic, as shown in the following G2 code fragment:

```
G2RandomGenerator(max: quantity) = (value)
retval: quantity;

begin
    retval = random(max);
    return retval;
end
```

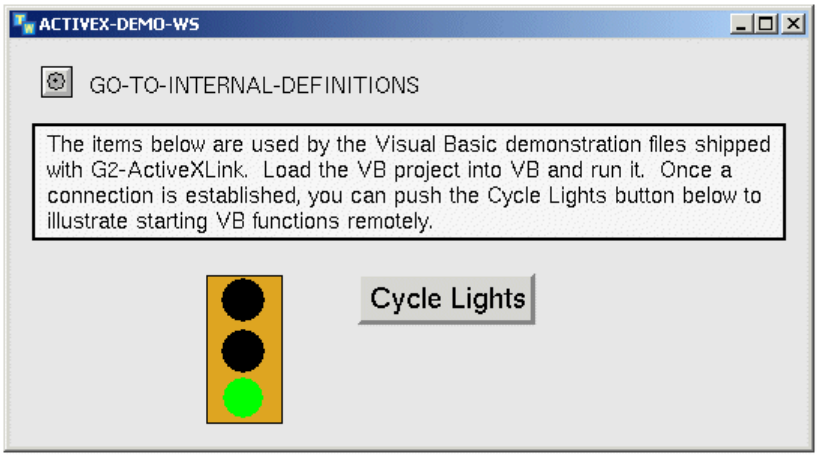
Calling a Procedure in G2 and Visual Basic

The button labeled "Cycle Lights" on the Visual Basic example form enables you to cycle through the lights in the traffic signal icon. Clicking on the Cycle Lights button causes the traffic signal to change color on both the Visual Basic form and the G2 workspace, as shown in the following figures.

Visual Basic



G2



Note For a complete listing of the Visual Basic code for the Cycle Lights program, see Using G2 ActiveXLink in Microsoft Visual Basic on page 135.

Clicking the Cycle Lights Button in Visual Basic

When you click the Cycle Lights button on the Visual Basic example form, the `CycleLights_Click()` function uses the `Start()` method to invoke the G2 procedure change-signal. The change-signal procedure uses the traffic light mode value passed from Visual Basic to change the traffic light in G2. G2 returns the new mode to Visual Basic to set the traffic light in the example form to the same mode.

The following Visual Basic code fragment specifies that the `Start()` method is invoked when you click the Cycle Lights button:

```
Private Sub CycleLights_Click()
   Call G2Gateway1.Start("CHANGE-SIGNAL", NextMode)
   If NextMode = "stop" Then
      NextMode = "slow"
   ElseIf NextMode = "slow" Then
      NextMode = "proceed"
   Else
      NextMode = "stop"
   End If
End Sub
```

The `CycleLights_Click` function also sets the value of `NextMode` for the next time the Cycle Lights button is pressed.

For more information on the `Start` method, see Start() on page 59.

In G2, the change-signal procedure sets the traffic signal icon TS (of the class traffic-signal) to the mode specified by the `NextMode` variable in Visual Basic, as shown in the following code fragment:

```
change-signal(mode: text)

begin
    case(mode) of
    "stop":
        begin
            change the green-region icon-color of TS to black;
            change the yellow-region icon-color of TS to black;
            change the red-region icon-color of TS to red;
            conclude that the mode of TS is stop;
        end;
    "slow";
        begin
            change the green-region icon-color of TS to black;
            change the yellow-region icon-color of TS to yellow;
            change the red-region icon-color of TS to black;
            conclude that the mode of TS is slow;
        end;
    otherwise;
```

```
        begin
            change the green-region icon-color of TS to green;
            change the yellow-region icon-color of TS to black;
            change the red-region icon-color of TS to black;
            conclude that the mode of TS is proceed;
        end;

call g2com-start-over-interface("CYCLELIGHTS", the mode of TS,
    the symbol of vbdemo-interface);
end
```

When the change-signal G2 procedure executes, it fires an `RpcStarted` event in Visual Basic by using the g2com-start-over-interface G2 procedure. For more information on the g2com-start-over-interface G2 procedure, see g2com-start-over-interface on page 69.

Visual Basic responds to the Rpc`Started` event by calling the *`Update_Light()`* function, as shown in the following Visual Basic code fragment:

```
Private Sub G2Gateway1_RpcStarted(ByVal Name As String,
      InArgs As Variant)
   Dim str As String
   str = InArgs
   If Name = "CYCLELIGHTS" Then Call Update_Light(str)
End Sub
```

For more information on the `RpcStarted` event, see RpcStarted() on page 74.

The `Update_Light()` function changes the traffic light icon to match the current mode, as shown in the following Visual Basic code fragment:

```
Private Sub Update_Light(Mode As String)
   If Mode = "PROCEED" Then
      Redlight.FillColor = RedOff
      Yellowlight.FillColor = YellowOff
      Greenlight.FillColor = GreenOn
   ElseIf Mode = "STOP" Then
      Redlight.FillColor = RedOn
      Yellowlight.FillColor = YellowOff
      Greenlight.FillColor = GreenOff
   Else
      Redlight.FillColor = RedOff
      Yellowlight.FillColor = YellowOn
      Greenlight.FillColor = GreenOff
   End If
End Sub
```

Clicking the Cycle Lights Button in G2

When you click the Cycle Lights button in G2, you invoke the advance-signal procedure, as shown in the following code fragment:

```
advance-signal()

begin
    case(the mode of TS) of
        stop: call change-signal(“proceed”);
        slow: call change-signal(“stop”);
        proceed: call change-signal(“slow”);
    end;
end
```

The advance-signal procedure determines the traffic signal mode and calls the change-signal procedure with the new mode.

Using G2 ActiveXLink with Visual Basic .NET

The examples in this section are drawn from the demonstration Visual Basic .NET project shipped with G2 ActiveXLink. The default location is `\activexlink\demos\vbnetdemo\bin\VBNetDemo.exe`.

You can also run this example from the Start menu by choosing:

Start > Programs > Gensym G2 2011 > Examples > ActiveXLink > VB .NET Project and VB .NET Demo

The examples described in this section apply only if you are using VB .NET.

Visual Basic .NET Terminology

G2 ActiveXLink provides a sample program for Visual Basic .NET (VB.NET). When discussing the use of G2 ActiveXLink with VB.NET, we use Microsoft's terminology for .NET. First, we will explain that terminology.

Before you can run a program built with .NET technology, the **.NET Framework** must be installed on your computer. The .NET Framework consists primarily of two parts: the **Common Language Runtime** (CLR) and the framework class library.

The part of the Framework of interest to us is the Common Language Runtime. This can be thought of as a layer of software that sits between the operating system and .NET applications. It provides numerous fundamental services to .NET applications such as memory management, thread control, and security.

Generally, Visual Studio .NET is used to build **managed code**, that is, code that runs using the Common Language Runtime. It is called managed code because it uses CLR to manage object lifetime, memory management, bound checking, etc.

G2 ActiveXLink is a **COM** object. COM is the acronym for Component Object Model. It is a specification for designing, developing, supporting, and using software components. In some respects, COM was Microsoft's predecessor to .NET.

The .NET equivalent to a COM object is a **.NET assembly**. They are similar in terms of the functionality they provide, but internally they are very different. They are not directly compatible.

To use a COM object with .NET, you need a .NET assembly that provides the **interop layer**, Microsoft's terminology for software that translates between COM and .NET. Fortunately, when you use COM objects with Visual Basic .NET, it can automatically build the required interop assembly for you. The assembly that VB.NET builds for you provides a **runtime callable wrapper** (RCW).

With this background, let's examine how we could build a .NET version of the G2 ActiveXLink demo.

Using ActiveXLink with Visual Basic .NET

`VBNetDemo` combines three separate demonstrations on a single form: posting a message to G2, calling a G2 procedure and displaying the returned value, and the traffic light demonstration. Each of these demonstrations uses a `G2Gateway` to communicate with G2.

To add G2 ActiveXLink to a VB.NET project:

1. After installing G2 ActiveXLink on your system and opening your VB.NET project, right click on the References folder in the Solution Explorer Window.

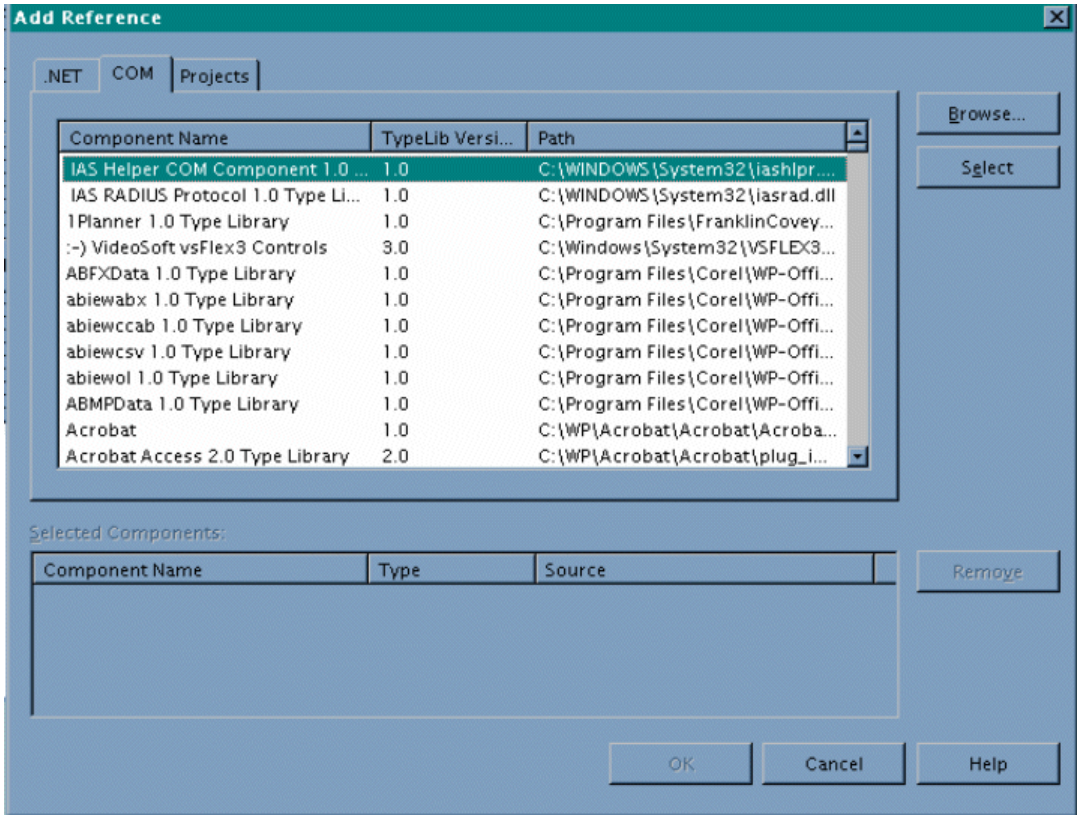
   If the Solution Explorer is not open, you can open it from the View menu. If it is open but you cannot see the References folder, then you probably need to click the Solution Explorer tab at the bottom of the window (VB.NET 2001 and VB.NET 2003) or the Show All Files icon at the top of the window (VB.NET 2005).

2. Click Add Reference.

   The Add Reference dialog appears.

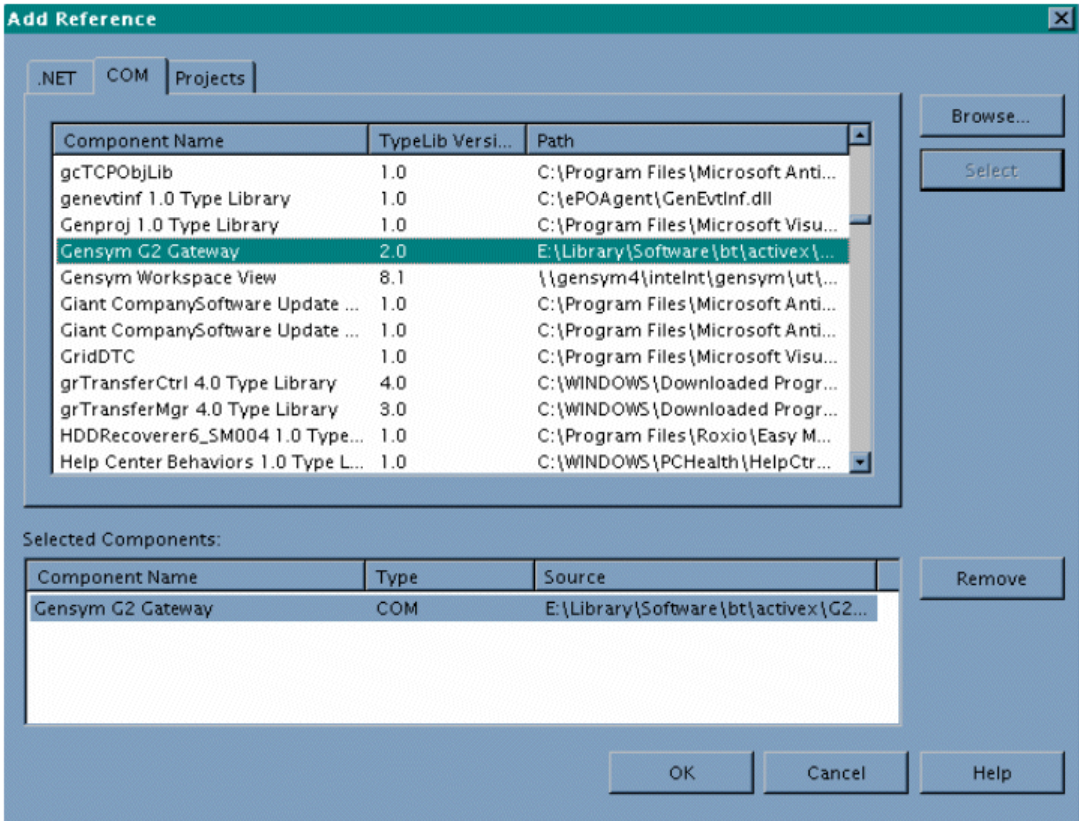
3 Click the COM tab at the top of the dialog.

It may take several seconds for the program to build the list of COM components:



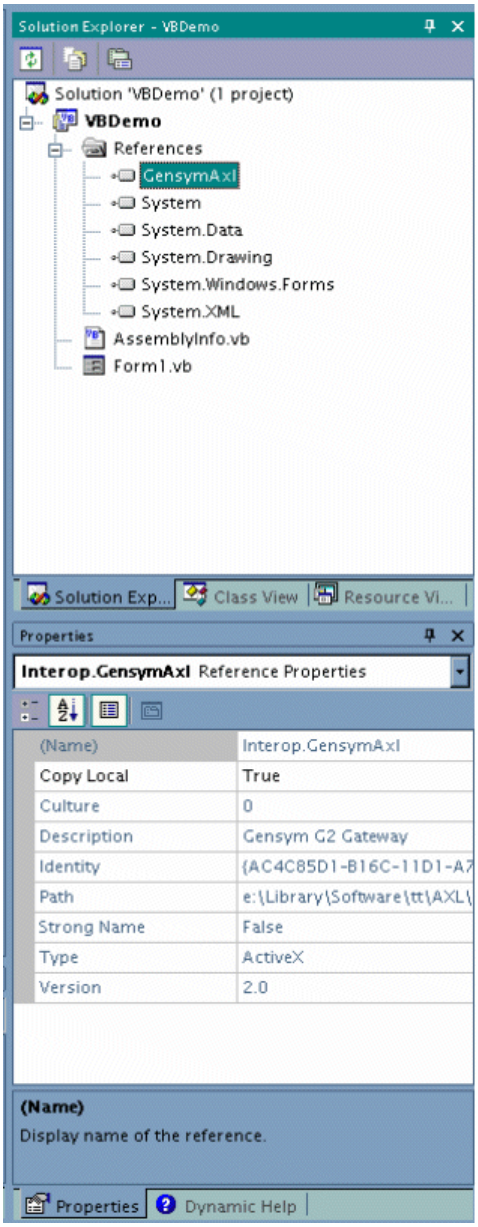
4 Scroll down the list to the component named Gensym G2 Gateway, click to select it, then click the Select button.

If there is more than one, select the one with TypeLib Version 2.0. Gensym G2 Gateway appears in the list of Selected Components at the bottom of the dialog box.



5 Click the OK button.

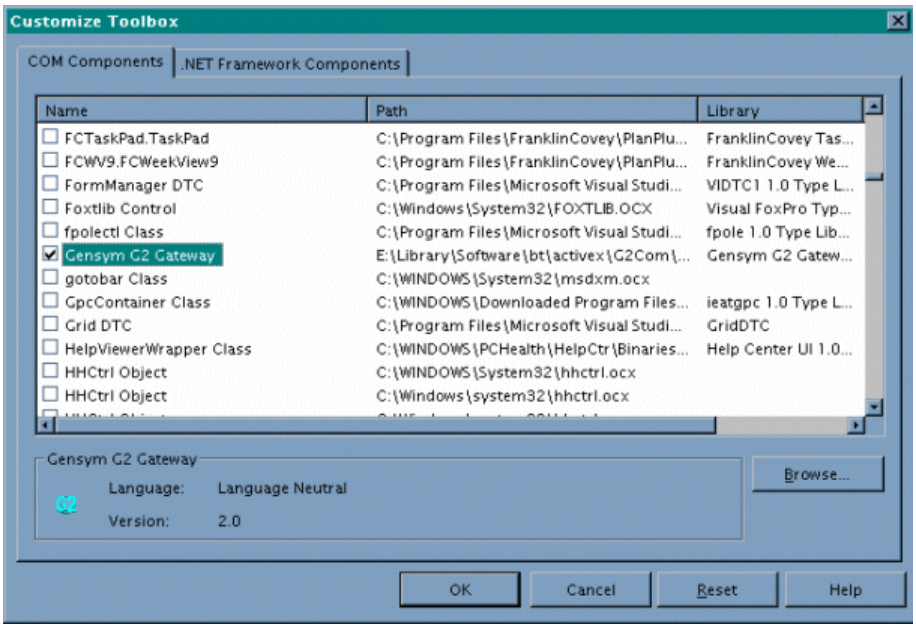
A new entry, GensymAxl, appears under the References icon in the Solution Explorer. This represents a .NET assembly that VB.NET just built for you. This assembly contains the Runtime Callable Wrapper for G2 ActiveXLink. Your program will work with GensymAxl, which will, in turn, work with G2 ActiveXLink.



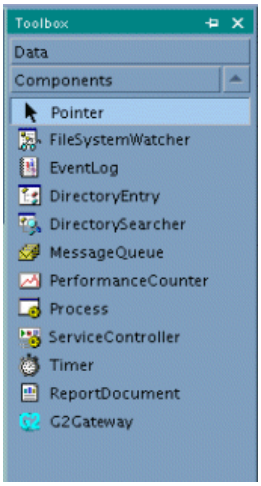
The .NET representation of a G2Gateway is an AxG2Gateway. To add an AxG2Gateway to a form, you must first add it to the toolbox. You use one of the Tools menu commands to do this. The name varies with each version of

Visual Studio .NET but the choice should be clear. For example, this command is called Choose Toolbox Items in Visual Studio .NET 2005.

6 Click the COM Components tab, scroll down to Gensym G2 Gateway, click its associated check box, then click OK.



7 Open the toolbox to see the icon for AxG2Gateway:



8 Add the component that is labeled G2Gateway to a form and fill in its properties just as you would with any other control.

Programs that use AxG2Gateways

Your program will not work directly with a G2Gateway. It will work with the .NET assembly that VB.NET built for you, an `AxG2Gateway`. The parameters to `AxG2Gateway` methods are packaged differently than those to a G2Gateway. Likewise, `AxG2Gateway` packages return parameters from events differently than G2Gateway does.

Suppose you want to call a G2 procedure named consumer, which accepts a string, an integer, and a floating point number and which returns an integer. To do this with a G2Gateway and Visual Basic 6 (VB6), your command would have looked something like:

```
Dim X As Integer
X = G2Gateway1.Call("Consumer", "XYZ", 3, 3.34)
```

Since the G2 procedure expects three parameters, you provided three parameters after the procedure name in the `Call` invocation.

As C++ programmers know, Visual Basic actually built a SafeArray of Variants from these three parameters and sent the SafeArray to ActiveXLink. Visual Basic hid this complexity from you.

Visual Basic .NET no longer has the `Variant` type. Instead, it uses the `Object` type. Unlike VB6, it does not hide the complexity from you. Instead of sending the three individual parameters to `AxG2Gateway`, you must create an array of `Object` types, fill it in with the parameters, then send the array to the `AxG2Gateway`. The one exception to this rule is when you are passing zero or one parameters, in which case, Visual Basic .NET will *cast* your input to an array of `Object` types.

Thus, in Visual Basic .NET, you would rewrite the prior example as:

```
Dim X As Integer
Dim ParamOut(2) As Object
ParamOut(0) = "XYZ"
ParamOut(1) = 3
ParamOut(2) = 3.34
X = G2Gateway1.Call("Consumer",ParamOut)
```

Visual Basic .NET event handlers are significantly different from those of VB6. There are three differences to note:

- Instead of using the name of the procedure to determine which procedure handles which event, the keyword `HANDLES` is used to tie a procedure to an event. For example, in VB6, the procedure with the signature:

  ```
  Private Sub StartRPC_Click()
  ```

handles the `Click` event for the button named `StartRPC`. With VB.NET the signature of this event handler is:

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

Although this procedure has the same name as the VB6 handler, it is the `Handles StartRPC.Click` statement that makes this the event handler, not the name of the procedure. The procedure could be renamed at it would still handle the `Click` event of the button named `StartRPC`.

- The first parameter received by a VB.NET event handler is a reference to the object that caused the event.

  In the above example, the VB6 event handler did not receive any parameters. However, the Visual Basic .NET event handler received two parameters.

  The first of these parameters is the reference to the object that caused the event, i.e., the button named `StartRPC`. Executing:

  ```
  MsgBox (TypeName(sender) & " named " & sender.Name)
  ```

  in the event handler would cause the text `Button named StartRPC` to be displayed in a message box.

- Whereas the number of parameters an event handler would receive in a VB6 program depended upon the event type, the number is always two with VB.Net. As explained above, the first is the reference to the object that caused the event. The second is a single structure that contains all the parameters that would have been delivered to a VB6 event handler.

  In the example above, the VB6 event handler did not receive any parameters. As a result the parameter named `e` in the VB.NET event handler is an empty structure.

  To provide another example, in the VB6 version of `VBDemo`, the procedure that handles error events is:

  ```
  Private Sub G2Gateway1_Error(ByVal ErrorMessage As
              String, ByVal ErrorCode As Long,
              DeferredCallIdentifier As Variant)
     MsgBox ErrorMessage
  End Sub
  ```

  The equivalent VB.NET event handler is:

  ```
  Private Sub G2Gateway1_Error(ByVal eventSender As
              System.Object, ByVal eventArgs As
              AxGensymAxl._G2Events_ErrorEvent)
     Handles G2Gateway1.Error
     MsgBox(eventArgs.errorMessage)
  End Sub
  ```

Notice the use of the dot notation to access the error message in the `MsgBox` command.

The Program

All three parts of the demo program use an AxG2Gateway to communicate with G2. Add the AxG2Gateway to your form as explained earlier. In the VB.NET demo, the name is `G2Gateway1`.

The PostMessage Demo

The VB.NET code to post a message to the G2 Message Board appears almost identical to the equivalent VB code. The only difference is the use of the parentheses with VB.NET.

The VB.NET code is:

```
G2Gateway1.PostMessage(MsgToPost.Text)
```

It is not necessary to place `MsgToPost.Text` in an array of Objects because `PostMessage` only passes a single parameter to G2 ActiveXLink.

The Call Demo

The `Call` method requires two parameters. The first is the name of the procedure to be called. The second is the array of Objects containing all the parameters required by the G2 procedure. The VB6 version would have passed each of the parameters individually to the `Call` statement. In other words, the VB6 code:

```
rannum = G2Gateway1.Call("G2RANDOMGENERATOR", Val(MaxVal.Text))
CallItemRetVal = str(rannum)
```

is equivalent to the following VB.NET code:

```
Dim InArgs(0) As Object
Dim rannum As Double

InArgs(0) = Val(MaxVal.Text)
rannum = G2Gateway1.Call("G2RANDOMGENERATOR", InArgs)
CallItemRetVal.Text = CStr(retVal) .
```

In the actual VBNetDemo program, the Try/Catch construct has been added to provide error handling.

The Traffic Light Demo

In the VB6 Traffic Light Demo, the procedure `Update_Light` uses the `FillColor` attribute of the Shape objects representing the lights to turn them on or off.

VB.NET graphics are not as easy to use as those of VB6. Shape objects are no longer available. Instead, it is necessary to use low-level Windows objects such as Graphics, BitMaps, and Brushes to display the traffic light.

The complexity of VB.NET traffic light demo is encapsulated entirely within the `TrafficLight` class. For details, look at the class definition in `TrafficLight.vb`.

An instance of `TrafficLight` is placed on the main form by defining it as a member variable of the form. The button to change its state simply sets the `CurrentState` attribute to the correct new state. The definition of `Set` in the class definition is responsible for updating the display.

After setting the `CurrentState`, the button's click event then Starts a G2 procedure to tell G2 to update the state of the G2 traffic light.

When G2 changes the state of the light, it uses `G2Com-Start` to inform the VB.NET program. The event handler in the VB.NET program simply changes `CurrentState` of the `TrafficLight` object on the form. The class definition takes care of updating the display.

Using G2 ActiveXLink with Microsoft Excel

In Excel, you can use the G2 ActiveXLink to call or start a procedure in G2. The link between G2 and Excel enables you to retrieve data from G2 for display in a spreadsheet. You can send data to occupy a single cell or send data that fills a block of cells.

The examples in this section are drawn from the demonstration spreadsheet shipped with G2 ActiveXLink. The default location is `\activexlink\demos\exceldemo\Gateway.xls`.

You can also run this example from the Start menu by choosing:

Start > Programs > Gensym G2 2011 > Examples > ActiveXLink > Excel Demo

---

Note The ActiveXLink Excel demo only works with Excel 2000 or later. The demo is located in the `\activexlink\demos\exceldemo` directory of your G2 product directory.

Making a Connection to G2

You can use the G2 ActiveXLink control on an Excel spreadsheet to send and receive data over the connection with G2. You must have G2 running and not paused to make the connection from Excel. You can connect to any G2 running anywhere in your TCP/IP network.

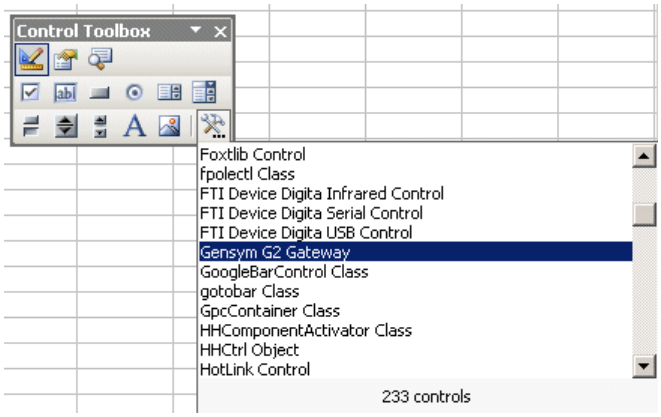
Placing the Control in Your Spreadsheet

You can place the control in your spreadsheet by getting the control from the Control Toolbox, as shown in the following figure:

The Control Toolbox contains three icons for handling controls and their properties and code, as shown in the following figure:

To add the control from the Toolbox:

1. Enter Design Mode by clicking on the Design Mode icon in the Control Toolbox.
2. Click on the More Controls icon in the Control Toolbox, as shown in the following figure:



3. Click on `Gensym G2 Gateway` in the list of controls.

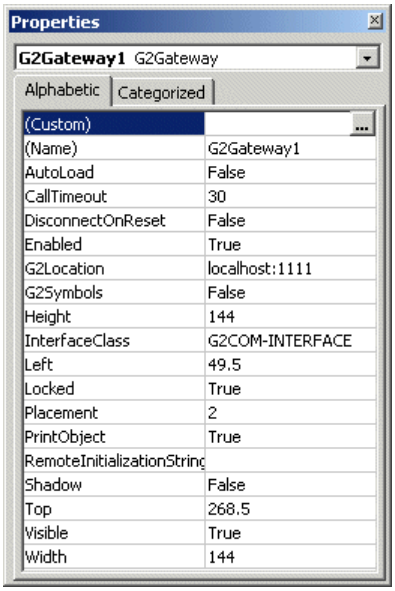
Excel adds the control to the spreadsheet. It appears as a white box, outlined with handles when you select it. The control disappears when you exit Design Mode.

Setting the Properties of the Control

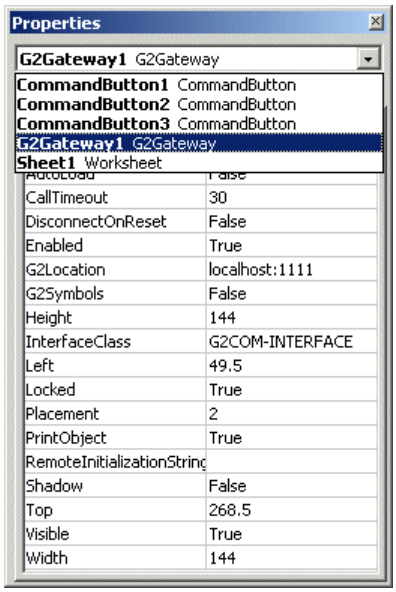
Like other controls in Excel, the G2 ActiveXLink control has properties you can set.

Using the Properties Window

You can set properties by clicking on the Properties icon in the Control Toolbox to display the Properties Window, as shown in the following figure:



Select the G2 ActiveXLink control in the dropdown listbox, as shown in the following figure:



You can set properties by typing values in the text boxes in the Properties Window. If you are modifying the four basic G2 ActiveXLink properties, use the Properties dialog, described in the next section.

Using the Properties Dialog

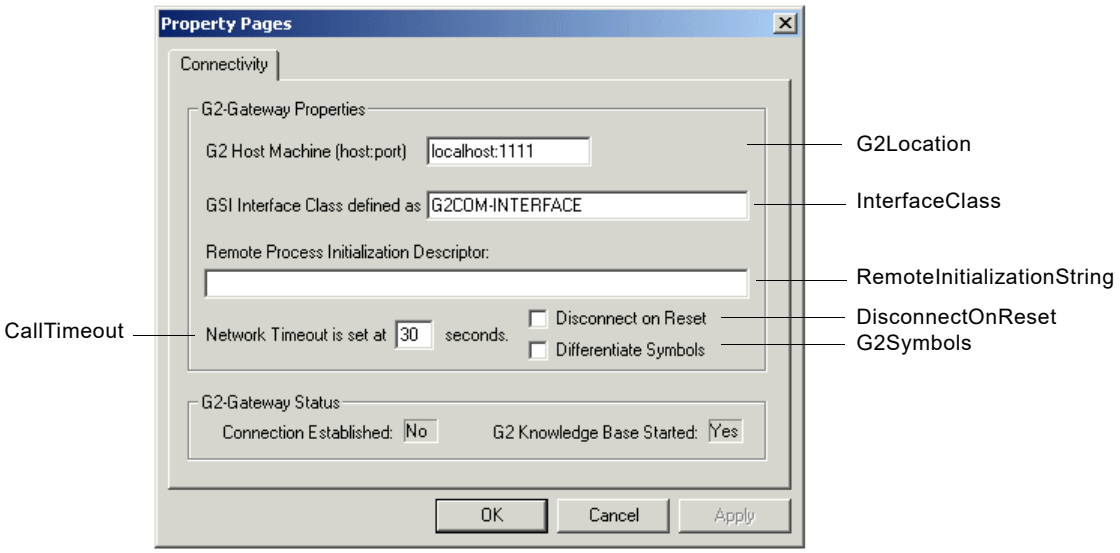
The Properties dialog enables you to set these four basic properties:

- `CallTimeout`
- `G2Location`
- `InterfaceClass`
- `RemoteInitializationString`

When you click in the text box for one of these properties, the following button with three dots appears:

To display Properties dialog for the G2 ActiveXLink control, click this button.

The following figure shows the Properties dialog with the properties you can set for the G2 ActiveXLink control:

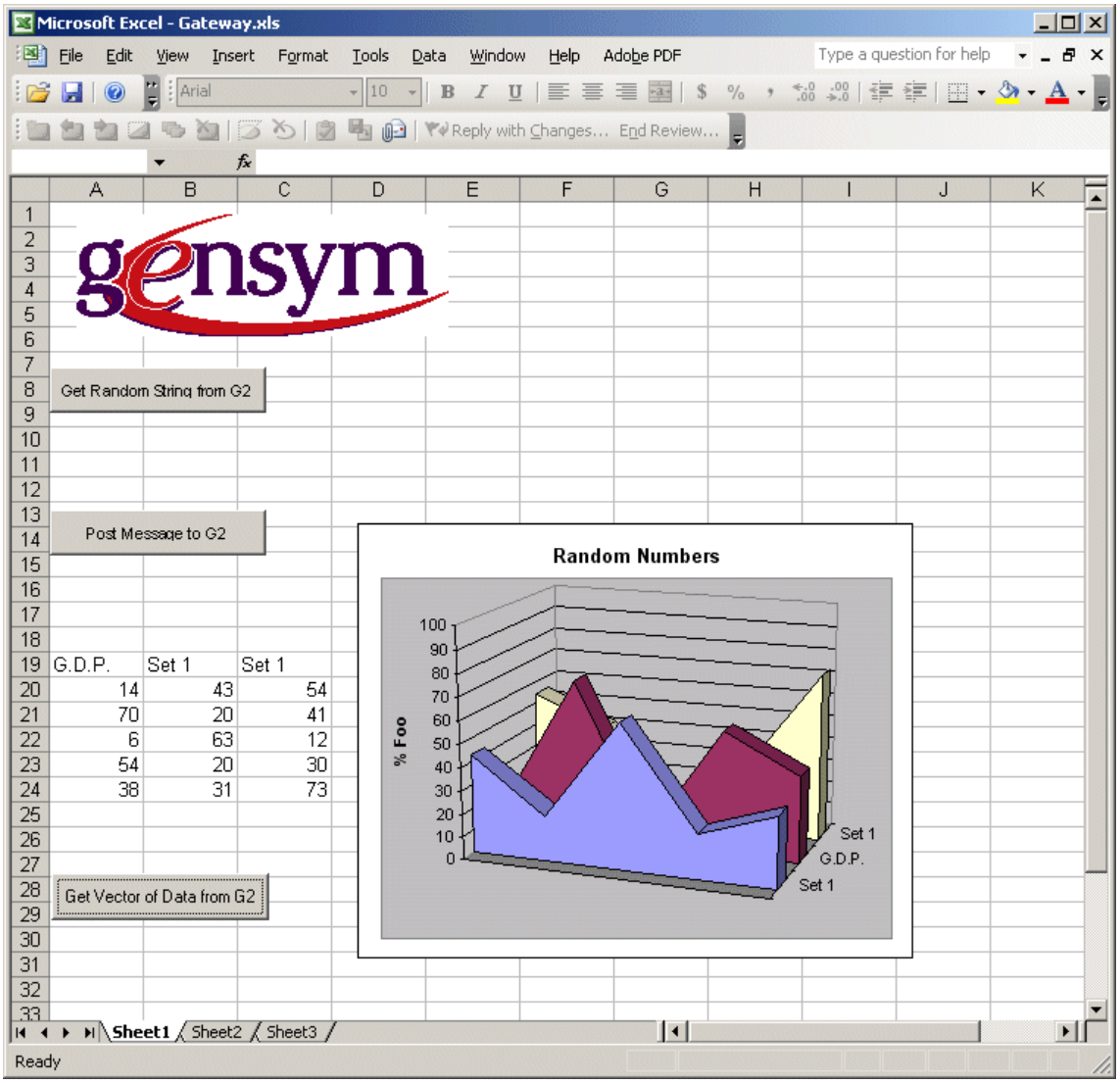


For more information on setting properties, see Setting the Basic Properties on page 11 and Setting Properties Programmatically on page 13.

Calling a Procedure in G2 and Excel

Using the G2 ActiveXLink, you can call a procedure in G2 and use the return values to populate cells in an Excel spreadsheet. The following example uses a command button to call a G2 procedure that generates random numbers, which are returned to specified cells in Excel. A chart uses the data in the cells to update the display for each cell.

The following figure shows a chart in which a three-dimensional display is mapped to the values in the cells to its left. Each time the button named "Get Vector of Data from G2" is pressed, the display updates with data from G2.



The "Get Vector of Data from G2" button uses the `Call()` method to invoke the G2ChartGenerator G2 procedure, which includes a value for generating a random number. The vector returned from G2 contains five values, which are returned to the three columns of cells in Excel, as shown in the following code fragment:

```
Private Sub CommandButton3_Click()
   Range("A19:C24") = G2Gateway1.Call("G2ChartGenerator",
      100) ' Get values from G2
End Sub
```

The G2ChartGenerator G2 procedure calculates three columns of five values between 0 and 100 and retains the values, which are returned to Excel as a G2 structure data type, as shown in the following code fragment:

```
G2ChartGenerator(max: quantity) = (structure)

Titles: sequence = sequence("Set 1", "G.D.P.", "profits", "R.O.I.", "Stock Price",
     "Yield", "Level");
Set1: sequence = sequence(Titles[random(6)], random(max), random(max),
     random(max), random(max), random(max),
Set2: sequence = sequence(Titles[random(6)], random(max), random(max),
     random(max), random(max), random(max),
Set3: sequence = sequence(Titles[random(6)], random(max), random(max),
     random(max), random(max), random(max),

begin
     return structure(com-dimensions: sequence(6,3), com-lower-bounds: 1,
          com-elements: concatenate(Set1, Set2, Set3));
end
```

G2 ActiveXLink converts the returned G2 structure to a two-dimensional array of values. These values are placed in the Excel spreadsheet cells by using the `Range()` function of Excel.

Using G2 ActiveXLink with Microsoft Internet Explorer

You can use the G2 ActiveXLink in Microsoft Internet Explorer to call or start a procedure in G2. The link between G2 and Internet Explorer enables you to retrieve data from G2 for display in a browser.

Adding the G2 ActiveXLink Control to an HTML File

You can specify a connection to G2 in an HTML file by using standard HTML markup and the G2 ActiveXLink control. You can add G2 ActiveXLink by using the HTML `object` tag, as shown in the following HTML fragment:

```
<p><object id="G2Gateway"
classid="CLSID:AC4C85D0-B16C-11D1-A718-006008C5F933"
align="baseline" border="0" width="163" height="33" <></object>
```

The attribute `classid` identifies G2 ActiveXLink as a registered ActiveX control.

Connecting with G2 on Startup

When you open a page with G2 ActiveXLink control in Internet Explorer, the browser displays an Explorer User Prompt dialog to get information on the running and started G2 to which you want to connect.

The default text is localhost:1111, as shown in the following figure:

The connection is specified in VBScript, which is embedded in HTML. The following VBScript specifies the connection with a G2 server:

```
Sub window_onLoad()
myNumber = 100
msg = "Enter the G2 host:port:"
initialTxt = G2Gateway.G2Location
G2Gateway.G2Location = window.prompt(msg, initialTxt)

G2Gateway.RemoteInitializationString = window.navigator.appName
' Make connection -- connect happens upon first method call.
lResult = G2Gateway.Connect(TRUE)
end sub
```

When the connection is established, the following messages appear in Internet Explorer and the Message Board in G2:

The following Visual Basic code fragment specifies the response to the `G2Connected` event. The code specifies the message displayed by Internet Explorer. The code uses the `PostMessage()` method to display the message in the G2 Message Board.

```
Sub G2Gateway_G2Connected()
msg = "Connected to G2 at " + G2Gateway.G2Location
call window.alert(msg)
call G2Gateway.PostMessage("Hello from Internet Explorer")
end sub
```

Sending a Message to G2 from Internet Explorer

In Internet Explorer, you can send a message to G2 by clicking a button after you specify the text with another button.

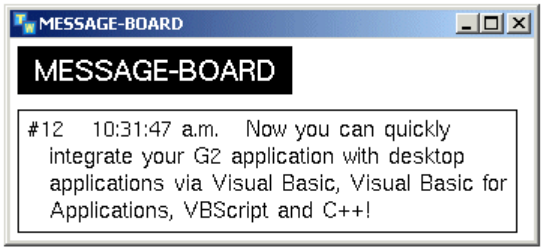
The following HTML code specifies the Post Above Message to G2 (`CommandButton2`) button. Various parameters (`param`) specify its properties:

```
<p><object id="CommandButton2" name="CommandButton2"
classid="clsid:D7053240-CE69-11CD-A777-00DD01143C57"
align="baseline" border="0" width="203" height="32">
   <param name="Caption" value="Post Above Message to G2">
   <param name="Size" value="5362;847"><param name="FontHeight"
      value="200">
   <param name="FontCharSet" value="0">
   <param name="FontPitchAndFamily" value="2">
   <param name="ParagraphAlign" value="3">
</object></p>
```

The following VBScript specifies that, when the Post Above Message to G2 button is clicked, the text in the text box, `TextBox1`, is displayed in G2 Message Board:

```
Sub CommandButton2_Click()
Call G2Gateway.PostMessage(Form1.TextBox1.Text)
end sub
```

The text in the text box in Internet Explorer appears on the Message Board in G2, as shown in the following figure:



Using G2 ActiveXLink with C++

G2 ActiveXLink includes an example of using ActiveXLink with C++. The example in this section is drawn from the demonstration file shipped with G2 ActiveXLink. The location is `\activexlink\demos\vcppdemo\VCppDemo`.

You can also run this example from the Start menu by choosing:

Start > Programs > Gensym G2 2011 > Examples > ActiveXLink > C++ Demo and C++ Project

G2 ActiveXLink, like all ActiveX objects, is based upon the Component Object Model (COM). Languages such Visual Basic hide the complexity of using COM objects. C++ does not.

A recent search of the web site of a popular bookseller showed there to be 125 books about the Component Object Model. There are so many books on the subject because there is so much to know about it. The moment you decide to use a COM object in C++, you will be faced with numerous decisions such as:

- Will you create objects by dropping the class into a Windows form, or will you dynamically instantiate instances of it?
- If you choose dynamic instantiation, will you use `CoCreateInstance`, will you create a class factory and then use it to create one or more instances of your class, or will you use some other technique?
- Will you use the `#import` extension of Microsoft Visual Studio to create smart pointers to your objects, or will you manage object life will calls to `AddRef` and `Release`.
- How are you going to add the GUIDs (Globally Unique Identifiers) for the COM elements to your program? With `#import`? By letting MFC generate a wrapper class? Manually?
- Are you going to use the fundamental COM data types such as `VARIANT` and `BSTR`, are you going to use the Visual Studio extensions such as `_bstr_t` and `_variant_t`, or are you perhaps going to use the smart ATL classes such as `CComBSTR` and `CComVariant`.

Furthermore, you will need to work with new data types such as `BSTR`, `HRESULT`, `SAFEARRAY` and `VARIANT`. You will also need to manage them carefully to avoid memory leaks.

Thus, even if you are an experienced C++ programmer, you need to understand COM to program with G2 ActiveXLink.

To facilitate programming with G2 ActiveXLink in C++, this example provides a console application that writes `"Hello, G2."` to the G2 Message Board. It uses the Visual Studio `#import` extension, which creates the `IG2GatewayPtr` class. It then uses this class to both create an instance of a `G2Gateway` and to access its methods.

```
// ====================================================================
//
// An Example of the Use of ActiveXLink in a C++ Program
// This program send the message "Hello, G2." to the G2 Message Board.
//
// ====================================================================
#include <iostream.h>

// Use Visual Studio extensions to simplify the task
// ------------------------------------------------
#import "c:\zd\bt\dw\activex\g2com\release\g2com.dll" no_namespace
```

```
named_guids

// Prototype:
// ----------
int demoError(int errorCode, HRESULT hr) ;

int     main()
{
    HRESULT         hr ;
    IG2GatewayPtr   pAxl ;

    // Initialize COM
    // --------------
    hr = ::CoInitialize(NULL) ;
    if (FAILED(hr))
        return demoError(0, hr) ;




    // Create an instance of G2Gateway
    // -------------------------------
    hr = pAxl.CreateInstance(CLSID_G2Gateway, NULL) ;
    if (FAILED(hr))
        return demoError(1, hr) ;

    // Specify location of G2
    // ----------------------
    BSTR    g2Loc = ::SysAllocString(L"Porangatu:1111") ;

    hr = pAxl->put_G2Location(g2Loc) ;
    ::SysFreeString(g2Loc) ;
    if (FAILED(hr))
        return demoError(2, hr) ;

    // Specify how long we should wait for a response from G2
    // ------------------------------------------------------
    hr = pAxl->put_CallTimeout(15L) ;
    if (FAILED(hr))
        return demoError(3, hr) ;

    // Connect to G2, waiting for the connection.
    // ------------------------------------------
    hr = pAxl->Connect(VARIANT_TRUE) ;
    if (FAILED(hr))
        return demoError(4, hr) ;

    // Say hello to G2
    // ---------------
    VARIANT Msg ;
    VariantInit(&Msg) ;
```

```
    Msg.vt = VT_BSTR ;
    Msg.bstrVal = ::SysAllocString(L"Hello, G2.") ;

    hr = pAxl->PostMessage(&Msg) ;
    VariantClear(&Msg) ;
    if (FAILED(hr))
        return demoError(5, hr) ;
    // Disconnect from G2
    // ------------------
    hr = pAxl->Disconnect() ;
    if (FAILED(hr))
        return demoError(6, hr) ;

    // Release the COM resources
    // -------------------------
    ::CoUninitialize() ;
    return 0 ;
}


// There was an error. Report the error to cerr and tell
// the COM system that we are done with it.
// ================================================
    char*   errMsg[] =  { "Unable to initialize COM.",
                          "Unable to create an instance of G2Gateway.",
                          "Failed to specify location of G2.",
                          "Failed to set timeout.",
                          "Failed to connect to G2",
                          "Failed to send message to message board.",
                          "Disconnect from G2 failed."} ;
    const int NR_ERR_CODES = sizeof(errMsg)/sizeof(char*) ;

int demoError(int errorCode, HRESULT hr)
{
    int retCode = -errorCode ;

    if (errorCode >= 0 && errorCode < NR_ERR_CODES)
        cerr << errMsg[errorCode] ;
    else
    {
        cerr << "Unknown error code (" << errorCode << ")." ;
        retCode = -NR_ERR_CODES ;
    }
    cerr << "  (" << hex << hr << dec << ")." << endl ;
    if (errorCode > 0)
        ::CoUninitialize() ;
    return retCode ;
}
```

Another way to use G2 ActiveXLink in C++ is with MFC (Microsoft Foundation Classes). This technique inserts a "wrapper class" for you when you insert ActiveXLink into a form. You can then use the Class Wizard to associate a member variable with the `G2Gateway` (ActiveXLink) object. You can use this variable to call ActiveXLink methods in much the same way you would in Visual Basic. The Class Wizard also makes it very easy to add functions to handle ActiveXLink events.

Unfortunately, MFC does not wrap the methods that take a `SAFEARRAY` of `VARIANT` values as parameters. Because COM requires that procedures that can accept a variable number of parameters pass them in a `SAFEARRAY` of `VARIANT` values, MFC does not help you with some of the commonly used ActiveXLink Methods such as `Call` and `Start`.

3

Data Types

The chapter describes the G2 ActiveXLink data types.

Introduction **43**

Mapping Data Types **44**

Overriding the Mapping of Simple Data Types **45**

Mapping Date and Time **46**

Mapping Currency **47**

Mapping Multidimensional Arrays **47**

Introduction

ActiveX controls are designed to work with various development languages such as COM, Visual Basic, or G2's natural language. Each language has its own data types. G2 ActiveXLink automatically maps data types between a container application and a G2 server.

Mapping Data Types

The following table describes the corresponding COM automation, Visual Basic, and G2 data types, starting with COM automation types:

COM Automation Type	Visual Basic Type	G2 Type
VARIANT_BOOL	Boolean	truth-value
long	Long	integer
short	Integer	integer
double	Double	float
date	Date	structure
currency	Currency	structure
byte	Byte	structure
BSTR	String	text
VARIANT	Variant	item-or-value
SAFEARRAY (VARIANT)	MyData() As Variant	sequence
null	Null	the symbol null

The following table describes the corresponding COM automation, Visual Basic, and G2 data types, starting with G2 data types:

G2 Type	COM Automation Type	Visual Basic Type
truth-value	VARIANT_BOOL	Boolean
quantity	VARIANT	Variant
integer	long	Long
float	double	Double
symbol	BSTR or null	String or Null
text	BSTR	String
item-or-value	VARIANT	Variant
value	VARIANT	Variant

G2 Type	COM Automation Type	Visual Basic Type
structure	`IDispatch`	`G2Structure`
sequence	`SAFEARRAY (VARIANT)`	`MyData() As Variant`
the symbol null	`null`	`Null`
simple data types	`single, double, integer, long`	`Single, Double, Integer, Long`
date structure	`date`	`Date`
currency structure	`currency`	`Currency`

G2 ActiveXLink supports 16-bit characters, which means it correctly represents Unicode characters, as well as any character in any of the extended G2 character sets.

Note The range of an integer in G2 is -2^{29} to $(2^{29} - 1)$. The range of a `long` in Visual Basic is–2,147,483,648 to 2,147,483,647. The range of a float in G2 is ±1.79 x 10^{308} to ±2.22 to 10^{-308}. The range of a double in Visual Basic is ±1.79769313486231^{308} to ±4.94065645841247^{-324}.

Overriding the Mapping of Simple Data Types

By using a G2 structure, you can override the default mapping of some G2 data types to Visual Basic data types. You can use the Visual Basic data types `Single`, `Double`, `Integer`, and `Long` by populating a G2 structure.

Visual Basic Data Type	G2 Structure Member	G2 Data Type
`Single`	com-single	float
`Double`	com-double	float
`Byte`	com-byte	integer
`Integer`	com-integer	integer
`Long`	com-long	integer

For example, to pass 56.249 as a `Single` to the container application, type in G2:

```
structure(com-single: 56.249)
```

Note Loss of precision may result when translating from one data type to another.

Mapping Date and Time

Date and time are mapped to a structure in G2. The structure can be composed of the following members:

G2 Structure Member	G2 Data Type	Range
com-year	integer	(any integer)
com-month	integer	1-12
com-day	integer	1-31
com-hour	integer	0-23
com-minute	integer	0-59
com-second	integer	0-59
com-day-of-week	integer	0-6 Sunday = 0

Because these members are also keywords in G2, the prefix "com" identifies them to G2 ActiveXLink.

For example, to pass the date March 15, 1998 to a container application, type in G2:

```
structure(com-month: 3, com-day: 15, com-year: 1998)
```

When a COM-compliant container passes a date to G2, G2 creates a `com-day-of-week` member for the date's structure. When G2 passes a date structure to a container, the container ignores the `com-day-of-week` member.

The `com-hour`, `com-minute`, and `com-second` members are optional when G2 passes a date structure to a container. If not present in the structure, the values of these members are zero.

Mapping Currency

The members of a structure for currency in G2 use the float data type.

G2 Structure Member	G2 Data Type
com-currency	float

For example, to pass an amount of $10.51 to a container application, type in G2:

```
structure(com-currency: 10.51)
```

Note Loss of precision may result when translating from one data type to another.

Mapping Multidimensional Arrays

You can map multidimensional arrays to a structure in G2 by mapping to a G2 sequence of values as defined in the following table:

G2 Structure Member	G2 Data Type
com-dimensions	sequence
com-lower-bounds	sequence
com-elements	sequence
com-array-type	symbol

The following table describes the G2 structure members of a mapped multidimensional array:

Member	Description
com-dimensions	Dimensions of the multidimensional array, for example, (2,3). If you omit the com-dimensions member, the array becomes a one-dimensional array equal to the number of members passed.
com-lower-bounds	Location of the first element in the array, for example, (1,1), (5,1), or (0,6). When Visual Basic passes a multidimensional array to G2, G2 ActiveXLink creates the com-lower-bounds member. When G2 passes a multidimensional array with the com-lower-bounds member omitted, G2 ActiveXLink sets the com-lower-bounds member to zero. You can use a single integer for the com-lower-bounds member if all the dimensions of the multidimensional array have the same lower bound, usually zero or 1. For example: com-lower-bounds: sequence(1)
com-elements	Values for each element of the array.
com-array-type	Data type for all elements of the array. Default is com-variant. You can use one of these data types: • com-boolean • com-byte • com-currency • com-date • com-double • com-float • com-integer • com-long • com-scode (for arrays of error codes) • com-string • com-variant

Example: One-Dimensional Array

The following G2 structure passes or receives the array:

```
structure(
    com-lower-bounds: 1,
    com-elements: sequence(“some”,”thing”,5),
    com-array-type: the symbol com-variant
)
```

The following Visual Basic code fragment passes or receives the array:

```
Dim Arr(1 to 3) as Variant
```

Example: Two-Dimensional Array

The following G2 structure passes or receives the array:

```
structure(
    com-dimensions: sequence(2,2),
    com-elements: sequence(“some”,”thing”,3,4),
    com-array-type: the symbol com-variant
)
```

The following Visual Basic code fragment passes or receives the array:

```
Dim Arr(0 to 1,0 to 1) as Variant
```

Example: Multidimensional Array

Suppose you pass the following 4 x 3 multidimensional array:

1 2 3 4

5 6 7 8

9 10 11 12

The following G2 structure passes or receives the array:

```
structure(
    com-dimensions: sequence(4,3),
    com-lower-bounds: sequence(1,1),
    com-elements: sequence(1,2,3,4,5,6,7,8,9,10,11,12),
    com-array-type: the symbol com-integer
)
```

The following Visual Basic code fragment passes or receives the array:

```
Dim Arr(1 to 4, 1 to 3) as Integer
Element Arr(4,2) is 8
```

4

Using G2Gateway

The chapter describes the G2Gateway properties, events, and methods.

Introduction **52**

Properties **52**

Methods **56**
- Call( ) **57**
- Start( ) **59**
- CallDeferred( ) **61**
- Connect( ) **63**
- Disconnect( ) **64**
- PostMessage( ) **65**

Events **66**
- g2com-call **67**
- g2com-start **68**
- g2com-start-over-interface **69**
- RpcCalled( ) **71**
- RpcStarted( ) **74**
- G2Connected **76**
- G2Disconnected **77**
- RpcReturned( ) **78**
- G2Paused **79**
- G2Resumed **80**
- G2Reset **81**
- G2Started **82**
- G2RunStateKnown **83**
- AttributeModified( ) **84**
- CustomEvent( ) **85**
- IconColorChanged( ) **86**
- ItemAdded( ) **87**
- ItemDeleted( ) **88**
- ItemRemoved( ) **89**
- ItemSelected( ) **90**
- ValueChanged( ) **91**

Introduction

`G2Gateway` provides properties, events, and methods for managing connections between G2 ActiveXLink and G2.

Properties

`G2Gateway` has the following properties, which you can set in a container application. Properties configure the behavior of the G2 ActiveXLink and indicate its status. Most container applications save the state of the properties in your document and restore them when you load the document.

The properties can be modified either by using the Property Page provided by the G2 ActiveXLink, the container's Property window, or programmatically. For a description of setting properties, see Setting the Properties of the Control on page 11.

Property	Description
`G2Location As String`	Sets the host machine name and port number of the G2 server to which G2 ActiveXLink connects. The format is "`hostname:port_number`" and the default is "`localhost:1111`," which is the network address of a G2 server running on the same system as G2 ActiveXLink at port 1111, the default G2 address.
`G2Symbols As Boolean`	Allows you to choose the behavior when sending symbols from G2. When set to `True`, the default, simple symbols are stored as instances of `G2Symbol`. When set to `False`, symbols are stored as `String` types. The exceptions to this rule are item attribute names and structure property names, which are always returned as `String` types. In both of these cases, you know that the original value in G2 was a symbol. `G2Item` instances store data in an internal format that is neither a `String` nor a `G2Symbol`. When you retrieve an item value or an attribute value that is a symbol from a `G2Item`, it is always returned as a `G2Symbol`, regardless of the value of the `G2Symbols` property of any `G2Gateway`. Lists and arrays are also items. The same rule applies to `G2ListOrArray`.
`InterfaceClass As String`	Specifies the interface class in G2. G2 creates an instance of this class when G2 ActiveXLink connects with G2. The instance represents the connection. The default is g2com-interface. This property must name a class that is defined in G2 and is a subclass of g2com-interface. You can add attributes to the subclass of g2com-interface or write rules or procedures that act on the subclass rather than the g2com-interface class.

Property	Description
`RemoteInitializationString As String`	Identifies the connection and is stored in the remote-process-initialization-string attribute of the G2 Gateway interface object, created in G2 when G2 ActiveXLink connects with it. G2 applications use this string to identify the interface, especially when there is more than one client connection to G2. You can also use the g2-current-remote-interface( ) system procedure to identify a specific interface object that launched an RPC call into G2. For more information, see the *G2 System Procedures Reference Manual*.
`IsG2Connected As Boolean`	Returns the connection status as `true` if connected or `false` if not. The `IsG2Connected` property is read only.
`CallTimeout As Long`	Sets the maximum amount of time for the G2 ActiveXLink to wait for G2 to respond. The default is 30 seconds. G2 ActiveXLink waits this length of time for a blocking call before aborting the call for both blocking connections to G2 and blocking RPC calls.

Property	Description
`G2RunState As Boolean`	Returns the G2 run state as one of the following: `g2UnknownState`, `g2Reset`, `g2Paused`, and `g2Running`. The value is initially set to `g2UnknownState` until G2 reports its run state to ActiveXLink.
`DisconnectOnReset As Boolean`	Allows you to choose the reset behavior that your application requires, as follows: • When set to `False`, the default, the connection between G2 ActiveXLink and G2 is maintained when G2 is reset. However, once you make a connection, you must delete your interface objects without permanence checks before you can save your KB. • When set to `True`, the connection between G2 ActiveXLink and G2 is broken when G2 is reset, and the interface objects are automatically deleted.

Methods

Using methods, the container application makes requests of the `G2Gateway` G2 ActiveXLink control. The methods enable you to manage the connection and invoke G2 procedures.

Note Methods that require a connection to G2, such as `Call()`, `Start()`, and `CallDeferred()`, automatically create a blocking connection to G2 if none already exists. Normally, you do not need to explicitly call the `Connect()` method.

Refer to Chapter 4, Using G2Gateway for a description of the use of some method handlers.

Call()

The `Call` method calls a procedure in G2 and waits for the G2 procedure to complete.

Visual Basic Syntax

```
Call(ProcedureName As String,
   [ParamArray InputArguments() As Variant])
   ReturnArguments As Variant
```

Microsoft Interface Description Language Syntax

```
HRESULT Call([in] BSTR ProcedureName,
   [in] SAFEARRAY(VARIANT) *InputArguments,
   [out, retval] VARIANT *ReturnArguments)
```

Arguments	Description
ProcedureName As String	The name of the G2 procedure you are calling.
ParamArray InputArguments() As Variant	Any of zero or more arguments to be passed to the G2 procedure.

Return Values	Description
ReturnArguments As Variant	Any of zero or more arguments to be returned from the G2 procedure. If a G2 procedure returns more than one value, Visual Basic can access the return values by using array subscripting, such as `Ret(1)`, `Ret(2)`, and so on. If only one value is returned, the array subscript is not necessary.

Description

The `Call` method is a "blocking call," in which the caller waits until the G2 procedure completes and returns values before executing the next statement following the call.

The `Call` method only works with G2 procedures; it does not work with G2 methods.

Note The container application can pass any number of arguments to G2 and G2 can return any number of arguments.

G2 ActiveXLink returns an exception to the caller under one of the following conditions:

- If G2 does not respond within the time specified by the `CallTimeout` property.
- If an error with the call is detected by G2 ActiveXLink.
- If the G2 procedure signals an error.

Example

Visual Basic calls the G2 procedure and passes three values: an `Integer`, a `String`, and a `Boolean`. The call waits for a return value. G2 displays these values on the Message Board and returns a `String` to Visual Basic.

Visual Basic

```
Private Sub Form_Load()
   Ret = G2Gateway1.Call("my-proc", 1, "Hello G2", True)
   MsgBox Ret
End Sub
```

G2

```
my-proc(arg1: value, arg2: value, arg3: value) = value

begin
    inform the operator that "Value1:[arg1], Value2:[arg2], Value3:[arg3]";
    return "Hello COM";
end
```

Start()

The `Start()` method starts a procedure in G2.

Visual Basic Syntax

```
Start(ProcedureName As String,
   [ParamArray Arguments() As Variant])
```

Microsoft Interface Description Language Syntax

```
HRESULT Start([in] BSTR ProcedureName,
   [in, optional] SAFEARRAY(VARIANT) *InputArguments)
```

Arguments	Description
ProcedureName As String	The name of the G2 procedure you are starting.
ParamArray Arguments() As Variant	Any of the arguments to be sent to G2. This array is a `Variant` and is constructed automatically for all remaining arguments. The array can contain one or more values.

Description

The started G2 procedure cannot return values to the container application. If G2 ActiveXLink is not already connected with G2, the `Start()` method makes the connection.

Note The container application can pass any number of arguments to G2.

Example

Visual Basic starts a G2 procedure and passes a string to it.

Visual Basic

```
Private Sub Form_Load()
   Call G2Gateway1.Start("my-started-proc",
      "my message to you")
End Sub
```

G2

```
my-started-proc(arg1: value)

begin
      inform the operator that "Message from COM:[arg1]";
end
```

CallDeferred()

The `CallDeferred()` method calls a procedure in G2, but it does not wait for the G2 procedure to complete.

Visual Basic Syntax

```
CallDeferred(ProcedureName As String,
   DeferredCallIdentifier As Variant,
   ParamArray InputArguments() As Variant)
```

Microsoft Interface Description Language Syntax

```
HRESULT CallDeferred([in] BSTR ProcedureName,
   [in] VARIANT DeferredCallIdentifier
   [in] SAFEARRAY(VARIANT) *InputArguments)
```

Argument	Description
ProcedureName As String	The name of the G2 procedure you are calling.
DeferredCall Identifier As Variant	An identifier used to match a `RpcReturned()` or `Error` event to a specific `CallDeferred()` call.
ParamArray InputArguments() As Variant	Any of the arguments to be sent to G2. This array is a `Variant` and is constructed automatically for all remaining arguments. The array can contain one or more values.

Description

If G2 ActiveXLink is not already connected with G2, the `CallDeferred()` method makes the connection.

You can specify a unique identifier (`DeferredCallIdentifier`) for the call. The `RpcReturned()` event uses this identifier to match the return values with the original call.

When the called G2 procedure completes, the `RpcReturned()` event fires, which receives the name of the G2 procedure that was called by `CallDeferred()`, the `DeferredCallIdentifier` argument, and the return arguments, if any.

Note The container application can pass any number of arguments to G2.

Use the `CallDeferred()` method when you want to send information to the G2 server, but you do not want to wait for the results of your call. The calling thread does not block and can continue to perform other processing while waiting for results.

Example

Visual Basic starts the G2 procedure my-proc and passes three values: an `Integer`, a `String`, and a `Boolean`. The call does not wait for a return value. G2 displays these values on the Message Board and returns a `String` to Visual Basic. Visual Basic displays the return value it is received.

Visual Basic

```
Private Sub Form_Load()
   Call G2Gateway1.CallDeferred("my-proc", IDhello, 1,
      "Hello G2", True)
End Sub

Private Sub G2Gateway1_RpcReturned(ByVal Name As String,
      RetArgs As Variant,
      ByVal DeferredCallIdentifier As Variant)
   MsgBox RetArgs
End Sub
```

G2

```
my-proc(arg1: value, arg2: value, arg3: value) = value

begin
    inform the operator that "Value1:[arg1], Value2:[arg2], Value3:[arg3]";
    return "Hello COM";
end
```

Connect()

The `Connect` method establishes a connection to G2 at the network address specified by the `G2Location` property.

Visual Basic Syntax

```
Connect(WaitFlag As Boolean)
```

Microsoft Interface Description Language Syntax

```
HRESULT Connect([in] VARIANT_BOOL WaitFlag)
```

Argument	Description
WaitFlag As Boolean	`Connect(true)` waits for the connection to complete (a *blocking* connection). `Connect(false)` does not wait (a *nonblocking* connection). In either case, G2 ActiveXLink fires a `Connected` event when the connection completes.

Description

You can safely make a `Connect()` call at any time, even if a previous nonblocking connection has not completed. If you make a `Connect()` call to a G2 ActiveXLink object in G2 that is already connected, the call returns with no error.

If you call `Call()`, `Start()`, or `CallDeferred()` and G2 ActiveXLink is not already connected, G2 ActiveXLink automatically creates a blocking connection. Normally, `Connect(true)` does not have to be called by the container application.

Several seconds can elapse before G2 ActiveXLink returns the host name lookup and connects to the G2 server, especially if the network is slow. You can avoid this delay by calling `Connect(false)` when the application starts. You can query the current connection status of G2 ActiveXLink by using the `IsG2Connected` property.

Disconnect()

The `Disconnect` method breaks the connection with G2.

Visual Basic Syntax

```
Disconnect()
```

Microsoft Interface Description Language Syntax

```
HRESULT Disconnect()
```

Description

You can use the `Call()`, `Start()`, and `CallDeferred()` methods to establish transient connections to G2 and use `Disconnect()` to explicitly disconnect these connections when not in use. The next time you invoke the `Call()`, `Start()`, and `CallDeferred()` methods G2 ActiveXLink reestablishes the connection with G2.

PostMessage()

The `PostMessage()` method displays text or values in G2's Message Board.

Visual Basic Syntax

```
PostMessage(Message As Variant)
```

Microsoft Interface Description Language Syntax

```
HRESULT PostMessage([in] Variant Message)
```

Argument	Description
Message As Variant	The data or text that appears in the G2 Message Board. You can send any supported COM automation data type to the G2 Message Board by using `PostMessage()`.

Events

The G2 ActiveXLink can raise the events described in this section. The events concern the connection with G2 and the sending of return values from G2 to the container application.

In your containing application, you can specify logic to respond to these events. You do not need to write handlers for events that are not needed by your container application.

The event handler for a specific event can freely call other methods of G2 ActiveXLink and procedures in G2. The exact sequence of event handling is dependent on the container application.

Refer to Chapter 4, Using G2Gateway for a description of some event handlers.

You can fire events in the container application by using the g2com-call, g2com-start, or g2com-start-over-interface procedures. These procedures raise specific events in the container application, specify the name of the method to invoke, and pass and receive the necessary arguments.

g2com-call

To call a method from G2, use the g2com-call remote procedure. Using g2com-call, the container application can return any number of arguments to G2. Invoke it by using the call procedure action. Calling g2com-call generates an `RpcCalled` event in the container application.

Syntax

g2com-call (*ProcedureName*: text, *InputArguments*: all remaining item-or-value)
across *InterfaceObject*: class g2com-interface
–> *ReturnArguments*: all remaining item-or-value

Arguments	Description
ProcedureName	The name indicates procedure logic to be executed by the container application.
InputArguments	The arguments to the method being called. G2 can pass any number of arguments, separated by commas.
InterfaceObject	The g2com-interface object created by the client when it connects to G2. For more information, see Finding the Interface Object on page 69.

Return Values	Description
ReturnArguments	The arguments to be returned to G2. The container application can return any number of arguments, separated by commas.

Example

```
return = call g2com-call(“LightProcedure”, “red”, “blue”) across interface;
```

g2com-start

To start a method from G2, use the g2com-start remote procedure. Using g2com-start, G2 does not receive any return values. Invoke it using the start procedure action. Calling g2com-start generates an `RpcStarted` event in the container application.

Syntax

g2com-start(*ProcedureName*: text, *InputArguments*: all remaining item-or-value)
across *InterfaceObject*: class g2com-interface

Arguments	Description
ProcedureName	The name indicates procedure logic to be executed by the container application.
InputArguments	The arguments to the method being called. G2 can pass any number of arguments, separated by commas.
InterfaceObject	The g2com-interface object created by the client when it connects to G2. For more information, see Finding the Interface Object on page 69.

Example

```
start g2com-start(“LightProcedure”, “green”, “white”) across interface;
```

g2com-start-over-interface

To start a method from G2 for all interface objects of a given class, use the g2com-start-over-interface KB procedure.

Syntax

g2com-start-over-interface
(*ProcedureName*: text, *InputArguments*: item-or-value, *InterfaceClass*: symbol)

Arguments	Description
ProcedureName	The name indicates procedure logic to be executed by the container application.
InputArguments	The arguments to the method being called. The arguments can be comprised of a single value or a G2 sequence of values.
InterfaceClass	The g2com-interface class or the name of a subclass of g2com-interface. The interface object class is specified by the `InterfaceClass` property of G2 ActiveXLink.

Description

The g2com-start-over-interface procedure does not require that the user find interface objects. By using this single G2 procedure, you can fire an event on all connected clients. Using g2com-start-over-interface eliminates the need to find the g2com-interface object for a given class.

The g2com-start-over-interface procedure calls g2com-start over each g2com-interface object of a specified class. You can use the g2com-start-over-interface procedure if a procedure is started, rather than called.

Finding the Interface Object

G2 creates an interface object for each G2 ActiveXLink client that connects to G2. The g2com-call and g2com-start remote procedure calls require this interface object to fire events in a COM client application.

To use g2com-call and g2com-start, G2 must find the interface object. There are two ways of finding the interface object by using:

- The system procedure g2-current-remote-interface( ) within a G2 procedure that was called or started by a G2 ActiveXLink client.
- The `InterfaceClass` and `RemoteInitializationString` properties of G2 ActiveXLink. These properties specify the interface object and the value of the attribute of the object. You can use these properties to find the interface object even if no G2 procedure is invoked from the G2 ActiveXLink client.

For example, if G2 ActiveXLink has an `InterfaceClass` property of `VBDEMO-INTERFACE` and a `RemoteInitializationString` property of "I love New York," then the following code fragment finds the interface object and calls a procedure across it.

```
if there exists a vbdemo-interface INT such that
    (the remote-process-initialzation-string of INT = "I love New York") then
    Return-Value = call g2com-call("Some procedure Id", "Argument 1", "Argument 2")
    across INT;
end;
```

The following code fragment shows how to invoke a procedure over each connected client. If more than one interface is connected that matches the interface class and remote initialization string, then one of the interfaces is picked. The interface object is found even if no client is connected; thus, you should verify connection status by checking that the gsi-interface-status of the interface object = 2 (ok).

```
INT: class vbdemo-interface;

for INT = each vbdemo-interface do
    if the remote-process-initialization-string of INT = "I love New York" then
        start g2com-start("Some Procedure", "Argument 1", "Argument2") across INT;
end;
```

Note If you use the g2com-start-over-interface KB procedure, you do not need to find the interface object.

RpcCalled()

The `RpcCalled()` event signals that a procedure in G2 has called a procedure in the container application and passed arguments to it. G2 can receive return values.

Visual Basic Syntax

```
RpcCalled(ProcedureName As String,
    InputArguments() As Variant,
    ReturnArguments As Variant)
```

Microsoft Interface Description Language Syntax

```
void RpcCalled([in] BSTR ProcedureName,
    [in] VARIANT *InputArguments,
    [out] VARIANT *ReturnArguments)
```

Arguments	Description
ProcedureName As String	The name indicates procedure logic to be executed by the container application.
InputArguments As Variant	The arguments to the method being called.

Return Values	Description
ReturnArguments As Variant	The arguments to be returned to G2.

Description

The input arguments are passed as a `Variant`. If only one value is passed, the `Variant` holds the value. If more than one value is passed, the `Variant` holds an array of values.

The output arguments can receive values. This may be a single value or an array of values. If an array of values is returned, it is returned as separate return values to G2, unless the first element is made an array, in which case it is sent as a single value that is a sequence.

Note G2 can pass any number of arguments to the container application and the container application can return any number of arguments to G2.

Example

The G2 procedure calls the procedure named g2com-call() to fire the `RpcCalled()` event.

In Visual Basic, the `Form_Load()` function calls `Connect()` to insure that the container application is connected and able to receive events from G2.

The G2 procedure makes two calls to the example Visual Basic program. The first call returns a single value and the second returns an array of three values. These are displayed on the G2 Message Board as a text and a sequence of text values.

G2

```
get-return-values()
G: class g2com-interface;
VAL: value;
VAL1, VAL2, VAL3: value;
SEQ: sequence;

begin
    if there exists a g2com-interface G then begin
        VAL = call g2com-call("Get") across G;
        inform the operator that "Got return value: [VAL]";

        VAL1, VAL2, VAL3 = call g2com-call("Get Many") across G;
        inform the operator that "Got return values: [VAL1] [VAL2] [VAL3]";

        SEQ = call g2com-call("Get Sequence") across G;
        inform the operator that "Got return values: [SEQ]";
    end;
end
```

Visual Basic

```
Private Sub G2Gateway1_RpcCalled(ByVal ProcedureName As String,
       InputArguments As Variant, ReturnArguments As Variant)
   If ProcedureName = "Get" Then
      ReturnArguments = "A Single Value"
   Else
      Dim Ret(2) As Variant
      Ret(0) = "hello"
      Ret(1) = 1
      Ret(2) = True
      If ProcedureName = "Get Many" Then
         ReturnArguments = Ret
      Else
         ReturnArguments = Array(Ret)
      End If
   End If
End Sub

Private Sub Form_Load()
   Call G2Gateway1.Connect(True)
End Sub
```

RpcStarted()

The `RpcStarted()` event signals that a procedure in G2 has started a procedure in the container application and passed arguments to it. G2 does not receive return values.

Visual Basic Syntax

```
RpcStarted(ProcedureName As String, InputArguments()
   As Variant)
```

Microsoft Interface Description Language Syntax

```
void RpcStarted([in] BSTR Name, [in] VARIANT *InputArguments)
```

Arguments	Description
ProcedureName As String	The name of the procedure in the container application being started.
InputArguments As Variant	The arguments to the procedure being started.

Description

The input arguments are passed as a `Variant`. If only one value is passed, the `Variant` holds the value. If more than one value is passed, the `Variant` holds an array of values.

Note G2 can pass any number of arguments to the container application.

Example

The G2 procedure calls the procedure named g2com-start() to fire the `RpcStarted()` event.

G2

```
Start-it()

G: class g2com-interface;

begin
    if there exists a g2com-interface G then begin
        start g2com-start("Hello", 1, 2, 3) across G;
    end
end
```

Visual Basic

```
Private Sub G2Gateway1_RpcStarted(ByVal ProdedureName As String,
      InputArgs As Variant)
   MsgBox InputArgs(1)
   MsgBox InputArgs(2)
   MsgBox InputArgs(3)
End Sub
```

Using the RpcCalled() and RpcStarted() Events

COM uses a publish/subscribe mechanism to distribute events. In some cases, more than one subscriber can receive events. If more than one subscriber receives an event, COM calls each subscriber one after the other and in no particular order. COM passes each subscriber the `InputArguments` and `ReturnArguments` values. After COM calls the last subscriber, it returns the final state of `ReturnArguments` to G2.

Use the `RpcCalled()` event only when you expect exactly one subscriber to receive it. If you expect more than one subscriber to receive this event, you may find the `RpcStarted()` event more appropriate because information can be returned to G2 by using a separate `Start()` call to G2.

However, a typical application using G2 ActiveXLink does not have more than one subscriber to the `RpcCalled()` event and using the `RpcStarted()` event does not apply unless your applet has specific requirements.

G2Connected

The `G2Connected` event signals that G2 ActiveXLink established a connection with G2.

Visual Basic Syntax

```
G2Connected()
```

Microsoft Interface Description Language Syntax

```
void G2Connected()
```

G2Disconnected

The `G2Disconnected` event signals when G2 ActiveX link has lost its connection with G2.

Visual Basic Syntax

```
G2Disconnected()
```

Microsoft Interface Description Language Syntax

```
void G2Disconnected()
```

RpcReturned()

The `RpcReturned()` event returns data from the results of a `CallDeferred()` call. For an example, see CallDeferred() on page 61.

Note G2 can return any number of arguments to the container application.

Visual Basic Syntax

```
RpcReturned(ProcedureName As String,
   ReturnArguments As Variant,
   DeferredCallIdentifier As Variant)
```

Microsoft Interface Description Language Syntax

```
void RpcReturned([in] BSTR ProcedureName,
   [in] VARIANT *ReturnArguments,
   [in] VARIANT DeferredCallIdentifier)
```

Argument	Description
ProcedureName As String	The name of the G2 procedure that was called by `CallDeferred()`.
DeferredCall Identifier As Variant	An identifier matching the identifier specified in the relevant `CallDeferred()` call. G2 ActiveXLink uses *DeferredCallIdentifier* to match specific return values with the call to `CallDeferred()` that produced them. The identifier can be any application-specific value or object that plainly identifies your application.

Return Values	Description
ReturnArguments As Variant	The return arguments from the method that was called. There can be a single return value, an array of return values, or no return values.

G2Paused

The `G2Paused` event signals that G2 has paused.

Visual Basic Syntax

```
G2Paused()
```

Microsoft Interface Description Language Syntax

```
void G2Paused()
```

G2Resumed

The `G2Resumed` event signals that G2 has resumed after it has been paused.

Visual Basic Syntax

```
G2Resumed()
```

Microsoft Interface Description Language Syntax

```
void G2Resumed()
```

G2Reset

The G2Reset event signals that G2 has been reset.

The G2Paused, G2Resumed, G2Reset, and G2Started events all update the G2RunState property appropriately. The G2Disconnected event causes G2RunState to be set back to g2RunStateUnknown.

G2 ActiveX Link maintains its connection to G2 when G2 is reset.

If you require G2 ActiveXLink to disconnect from G2 on reset, call the Disconnect method from the event handler for the G2Reset event. For example, if you have a Visual Basic program with a G2Gateway object named Axl1, the following code causes axl1 to disconnect from G2 on reset:

```
Private Sub Axl1_G2Reset()
   Axl1.Disconnect
End Sub
```

Visual Basic Syntax

```
G2Reset()
```

Microsoft Interface Description Language Syntax

```
void G2Reset()
```

G2Started

The `G2Started` event signals that G2 has been started.

The `G2Paused`, `G2Resumed`, `G2Reset`, and `G2Started` events all update the `G2RunState` property appropriately. The `G2Disconnected` event causes `G2RunState` to be set back to `g2RunStateUnknown`.

Visual Basic Syntax

```
G2Started()
```

Microsoft Interface Description Language Syntax

```
void G2Started()
```

G2RunStateKnown

The `G2RunStateKnown` event signals when G2 reports its run state to G2 ActiveXLink. You should place any initialization that depend on the run state in the handler for this event, instead of in the `G2Connected` handler.

Visual Basic Syntax

```
G2RunStateKnown()
```

Microsoft Interface Description Language Syntax

```
void G2RunStateKnown()
```

AttributeModified()

An attribute of a `G2Item` connected through this `G2Gateway` has been modified.

Visual Basic Syntax

```
AttributeModified (G2Item item, VARIANT attrName, VARIANT newVal,
   long subscriptionHdl, VARIANT userData)
```

Microsoft Interface Description Language Syntax

```
void AttributeModified([in]G2Item **localItem, [in] BSTR
   attributeName, [in] VARIANT newValue,  [in] long
   subscriptionHndl, [in] VARIANT userData) ;
```

Description

For information on subscribing to this event, see Subscribing to Attribute Changes on page 128.

CustomEvent()

A custom event on a `G2Item` connected through this `G2Gateway` has been sent.

Visual Basic Syntax

```
CustomEvent (G2Item item, VARIANT evName, VARIANT newVal,
   long subscriptionHdl, VARIANT userData)
```

Microsoft Interface Description Language Syntax

```
void CustomEvent([in]G2Item **localItem, [in] BSTR EventName,
   [in] VARIANT newValue,  [in] long subscriptionHndl,
   [in] VARIANT userData) ;
```

Description

For information on subscribing to this event, see Subscribing to Custom Events on page 130.

IconColorChanged()

The icon color of a `G2Item` connected through this `G2Gateway` has been modified.

Visual Basic Syntax

```
IconColorChanged (G2Item item, long subscriptionHdl,
   G2Structure chgStruct, VARIANT userData)
```

Microsoft Interface Description Language Syntax

```
void IconColorChanged([in]G2Item **localItem, [in] long
   subscriptionHndl, [in] LPDISPATCH chgStruct,
   [in] VARIANT userData) ;
```

Description

For information on subscribing to this event, see Subscribing to Icon Color Changes on page 129.

When you create an `IconColorChanged` event handler, *`chgStruct`* is a `G2Structure`, where each key-value pair consists of the name of a region of the icon and the name of a color. For example:

```
Private Sub AxL1_IconColorChanged(localItem As GensymAxlCtl.G2Item,
ByVal subscriptionHndl As Long, ByVal
chgStruct As Object, ByVal userData As Variant)
   Dim stru As G2Structure
   Dim n As Integer, i As Integer
   Dim atNames As Variant, atVals As Variant
   Debug.Print "Color of Icon Changed"
   Debug.Print TypeName(chgStruct) ‘ Prints “IG2Structure”
   Set stru = chgStruct
   n = stru.Count
   atNames = stru.Names
   atVals = stru.Values
   For i = 0 To n - 1
   Debug.Print atNames(i) & " : " & stru(i)
   Next i
End Sub
```

This example prints:

```
Color of Icon Changed
IG2Structure
ALARM : RED
```

ItemAdded()

An item has been added to a `G2Workspace` connected through this `G2Gateway`.

Visual Basic Syntax

```
ItemAdded(G2Workspace localWkspc, long subscriptionHdl,
VARIANT userData)
```

Microsoft Interface Description Language Syntax

```
void ItemAdded([in]G2Workspace **localItem,
   [in] long subscriptionHndl, [in] VARIANT userData) ;
```

Description

For information on subscribing to this event, see Subscribing to Workspace Events on page 112.

ItemDeleted()

A `G2Item` connected through this `G2Gateway` has been deleted.

Visual Basic Syntax

```
ItemDeleted (G2Item item, long subscriptionHdl, VARIANT userData)
```

Microsoft Interface Description Language Syntax

```
void ItemDeleted([in]G2Item **localItem, [in] long subscriptionHndl,
   [in] VARIANT userData) ;
```

Description

For information on subscribing to this event, see Subscribing to Item Deletions on page 129.

ItemRemoved()

An item has been removed from a `G2Workspace` connected through this `G2Gateway`.

Visual Basic Syntax

```
ItemRemoved(G2Workspace localWkspc, long subscriptionHdl,
VARIANT userData)
```

Microsoft Interface Description Language Syntax

```
void ItemRemoved([in]G2Workspace **localItem,
   [in] long subscriptionHndl, [in] VARIANT userData) ;
```

Description

For information on subscribing to this event, see Subscribing to Workspace Events on page 112.

ItemSelected()

An item has been selected in a `G2Window` connected through this `G2Gateway`.

Visual Basic Syntax

```
ItemSelected(G2Window localWin, long subscriptionHdl,
VARIANT userData)
```

Microsoft Interface Description Language Syntax

```
void ItemSelected([in]G2Window **localWindow,
   [in] long subscriptionHandle, [in] VARIANT userData);
```

Description

For information on subscribing to this event, see Subscribing to Window Events on page 114.

ValueChanged()

The value of a `G2Item` that is a variable or parameter and that is connected through this `G2Gateway` has changed.

Visual Basic Syntax

```
ValueChanged (G2Item item, VARIANT newVal, long subscriptionHdl,
   VARIANT userData)
```

Microsoft Interface Description Language Syntax

```
void ValueChanged([in]G2Item **localItem, [in] VARIANT newValue,
   [in] long subscriptionHndl, [in] VARIANT userData);
```

Description

For information on subscribing to this event, see Subscribing to Variable and Parameter Value Changes on page 129.

Error

The `Error` event signals that an error has occurred. If an error occurs during a G2 ActiveXLink method call, the error is returned to the application, along with a description of the error.

Visual Basic Syntax

```
Error(ErrorMessage As String,
   ErrorCode As Long,
   DeferredCallIdentifier As Variant)
```

Microsoft Interface Description Language Syntax

```
void Error([in] BSTR ErrorMessage,
   [in] long ErrorCode
   [in] VARIANT DeferredCallIdentifier)
```

Argument	Description
ErrorMessage	The text of the error message.
ErrorCode	A numeric representation of the error that can be used to identify the specific error.
DeferredCall Identifier As Variant	An identifier for errors for nonblocking calls so that they can be associated with the original nonblocking call. The value is empty for errors not associated with a nonblocking call.

Description

Most container applications, including Visual Basic and Visual Basic for Applications, allow an error to be intercepted by the application so that it can handle the error. Most containers display the error with a description and abort the operation. For more information on handling exceptions, see the documentation for the container application.

If G2 ActiveXLink calls a G2 procedure by using `Call()` or `CallDeferred()`, the invoked G2 procedure can indicate an exception with the signal action. The exception is returned to the container application, along with a description.

For example, the following G2 procedure returns an exception:

```
begin
    signal the symbol error, “This description will be returned with the exception”;
end
```

Errors that are not associated with a blocking method call are reported with the `Error` event.

5

Custom Classes

The chapter describes the G2 ActiveXLink custom classes.

Introduction **95**

Using G2Symbol **96**

Using G2Structure **97**

Using G2Item **101**

Using G2List and G2Array **108**

Using G2Workspace **112**

Using G2Window **113**

G2 Type Names **114**

Subscription Types **115**

Introduction

G2 ActiveXLink defines a number of classes that map directly to G2 classes:

- `G2Symbol`
- `G2Structure`
- `G2Item`
- `G2List` and `G2Array`

- G2Workspace
- G2Window

Using G2Symbol

G2 ActiveXLink provides support for passing G2 symbols from G2 to Visual Basic and from VB to G2 by providing the G2Symbol class.

G2Gateway also defines the G2Symbols property, which controls the behavior when sending symbols from G2 to VB. For details, see the description of G2Symbols in Properties on page 52.

To send a symbol from Visual Basic to G2, set a parameter to a G2Symbol. You should always set the name of a property or structure element to a String.

To send symbolic elements such as symbolic attribute values, and symbolic elements of lists and arrays, set the element type to be a G2Symbol. You should always set the element name to be a String.

G2 usually stores symbols as uppercase characters. However, this is not always obvious since G2 sometimes changes the case for display purposes. To force a character in a symbol to lowercase in G2, you precede the character with an at sign (@).

In general, when sending a symbol to G2, you want it to be in uppercase. Thus, the G2Symbol class provides the UpperCase property. When set to True, the text in the symbol is treated as uppercase. When set to False, the true case is shown.

When you receive a G2Symbol from G2, UpperCase is set to False so you can see the true case of the symbols. However, when you create a new G2Symbol as shown in the example below, UpperCase is set to True to conform with what G2 normally expects.

The following commented Visual Basic example shows how to send a G2Symbol to G2:

```
' Create a G2Symbol
' ------------------
Dim symX as New G2Symbol

' Set its text
' ------------
symX = "Example"

' Display its text
' ----------------
Debug.Print symX     ' Displays EXAMPLE

' Send it as a parameter to a G2 procedure
' ---------------------------------------
G2Gateway.Start "EatAVar", symX     'G2 gets the symbol EXAMPLE

' Stop forcing uppercase
' ----------------------
symX.UpperCase = False

' Display its text
' ----------------
Debug.Print symX     ' Displays Example

' Send it as a parameter to a G2 procedure
' ---------------------------------------
G2Gateway.Start "EatAVar", symX     'G2 gets the symbol E@x@a@m@p@l@e
```

Using G2Structure

G2 ActiveXLink provides support for G2 structures, using the COM (Visual Basic) object type, G2Structure. The following section uses Visual Basic to show the capabilities of the G2Structure object type.

Creating a Variable to Represent a G2Structure

There are two ways to create a G2Structure:

- Declare storage for it, create it with New, then fill in the details with the Add method:

```
Dim g2struct As G2Structure

Set g2struct = New G2Structure
g2struct.Add "Line","Conn-Rod"
g2struct.Add "Machine", 5
g2struct.Add "Speed",240.6
```

If at this point you were to use the `Call` or `Start` methods to send `g2struct` to G2, for example, `G2Gateway1.Start("MyG2Procedure",g2struct)`, G2 would interpret it as:

structure(line:"Conn-Rod", machine: 5, speed: 24.6)

- Use `MakeG2ComVariable`, a new method of the `G2Gateway` object.

  The first parameter to this method is `G2StructureType`, a constant that is defined once you add a reference to G2 ActiveXLink to your Visual Basic project. In the future, this method might be expanded so it can create other object types. Currently, `MakeG2ComVariable` can only be used to create instances of `G2Structure`.

To provide flexibility, you can use one of several techniques to pass the structure details to `MakeG2ComVariable`. If you just want to make one short structure object, then it might be easiest just to pass in the names and properties to `MakeG2ComVariable` in alternate positions of the parameter list. On the other hand, if you are creating numerous structure objects, all with the same property names but with different values, then it would be easier to put the property names in an array and then pass the array to `MakeG2ComVariable` in the multiple calls you would make to create multiple structure objects. The values could be passed from a second array or directly from the parameter list.

- Method 1: Names and values from a parameter list.

```
Dim g2Struct As G2Structure

Set g2Struct = G2Gateway1.MakeG2ComVariable(G2StructureType, _
"Line","Conn-Rod","Machine",5,"Speed",240.6)

Call G2Gateway1.Start("MyG2Proc",g2Struct)
```

- Method 2: Names and values from a single array.

```
Dim g2Struct As G2Structure
Dim X

X = Array("Line","Conn-Rod","Machine",5,"Speed",240.6)
Set g2Struct = G2Gateway1.MakeG2ComVariable(G2StructureType,X)
Call G2Gateway1.Start("MyG2Proc",g2Struct)
```

- Method 3: Names from an array, values from a parameter list.

```
Dim g2Struct As G2Structure
Dim X

X = Array("Line","Machine","Speed")
Set g2Struct = G2Gateway1.MakeG2ComVariable(G2StructureType, _
X, "Conn-Rod", 5, 240.6)
Call G2Gateway1.Start("MyG2Proc",g2Struct)
```

- Method 4: Names from an array, values from a second array.

```
Dim g2Struct As G2Structure
Dim X, Y

X = Array("Line","Machine","Speed")
Y = Array("Conn-Rod", 5, 240.6)
Set g2Struct = G2Gateway1.MakeG2ComVariable(G2StructureType,X,Y)
Call G2Gateway1.Start("MyG2Proc",g2Struct)
```

Example: Reading the Value of a Structure Property

The normal way to access the value of a structure's property is to use the property's name as an index, for example:

```
Print g2Struct("Speed")
```

Assuming that `g2Struct` has been initialized with any of the examples shown in the previous section, `240.6` would be printed.

For two less commonly used methods of reading the value of a structure property, see Example: Iterating over Name/Value Pairs on page 100.

Example: Setting the Value of a Structure Property

You use the same method of specifying the property to set as when you read it. The normal technique is to use the property name as an index. For example, the following code changes the value of the property named `Speed` to `305.0`:

```
g2Struct("Speed") =305.0
```

Example: Determining the Number of Name/Value Pairs

A `G2Structure` has a `Count` property that specifies the number of name/value pairs it contains.

Given the examples shown above, adding the following after the initialization of `g2Struct`:

```
Print g2Struct.Count
```

would print `3`, because there are three name value pairs: `("Line"/"Conn-Rod", "Machine"/5, "Speed"/240.6)`.

Example: Obtaining Lists of Property Names or Values

The `Names` method of the `G2Structure` object returns an array containing the names of the contained name/property pairs. For an example, see Example: Iterating over Name/Value Pairs on page 100.

The `Values` method works in a similar manner. It returns an array containing the values of the contained name/value pairs.

Example: Removing a Name/Value Pair from a G2Structure

The `Remove` method of a `G2Structure` removes a name/value pair. You pass the name of the pair to remove, for example:

```
g2Struct.Remove("Machine")
```

Example: Iterating over Name/Value Pairs

A `G2Structure` object is actually a collection of name/value pairs. It has a property named `Count` that specifies how many name/value pairs the structure contains.

There are two ways to iterate over the pairs:

- Iterating with `For Each`

  `G2Structure` support the `For Each` construction, for example:

  ```
  Dim NameVal

  For Each NameVal In g2Struct
     Print NameVal.Name; " : "; NameVal.Value
  Next
  ```

  The results that would be printed are:

  ```
  Line : Conn-Rod
  Machine : 5
  Speed : 240.6
  ```

- Iterating with a numeric index

  Although it is normal to reference the value of a name/value pair by indexing the structure with the name, for example, `g2Struct("Line")`, it is also possible to reference it, using a numeric index. Referencing values by index is generally not advisable, because it means you need to know the position of the name/value pair in the structure. The one time it might be useful is when

you are iterating though the name/value pairs with a numeric index. The following example prints the same results as the previous example:

```
Dim i As Integer
Dim arX As Variant
Dim g2Struct As New G2Structure

g2Struct.Add "Line","Conn-Rod"
g2Struct.Add "Machine", 5
g2Struct.Add "Speed",240.6

arX = g2Struct.Names

For i = 0 to g2Struct.Count-1

Print arX(i); " : "; g2Struct(i)

Next i
```

Using G2Item

G2 ActiveXLink provides the `G2Item` class for representing G2 items in Visual Basic. You use this class to:

- Get and set user-defined attributes of items.
- Read lists or arrays that were sent to ActiveXLink from G2.
- For items of classes that have values, set the value. The value can be an elementary type such as `g2String` or `g2Integer` or a more complex type such as a list or an array.
- Create instances of existing classes in G2.

`G2Item` defines a number of additional methods and events for linked items. For details, see Chapter 6, Item References on page 117.

Specifying the G2 Class

To create an instance of an existing G2 class, you create a `G2Item` and set its `ClassName` property to determine the attributes, their types, and the item value type, if any. You should set the `ClassName` to the name of a known class, which could be a user-defined class or a system-defined class such as integer-parameter.

If you send a `G2Item` without first setting its `ClassName` property, G2 reports that it has received a bad value, which will cause an error in your ActiveXLink program. In Visual Basic, you can use the `On Error` clause to detect and process this error.

If you send a `G2Item` with the `ClassName` property set to an unknown class, G2 reports that the class name does not exist. If you use the `Call` method on `G2Gateway` to send the item, your program receives a timeout error.

If you define a `G2Item` with attributes that do not exist in G2's definition for the class, the extraneous attributes are ignored.

If you send an attribute value to G2 that does not match the type defined in the class definition, G2 reports that it cannot conclude the value into the attribute.

Setting the G2Item Name

You use the `Name` property to specify the name of the item you are creating.

Note G2 ActiveXLink uses G2 Gateway to communicate with G2. Due to a limitation in the current version G2 Gateway, the `Name` attribute is not included in the `GsiItems` that are created by the transmission of an item from G2 to your program. If you require the name, you should send it as a separate parameter.

Determining the Number of User-Defined Attributes

You use the read-only `Count` property to determine the number of user-defined attributes the `G2Item` defines.

Getting and Setting Attribute Values

To get and set values of user-defined attributes, you refer to the index of the attribute. The index can be either a number (0-based) or an attribute name.

For example, suppose you have a G2 class definition named example-class, whose direct-superior-class is object and whose class-specific-attributes are defined as follows:

EXAMPLE CLASS, a class definition	
Class name	example-class
Direct superior classes	object
Class specific attributes	deptno is an integer, initially is 0; manager is a text, initially is ""; rating is a symbol, initially is average; performance is a float, initially is 5.0

Here is the table for an item of type example-class named example-item:

EXAMPLE-ITEM, an example-class	
Notes	OK
Item configuration	none
Names	EXAMPLE-ITEM
Deptno	57
Manager	"Santos"
Rating	superior
Performance	9.7

This is the G2 procedure that G2 ActiveXLink calls to read the item:

READANITEM, a procedure

```
ReadAnItem() = (item-or-value)
begin
  return Example-Item
end
```

Here is Visual Basic code that gets the values of the attributes of example-item. As the example shows, when you use the attribute name as the index, it is not case-sensitive.

```
Dim g2iX As G2Item
   Dim iX   As Integer

   ' Assume that ReadAnItem sends Example-item
   ' ----------------------------------------
   Set g2iX = G2Gateway1.Call("ReadAnItem")

   ' Index by position. Prints:
   '     57
   '     Santos
   '     SUPERIOR
   '     9.7
   ' -------------------------
   For iX = 0 To g2iX.Count-1
       Debug.Print g2iX(iX)
   Next ix

   ' Index by attribute name.  Prints the same results as above.
   ' ----------------------------------------------------------
   Debug.Print g2iX("deptno")
   Debug.Print g2iX("Manager")
   Debug.Print g2iX("RaTiNg")
   Debug.Print g2iX("PERFORMANCE")
```

Getting Attribute Names, Values, and Types

You use the `AttrNames`, `AttrValues`, `AttrTypes` properties to get an array of attribute names, values, and types of a `G2Item`.

In the example above, instead of referring to the attributes by name, you could access them by referring to the `AttrNames` properties, as follows:

```
Dim atNames

   atNames = g2iX.AttrNames
   For iX = 0 to g2iX.Count-1
       Debug.Print g2iX(atNames(iX))
   Next iX
```

The following example uses all three properties to access the attribute names, values, and types:

```
Dim vn, vv, vt
Dim i As Integer

If Not G2iX Is Nothing Then
   vn = G2iX.AttrNames
   vv = G2iX.AttrValues
   vt = G2iX.AttrTypes
   For i = 0 To G2iX.Count - 1
      Debug.Print vn(i)& " " & CStr(vv(i)) & " " & CStr(vt(i))
   Next i
Else
   Debug.Print "G2iX not initialized."
End If
```

This example prints as follows:

```
DEPTNO 57 1
MANAGER Santos 4
RATING SUPERIOR 3
PERFORMANCE 9.7 6
```

Creating G2Items

To create a new item in G2, first, you create a new `G2Item`, then you use the `Add` method to add attributes to the item.

The `Add` method takes two parameters: the attribute name and its value. The attribute's type is set based on the value you provide.

This example creates a new item named example-item of type example-class in G2. The UseAnItem procedure in G2 receives a transient object of type example-class and transfers it to a workspace. Notice that to create a symbolic attribute, first, you create a `G2Symbol` and set its value, then you use it as a parameter to the `Add` method.

```
Dim newEC As New G2Item
Dim symX  As New G2Symbol

newEC.ClassName = "Example-Class"
newEC.Add "DeptNo", 58
newEC.Add "Manager", "Oiltree"
symX = "miserable"
newEC.Add "Rating", symX
NewEC.Add "Performance", 0.3
G2Gateway.Start "UseAnItem", newEC
```

Creating Symbolic Attributes

Rather than creating a G2Symbol and using it as a parameter to the Add method, you can use the Symbolize command to create symbolic attributes, as follows:

```
G2ItemName.Symbolize index as Variant, convert-to-symbol
as Boolean
```

As with indexed access, *index* can be either a number or a string containing the name of an attribute.

Set *convert-to-symbol* to True to convert a text attribute to a symbol. Set it to False to convert a symbol to text. If the attribute is neither a text nor symbol, the argument is ignored.

Unlike with G2Symbol, Symbolize does not automatically convert the value to upper case.

For example, suppose you create an attribute as follows:

```
g2iX("Msg") = "Hello""

g2iX.Symbolize "Msg", True
```

When the attribute is sent to G2, its value would be H@e@l@l@o. Thus, when using Symbolize, you are responsible for entering the text in uppercase.

To rewrite the previous example, using Symbolize instead of G2Symbol:

```
Dim newEC As New G2Item
Dim symX  As New G2Symbol

newEC.ClassName = "Example-Class"
newEC.Add "DeptNo", 58
newEC.Add "Manager", "Oiltree"
newEC.Add "Rating", "MISERABLE"     '  Uses uppercase
newEC.Symbolize("Rating")
NewEC.Add "Performance", 0.3
G2Gateway.Start "UseAnItem", newEC
```

Removing Attributes

If, for some reason, you want to remove an attribute from a G2Item, you can use the Remove method.

```
G2ItemName.Remove (index as Variant)
```

As with indexed access, *index* can be either a number or a string containing the name of an attribute.

Iterating Over the Attributes of a G2Item

`G2Item` supports Visual Basic's special `For Each x In G2Item` syntax, which enables iteration over the attributes of a `G2Item` without specifying an index.

The element stored in `x` is an object of type `G2Attribute`, which is a class defined for use in this one special case. This class supports the following properties:

- `Name` — The name of the attribute, which is read-only.
- `Value` — The value of the attribute.
- `Type` — The `g2Types` enumeration code of the attribute. In most cases, it is read-only and depends upon the value type. However, you can set it to `g2String` or `g2SymbolType` if the value contains text. Converting from type `g2String` to `g2SymbolType` causes the text to be converted to uppercase.

This example shows how to iterate over the attributes of the G2 item named example-item. Note that when you set an attribute of type `G2SymbolType` (3) to a text string, the attribute type is not changed; it continues to be a symbol and the text is converted to uppercase. If you want a symbol with lowercase characters, you must set the attribute value to a `G2SymbolType` with `UpperCase` set to `False`.

```
Dim g2iX As G2Item
Dim x

' Assume that ReadAnItem sends Example-item
' ----------------------------------------
Set g2iX = G2Gateway1.Call("ReadAnItem")

' Index by position. Prints:
'     DEPTNO 57 1
'     MANAGER Santos 4
'     RATING SUPERIOR 3
'     PERFORMANCE 9.7 6
' --------------------------

For Each x in g2iX
   Debug.Print x.Name & " " & CStr(x.Value) & " " & CStr(x.Type)
      If x.Type = 1 Then
         x.Value = 59
      ElseIf x.Type = 4 Then
         x.Value = "Martin"
      ElseIf x.Type = 3 Then
         x.Value = "average"
      Else
         x.Value = 5.8
      End If
Next
```

```
'prints
      'DEPTNO 59 1
      'MANAGER Martin 4
      'RATING AVERAGE 3
      'PERFORMANCE 5.8 6
' ---------------------
For Each x in g2iX
   Debug.Print x.Name & " " & CStr(x.Value) & " " & CStr(x.Type)
Next
```

Using G2Item Value and Type

Commonly, G2 items do not have an associated value. For example, example-class, the class used in the examples above, has a direct superior class of object. Because the object class does not have an associated `Value` property, neither does example-class.

On the other hand, classes derived from parameter or variables do have a value of a type that matches the superior class. A class derived from integer-parameter or integer-variable has a value of type `g2Integer` (1), a class derived from symbolic-variable or symbolic-parameter has a value of type `g2SymbolType` (3), and so on.

However, currently, there is the same restriction reading the value of an item as there is with reading its name. With the exception of lists and arrays, G2 Gateway does not reliably transmit values from G2 to G2 ActiveXLink. Furthermore, if you try to read the value of an item that does not have a value or the value of unsupported data types (value, handle, quantity, short vector), an error occurs in your program. You should make sure to add error processing code if you are dealing with item values.

You use the `Value` and `Type` properties primarily for setting the value of an item that you want to send to G2 and for reviewing the resulting type. You can also use the `Type` property to change a string type to symbol or vice versa.

Using G2List and G2Array

G2 lists and arrays are both items. As a result, when G2 receives a G2 array or list, G2 ActiveXLink returns is as a `G2Item`.

To access the element values of the list or array, read the `Value` property of the `G2Item`. G2 ActiveXLink returns an object of type `G2ListOrArray`.

Determining the Number of Elements

You use the read-only `Count` property to determine the number of elements in the `G2ListOrArray`.

Getting and Setting Element Values

To refer to a specific element of a list or array, follow the name of the variable holding the `G2ListOrArray` with a numeric index contained within parentheses. The index should be a number between 0 and `Count-1`. You use indexed access to both get and set elements of a `G2ListOrArray`.

For example, suppose a call to the procedure `GetArray` returns an integer array containing the value 7,5,3,1, and -1. The following code sets the values of `iar` to -1, 7, 5, 3, and 1.

```
Dim iar As G2ListOrArray
Dim i As Integer
Dim Hold As Integer

Set iar = G2Gateway1.Call("GetArray")
If iar.Count > 0 Then Hold = iar(iar.Count-1)
For i = iar.Count-2 To 0 Step -1
   iar(i+1) = iar(i)
Next i
If iar.Count > 0 Then iar(0) = Hold
```

Determining the Type

You use the `Type` property to determine whether a `G2ListOrArray` contains a list or array, and the type of its elements.

You can also use it to set the list or array, and the element type when you create a new `G2ListOrArray` that you want to send to G2.

Caution Setting the `Type` of a `G2ListOrArray` causes all data that was previously stored to be lost.

Inserting, Appending, and Adding Elements to the List or Array

You use the following methods on `G2ListOrArray`:

`G2ListOrArray``.Insert` *`index`*, *`value`*

Inserts *`value`* before the element at position *`index`*.

G2ListOrArray.Append *index*, *value*

Inserts *value* into the list or array after the element at position *index*.

G2ListOrArray.Add value

Inserts *value* at the end of the list or array.

Removing Elements from a List or Array

You use the following method to remove elements from local copies of a list or array:

G2ListOrArray.Remove *index*

Removes the element at position *index* from the list or array.

Iterating Over Elements of a List or Array

G2 ActiveXLink supports the `For Each x In` syntax for iterating over elements of a `G2ListOrArray`. The value stored in `x` is an item of the `G2LAElement` class, which is a class defined for use in this one special case.

`G2LAElement` supports the `Value` property for getting and setting the value of an element. It also supports getting the value of an element directly through the variable, without specifying a property, but not setting it.

To avoid the ambiguity that would occur in the following situation, setting the value directly through the variable is not allowed. This example repeatedly sets the variable `x` to `4`, which is probably not what you want.

```
Dim x As Variant

For Each x In MyG2ListOrArray
   x = 4
Next
```

To set each element to 4, you must use this code:

```
Dim x As Variant

For Each x In MyG2ListOrArray
   x.Value = 4
Next
```

Sending Lists and Arrays to G2

Because G2 lists and arrays are objects, you can create subclasses. Thus, you can have an item that is a G2 list or array that also has attributes. As a result, you use one of two techniques to send lists and arrays to G2, depending on whether it is a simple list or array, or a subclass.

To send a simple list or array to G2:

1. Create a new `G2ListOrArray`.
2. Set its `Type`.
3. Fill in the elements by using the `Add` method of the `G2Item`.
4. Send it to G2.

For example:

```
Dim g2ValList as New G2ListOrArray

g2ValList.Type = g2ValueList
g2ValList.Add 2
g2ValList.Add "Oi"
g2ValList.Add 29.8

G2Gateway1.Start "Consume", g2ValList
```

To send a subclass of a G2 list or array to G2:

1. Follow steps 1 - 3 above.
2. Create a new `G2Item`.
3. Set the class name of the `G2Item` to the name of a class defined in G2 as a subclass of a G2 list or array.
4. Set the `Value` property of the `G2Item` to the `G2ListOrArray` you created in step 1.
5. Optionally, specify the `G2Item` name.
6. Set any desired attributes by using the `Add` method of the `G2Item`.
7. Send the `G2Item` to G2.

For example:

```
Dim itemX As New G2Item
Dim arrayX as New G2ListOrArray

arrayX.Type = g2SymbolArray
arrayX.Add "unacceptable"
arrayX.Add "poor"
arrayX.Add "fair"
arrayX.Add "good"
arrayX.Add "excellent"

itemX.ClassName = "symarsub"
itemX.Name = "Grades"
set itemX.Value = arrayX
```

```
itemX.Add "Language","English"
itemX.Add "Group",1
G2Gateway1.Start "Consume",itemX
```

Using G2Workspace

A G2Workspace is a specialized form of a G2Item. You have access to all properties, events, and methods on G2Item. However, to access the properties and methods of G2Item, you must cast the G2Workspace to a G2Item.

For example:

```
Dim g2Wkspc As New G2Workspace
Dim g2Itm As G2Item

Set g2Itm = g2Wkspc
g2Itm.Name = "Made-by-AxL"
g2Itm.Create (G2Gateway1)
```

Subscribing to Workspace Events

G2Workspace provides event notification for these G2Gateway events:

- ItemAdded(G2Workspace *localWkspc*, long *subscriptionHdl*, VARIANT *userData*)
- ItemRemoved(G2Workspace *localWkspc*, long *subscriptionHdl*, VARIANT *userData*)

Subscribing to Workspace Additions

This method requests notification by the G2Workspace via the ItemAdded event when an item is added to a workspace in G2.

```
SubscribeToItemAddition(VARIANT userData) As VARIANT
```

The method returns a subscription handle or an error message.

Subscribing to Workspace Removals

This method requests notification by the G2Workspace via the ItemRemoved event when an item is removed to a workspace in G2.

```
SubscribeToItemRemoval(VARIANT userData) As VARIANT
```

The method returns a subscription handle or an error message.

Unsubscribing from Workspace Additions

This method cancels requests for notification when an item is added to a workspace:

```
UnsubscribeFromItemAddition( )
```

Unsubscribing from Workspace Removals

This method cancels requests for notification when an item is removed from a workspace:

```
UnsubscribeFromItemRemoval( )
```

Using G2Window

A `G2Window` is a specialized form of a `G2Item` with one custom property, `g2UserMode`. You have access to all properties, events, and methods on `G2Item`. However, to access the properties and methods of `G2Item`, you must cast the `G2Window` to a `G2Item`.

For example:

```
Dim g2Win As G2Window
Dim g2Itm As G2Item

' read g2Win from G2
' ------------------
Set g2Win = G2Gateway.Call("SendAWindow")
Set g2Itm = g2Win
g2Itm.Name = "Made-by-AxL"
g2Itm.Update
```

Getting the G2 User Mode of a Window

`g2UserMode` contains a string specifying the g2-user-mode of the associated g2-window in G2.

To change the user mode of the g2-window, set the `g2UserMode` property, then execute the `G2Item.Update()` method.

For example:

```
g2Win.g2UserMode = "developer"
Set g2Itm = g2Win
g2Itm.Update
```

Subscribing to Window Events

G2Window provides event notification for this G2Gateway event:

```
ItemSelected(G2Window localWin, long subscriptionHdl,
VARIANT userData)
```

Subscribing to Selection Events

This method requests notification by the G2Window via the ItemSelected event when an item is selected in the window:

```
SubscribeToSelection(VARIANT userData) As VARIANT
```

The method returns a subscription handle.

Unsubscribing from Selection Events

This method cancels requests for notification when an item is selected in a window:

```
UnsubscribeFromSelection( )
```

G2 Type Names

The following table defines the COM types for each of the G2 types:

G2 Type	COM Type
G2Item	IG2Item4
G2Workspace	IG2Workspace
G2Window	IG2Window

You can use these names with Visual Basic's TypeName function to determine the type of item you received from G2. Note that with each new version of G2 ActiveXLink, the strings used to identify the COM types will change. For example, in the next version, the default interface will be G2Item is IG2Item5.

Note To avoid having to change your code each time the default interface changes, we recommend that you compare the first 7 characters with IG2Item, which will remain consistent from release to release.

Subscription Types

This table summarizes the Visual Basic codes for subscription types:

Code	Subscription Type
1	Modify attribute
2	Delete item
3	Icon color change
4	Custom event
5	Value change
6	Add item to workspace
7	Remove item from workspace
8	Item selection

6

Item References

The chapter describes the G2 ActiveXLink data types.

Introduction **117**

Creating and Linking a G2Item **118**

Getting the Icon for a G2Item **121**

Deleting a G2Item **122**

Updating the Item in G2 **122**

Refreshing a G2Item **124**

Verifying Linked Items **124**

Unlinking a G2Item **124**

Getting G2Item Attribute Names, Values, and Types **125**

Using Linked Items as Parameters to RPCs **126**

Subscribing to Item Events **127**

Introduction

If your COM program uses a `G2Gateway` to call a procedure in G2 and G2 returns an item, G2 ActiveXLink creates a `G2Item` that automatically contains an internal reference to the item in G2. One part of the reference tracks the `G2Gateway` through which this item was received. As will be explained later, there are cases where you may want to send a reference to G2 as a parameter to a `Call`, `Start`, or

`CallDeferred` method. In these cases, the transmission must occur across the same `G2Gateway` through which it was originally received.

You can use these methods to verify and remove references:

- The `Linked` method verifies that the reference exists. This method returns `True` when the item contains a reference.
- The `Unlink` method removes the reference from a `G2Item`.

Caution A reference consists of a UUID and information about the `G2Gateway` over which the item was transmitted. This reference may change in a future version of G2 ActiveXLink.

G2 normally tags each item with a unique identifier known as its UUID. However, as the *G2 Reference Manual* explains, there are things you can do in G2 that will set the UUID of more than one item to the same value.

If more than one item in G2 has the same UUID, the new methods of `G2Item` may fail or produce unexpected results. You should ensure that each G2 item has a unique UUID. Furthermore, if your G2 ActiveXLink program has a linked `G2Item`, you should not perform actions in G2 that would interfere with the linkage, such as loading a different knowledge base.

Creating and Linking a G2Item

This method creates an item in G2 and links it to the local `G2Item`. The local item must be unlinked when you call this method.

```
G2Item.Create(G2Gateway)
```

You can specify a G2 procedure customize the behavior of the `Create` method. To do this:

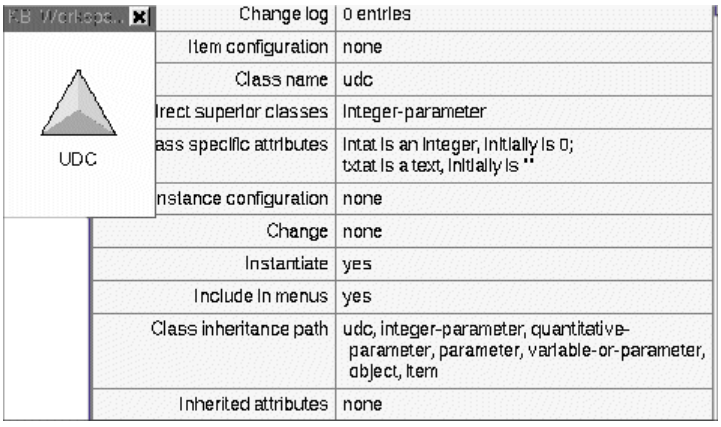
1. Define in G2 a class that is a subclass of g2com-interface with a symbolic attribute named create-handler, and set its initial value to the name of the procedure to call.
2. In G2, write the procedure to process the item.
3. In your COM program, configure the `G2Gateway` to use the new class as the interface class.

When you use the `Create` method to create a new item, the specified procedure is called and the new item is passed as its only parameter.

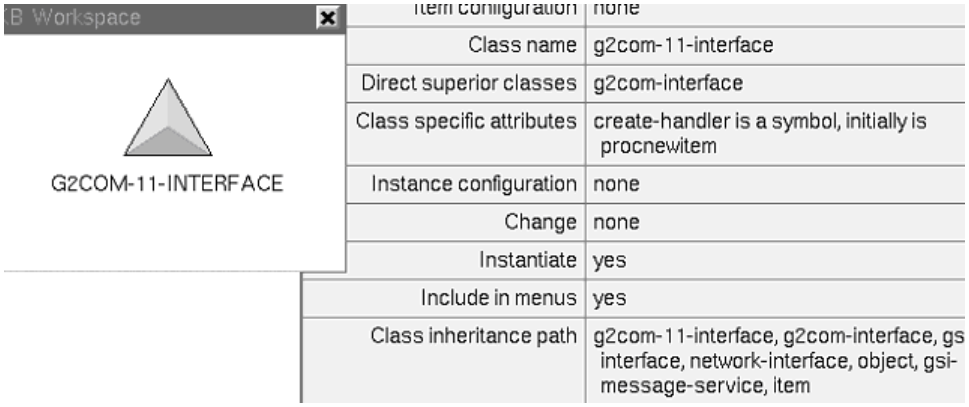
The following example creates a new `G2Item`. Because the example is creating a new item, rather than reading the item from G2, you are assured that it is initially unlinked.

```
Dim g2it As New G2Item

' Create a new G2Item
' -------------------
g2it.ClassName = "UDC"
g2it.Name = "VBC01"
g2it.Add("TxtAt","Created with ActiveXLink")
g2it.Add("IntAt", 22)

' Create an item in the G2 referenced by the
' G2Gateway with the name AXL1.
' ------------------------------------------
On Error GoTo CreateFailed
g2it.Create(AXL1)

' If we get here, we succeeded
' ----------------------------
Debug.Print "Success"

' Bypass the next section of code in whatever manner
' is appropriate (Exit Sub, Exit Function, GoTo, etc.)
' ----------------------------------------------------
      :    :    :

' Report Error
' ------------
CreateFailed:
   Debug.Print "Create Failed. Error Code=" & _
     CStr(Err.Number) & " : " & Err.Description
```

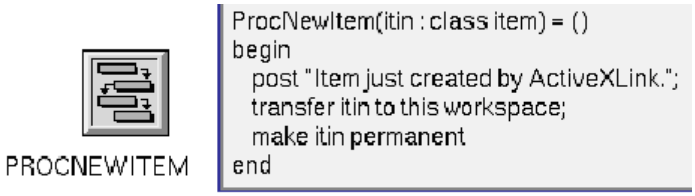
The G2 referenced by `AXL1` should contain a definition for a class named UDC, which should define attributes TxtAt and IntAt, as follows:



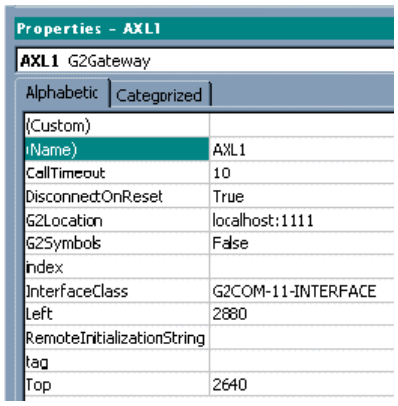
Once the new item is created, G2 checks if the g2com-interface object has an attribute named create-handler. If it does, it calls the procedure specified by that attribute. Here is the subclass of g2com-interface that defines the create-handler procedure, whose initial value is procnewitem:



Here is procnewitem procedure that gets called when the item is created in your COM program:



In the example COM program, `AXL1` is specified as the `G2Gateway` that will be used as the bridge to G2. The `InterfaceClass` attribute of `AXL1` is set to `G2COM-11-INTERFACE`.



Getting the Icon for a G2Item

If a G2Item is linked to an item in G2, the Icon method returns a Picture object that contains a picture of the icon that G2 uses for the linked item. If the item does not have an icon, calling this method throws an exception.

```
G2Item.Icon(Long backgroundColor)
```

Note that each call to the Icon method returns a new Picture object, downloading a new bitmap from the G2 server. Since a bitmap can be quite large, this can result in a perceptible delay. If you need to work with the same icon more than once in the same program, you can improve performance by only calling the Icon method once, then working with the local image. In many cases, the ImageList control is ideal for storage of the local image.

You must provide a Visual Basic-style background color to the Icon method. If the icon is not rectangular, any areas that that are not defined for the icon in G2 are displayed in the background color that you specify.

A Visual Basic-style color is a Long that is either a code for a type of data displayed for Windows, for example, button face or active title bar, or a combination of three numbers with values between 0 and 255 that represent the intensity of the red, green, and blue components of the color.

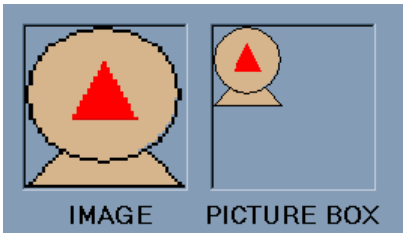
The easiest way to set the color is to set it to the BackColor property of a control that uses the background color you want to apply to the icon. Visual Basic provides numerous constants that you can use for the background color, including: vbBlack, vbRed, vbGreen, vbYellow, vbBlue, vbMagenta, vbCyan, vbWhite, as well as system colors such as vbDesktop and vbButtonFace. Search for "Color Constants" in Visual Basic help to get a complete list of the symbolic constants that you can use.

The size of the Picture object depends on the size of the icon in G2. You can force the display of the Picture to be a specific size by storing it in an Image Control of the desired size and setting the Stretch attribute of the Image Control to True.

This example shows how to use an Image control, Image1, and a PictureBox control, Picture1 to display a G2 icon in a program that uses G2 ActiveXLink. The Stretch property of the Image control is set to True.

If you have a G2Item named g2itX that is linked to an item with an icon in G2, then the code below displays the icon in both Image1 and Picture1.

```
Dim px As Picture
Set px = g2itX.Icon(Picture1.BackColor)
Set Image1.Picture = px
Set Picture1.Picture = px
```



The icon always fills the `Image` control. However, the size of the icon in the `PictureBox` control depends on its size in G2. You can see this by using the Change Size command on the icon in G2 and then re-executing the above code.

Deleting a G2Item

This method deletes in G2 the item referenced by a `G2Item`, then it removes the reference from the `G2Item`:

```
G2Item.Delete()
```

After running the example shown for the `Create` method, if your program executes the following command, the item named VBC01 that was created in G2 by the `Create` method would be deleted:

```
g2it.Delete
```

Afterward, the following method would print `False`:

```
Debug.Print g2it.Linked
```

Updating the Item in G2

This method refreshes the referenced item in G2 with the current state of the `G2Item` in your COM program:

```
G2Item.Update()
```

After the example for the `Create` method, the table for the new item would appear as follows:

VBC01, an udc	
Options	do not forward chain
Notes	OK
Item configuration	none
Names	VBC01
Tracing and breakpoints	default
Data type	integer
Initial value	0
Last recorded value	0
History keeping spec	do not keep history
Intat	22
Txtat	"Created with ActiveXLink"

Now, suppose you execute the following code:

```
g2it("TxtAt") = "Update Example"
g2it("IntAt") = 33
g2it.Value = 15
g2it.Update
```

As soon as you execute the `g2it.Update` command, the table changes to:

VBC01, an udc	
Options	do not forward chain
Notes	OK
Item configuration	none
Names	VBC01
Tracing and breakpoints	default
Data type	integer
Initial value	0
Last recorded value	15
History keeping spec	do not keep history
Intat	33
Txtat	"Update Example"

Refreshing a G2Item

This method modifies a G2Item to match the data in the referenced item in G2:

```
G2Item.Refresh()
```

Suppose the following call returns an integer-parameter with value 101:

```
Set g2it = AXL1.Call("EmitItem")
```

The value of g2it.Value would be 101.

Now, suppose the value of the item in G2 changes to 112. In the following segment of code, the first print statement would print 101, and the second print statement would print 112:

```
Debug.Print g2it.Value          ' Prints 101
g2it.Refresh
Debug.Print g2it.Value          ' Prints 112
```

This behavior applies to attributes and the item name, and to their values.

Verifying Linked Items

This method returns TRUE or FALSE indicating whether the G2Item references an item in G2:

```
G2Item.Linked()
```

This method is important because some methods such as Delete only work with linked items, whereas others such as Create only work with a G2Item that does not contain a remote reference.

Unlinking a G2Item

This method removes the reference to an item in G2 from a G2Item:

```
G2Item.Unlink()
```

For example, you would use this method when creating a new G2Item from an existing item in G2.

This example unlinks an item before creating a new item:

```
Set g2it = AXL1.Call("EmitItem")
Debug.Print g2it.Linked           ' Prints True
g2it.Unlink
Debug.Print g2it.Linked           ' Prints False
g2it.Name = "VBC02"
g2it.Create(AXL1)                 ' Creates item in G2 named VBC02
Debug.Print g2it.Linked           ' Prints True
```

Getting G2Item Attribute Names, Values, and Types

You use the `AttrNames`, `AttrValues`, `AttrTypes` properties to get an array of attribute names, values, and types of a `G2Item`. You can also use the `AttrType` property to get the type of a particular attribute.

You can access attributes by referring to the `AttrNames` properties, as follows:

```
Dim atNames

   atNames = g2iX.AttrNames
   For iX = 0 to g2iX.Count-1
       Debug.Print g2iX(atNames(iX))
   Next iX
```

The following example uses all four properties to access the attribute names, values, and types:

```
Dim vn, vv, vt
Dim i As Integer

If Not G2iX Is Nothing Then
   vn = G2iX.AttrNames
   vv = G2iX.AttrValues
   vt = G2iX.AttrTypes
   For i = 0 To G2iX.Count - 1
      Debug.Print vn(i)& " " & CStr(vv(i)) & " " & CStr(vt(i)))
   Next i
Else
   Debug.Print "G2iX not initialized."
End If
```

This example prints as follows:

```
DEPTNO 57 1
MANAGER Santos 4
RATING SUPERIOR 3
PERFORMANCE 9.7 6
```

Alternatively, you can access attribute types by using `AttrType` as follows:

```
Debug.Print vn(i) & " " & CStr(vv(i)) & " " & G2iX.AttrType(i)
```

or

```
Debug.Print vn(i) & " " & CStr(vv(i)) & " " & G2iX.AttrType(vn(i))
```

Using Linked Items as Parameters to RPCs

If you send a linked `G2Item` to G2 by passing it, for example, as a parameter to `Call` or by returning it to a g2com-call, you create a new item that is distinct from the item to which the `G2Item` is linked.

To use the linked item to which a `G2Item` refers, you must:

- Pass `G2Item.Reference` to G2.
- Define the corresponding parameter in the G2 procedure to be of a compatible class type. It will not work if you define the parameter to be of type item-or-value.

For example, suppose the following procedure is defined in G2:

```
send-window()=()
    g2cX : class g2com-interface;
    winX : class g2-window ;
begin
    if there exists a g2com-interface g2cX such that
    (the remote-process-initialization-string of g2cX = "DemoProg")
    and there exists a g2-window winX such that
    (the g2-window-remote-host-name of winX = "BELEM") then
        start g2com-start("ConsumeWindow", winX, audc) across g2cX
end
```

If you start this procedure and it finds there is a Telewindows connection from a machine named belem and there is a G2 ActiveXLink connection that uses a `RemoteInitializationString` set to `DemoProgram`, then the procedure uses g2com-start to send the g2-window and an item named audc to your COM application.

Further, suppose that the `RpcStarted` event processor in your program includes the code:

```
Dim winX As G2Window
Dim gitX as G2Item

If ProcedureName = "ConsumeWindow" then
   Set winX = InputArguments(1)
   Set gitX = InputArguments(2)
EndIf
```

You can use the system procedure g2-ui-select to select a specific item in a specific window. You might think that you could directly use the `Start` method of a `G2Gateway` to select `audc` in the window `winX` as follows:

```
Dim winAsGit As G2Item

' Note: this code segment will NOT work
' -----------------------------------

Set winAsGit = winX
G2Gateway.Start("g2-ui-select", winAsGit.Reference, gitX.
Reference)
```

However, this code does not work because both of the parameters to g2-ui-select are defined as item-or-value. The received values are treated as text, not the items to which you want to refer.

The solution is to add an intermediate procedure to G2, as follows:

```
procedure xlator(theItem:class item, theWindow:class g2-window)
begin
    call g2-ui-select(theItem, theWindow)
end
```

With this procedure defined, you would then replace the above Visual Basic code with this code:

```
Dim winAsGit As G2Item
' Note: this code segment WILL work
' --------------------------------

Set winAsGit = winX
G2Gateway.Start("Xlator", winAsGit.Reference, gitX.Reference)
```

This code works because the parameters to xlator are defined to be of types of specific classes. `G2Item.References` passed to these parameters are interpreted as the items to which the references refer.

Subscribing to Item Events

You can use various subscription methods to request notification when various events occur on an item. You use various unsubscribe methods to cancel one or more of your notification requests. You use the `Subscriptions` method to return a list of active notification requests.

`G2Item` provides event notification for these `G2Gateway` events:

- `AttributeModified (`*`item`*`, `*`attrName`*`, `*`newVal`*`, `*`hndl`*`, `*`userData`*`)`
- `ItemDeleted (`*`item`*`, `*`hndl`*`, `*`userData`*`)`
- `IconColorChanged (`*`item`*`, `*`hndl`*`, `*`chgStruct`*`, `*`userData`*`)`

- CustomEvent (*item*, *evName*, *newVal*, *hndl*, *userData*)
- ValueChanged (*item*, *newVal*, *Hndl*, *userData*)

To provide maximum flexibility, notification of changes to the value of an attribute in G2 does not automatically update the value in the G2Item.

Subscribing to Attribute Changes

This method requests notification by the G2Item via the AttributeModified event when a specified attribute or attributes of the linked item are modified or when the item is deleted:

```
G2Item.SubscribeToAttributeModification(VARIANT ToWhat,
   VARIANT UserData) As VARIANT
```

The *ToWhat* parameter can be either a String (BSTR in COM terminology) or an array of Strings, where each String can be:

- The name of an attribute to receive notification when the value of the specified attribute changes in G2.
- "All" to receive notification when any attribute value of the item changes in G2.

The *UserData* parameter is any user-defined data that you want to attach to the subscription request. When the event is triggered, *UserData* is returned to the event procedure. You can use this data to help distinguish the event that was triggered, which can greatly simplify event processing, depending upon your application.

The return value from this method is either a subscription handle or an array of subscription handles. A subscription handle is an integer assigned by G2 to the notification request. You can use this handle to identify the request that triggered an event. You can also use it to cancel the request for notification.

If the *ToWhat* parameter is a String, the value returned from this method is a single subscription handle. If *ToWhat* is an array of strings, the return value is an array of subscription handles, one for each subscription request.

Caution You should avoid subscribing to the same event more than once. Each time you subscribe to an event, G2 assigns a new subscription handle to that event. If you have subscribed multiple times to the same event, the event procedure will be called multiple times when that event occurs.

Subscribing to Item Deletions

This method requests notification by the `G2Item` via the `ItemDeleted` event when the item is deleted:

```
G2Item.SubscribeToDeletion(VARIANT UserData) As VARIANT
```

The method returns a subscription handle or an error message. When the item is deleted, the `G2Item` is also unlinked.

Note Deleting an item causes its attributes to be changed before the deletion is completed. Thus, subscribing to both attribute modifications and item deletion causes the `G2ItemModified` event to be triggered for each attribute to which you have subscribed before the `G2ItemDeleted` event is triggered. In addition, subscribing to `"all"` causes the `G2ItemModified` event to be triggered for several system attributes such as item-active when the item is deleted.

See Subscribing to Attribute Changes on page 128 for a description of the *UserData* argument, the return value, and a caution.

Subscribing to Icon Color Changes

This method requests notification by the `G2Item` via the `IconColorChange` event when any region of the icon associated with the item changes color:

```
G2Item.SubscribeToIconColorChange(VARIANT UserData) As VARIANT
```

The method returns a subscription handle or an error message.

See Subscribing to Attribute Changes on page 128 for a description of the *UserData* argument, the return value, and a caution.

Subscribing to Variable and Parameter Value Changes

This method requests notification by the `G2Item` via the `ValueChanged` event when the value of a variable or parameter changes:

```
G2Item.SubscribeToValueChange(VARIANT UserData) As VARIANT
```

The method returns a subscription handle or an error message.

The event is triggered only when the value changes from its previous value, or when the value of a variable expires and the variable receives a new value. If the item is not a variable or parameter, an exception is thrown.

See Subscribing to Attribute Changes on page 128 for a description of the *UserData* argument, the return value, and a caution.

Subscribing to Custom Events

This method requests notification by the `G2Item` via the `CustomEvent` event when a custom event occurs on the item:

```
G2Item.SubscribeToCustomEvent(STRING EventName, VARIANT ToWhat,
   VARIANT UserData) As VARIANT
```

The method returns a subscription handle or an error message.

To trigger the custom event, call g2-send-notification-for-item with the specified event name for the item.

See Subscribing to Attribute Changes on page 128 for a description of the *UserData* argument, the return value, and a caution.

Unsubscribing from Attribute Changes

This method cancels requests for notification when attribute values change:

```
G2Item.UnsubscribeFromAttributeModification(VARIANT FromWhat)
```

The *FromWhat* parameter can be:

- The name of an attribute to which you have subscribed to cancel modification notification requests for that attribute.
- A subscription handle to cancel a specific notification request.
- An array containing any combination of the above types.

Unsubscribing from Item Deletions

This method cancels requests for notification when an item is deleted:

```
G2Item.UnsubscribeFromDeletion( )
```

Unsubscribing from Icon Color Changes

This method cancels requests for notification when any region of the icon associated with an item changes:

```
G2Item.UnsubscribeFromIconColorChange( )
```

Unsubscribing from Custom Events

This method cancels requests for notification when a custom event occurs on an item:

```
G2Item.UnsubscribeFromCustomEvent(VARIANT EventNames)
```

The *EventNames* parameter can be the name of a custom event or an array containing the names of several events.

Unsubscribing from Variable and Parameter Value Changes

This method cancels requests for notification when a variable or parameter value changes:

```
G2Item.UnsubscribeFromValueChange( )
```

Unsubscribing from All Event Notification

This method cancels requests for notification of all events associated with an item, including G2Workspace and G2Window items:

```
G2Item.UnsubscribeFromAll( )
```

Note UnsubscribeFromAll is a method on a particular G2Item, which unsubscribes from all subscriptions created by calls to methods on that G2Item. It does not unsubscribe from events on other G2Item instances, or from events that were requested by some other means.

Note Calling g2-deregister-subscription in G2 cancels the active G2 ActiveXLink subscription to the event in that G2; however, G2 ActiveXLink is not notified of the cancellation. If you then try to unsubscribe in G2 ActiveXLink, G2 reports an error and G2 ActiveXLink throws an exception. You should not subscribe to events in G2 ActiveXLink and unsubscribe from the event in G2. For more information, see the description of g2-deregister-subscription in Chapter 26 "Publish/Subscribe Operations" in the *G2 System Procedures Reference Manual*.

Getting Information about Subscriptions

This method returns a three-column array containing information about the active requests for notification:

```
G2Item.Subscriptions( )
```

The first column contains subscription handle values, and the second column contains codes indicating the type of subscription (1 for modify, 2 for delete, etc.). For subscriptions of type 1, the third column contains the name of the attribute being monitored or "all"; for subscriptions of type 4 (custom event), the third column contains the name of the custom event; otherwise, for subscriptions of type 2, the third column contains "".

The following Visual Basic code segment prints out, among other things, information about which subscriptions are active for `g2it`, a `G2Item`:

```
Dim i As Integer, n As Integer
   Dim atNames As Variant, vX as Variant

  If g2it Is Nothing Then
    Debug.Print "Item has not been read or created"
  Else

    ' Display the available system attributes
    ' -------------------------------------
    Debug.Print "Class Name: " & g2it.ClassName
    Debug.Print "Name: " & g2it.Name
    Debug.Print "Value: " & CStr(g2it.Value)

    ' Display the user defined attributes
    ' This will only work with simple attribute types
    ' ----------------------------------------------
    n = g2it.Count
    atNames = g2it.AttrNames
    For i = 0 To n - 1
      Debug.Print CStr(i + 1) & ": " & atNames(i) & _
        ": " & CStr(g2it(i))
    Next i

    ' Display whether or not g2it references an
    ' an item in G2
    ' ------------------------------------
    If g2it.Linked Then
      Debug.Print "Linked to an item in G2"
    Else
      Debug.Print "Not linked to an item in G2"
    End If

    ' Print the list of active subscriptions
    ' -------------------------------------
    Debug.Print "================================")
   Debug.Print "Subscriptions:"
   vx = git.Subscriptions()
   For i = LBound(vx) To UBound(vx)
        Debug.Print i ;
        If vx(i, 1) = 1 Then
            Debug.Print " : Modify : " & vx(i, 2)
        ElseIf vx(i, 1) = 2 Then
            Debug.Print " : Delete"
        ElseIf vx(i, 1) = 3 Then
            Debug.Print " : Color Change"
        ElseIf vx(i, 1) = 4 Then
            Debug.Print " : Custom : " & vx(i, 2)
```

```
        ElseIf vx(i, 1) = 5 Then
            Debug.Print " : Value change"
        ElseIf vx(i, 1) = 6 Then
            Debug.Print " : Add Item to Workspace"
        ElseIf vx(i, 1) = 7 Then
            Debug.Print " : Remove Item From Workspace"
        ElseIf vx(i, 1) = 8 Then
            Debug.Print " : Select Item in Window"
        Else
            Debug.Print " : Update this program"
        End If
    Next i
    Debug.Print "================================"
End If
```

A

Example Code

This appendix presents some of the sample code that accompanies the

Introduction **135**

Using G2 ActiveXLink in Microsoft Visual Basic **135**

Using G2 ActiveXLink in Microsoft Excel **137**

Introduction

The appendix contains the complete code, HTML markup, and VBScript for the examples in this guide.

Using G2 ActiveXLink in Microsoft Visual Basic

The Cycle Lights program enables you to communicate with G2 by using G2 ActiveXLink. You can click the action button Cycle Lights in G2 to change the traffic light settings in G2 and Visual Basic. In Visual Basic, you can click the Cycle Lights buttons to change the traffic light settings in Visual Basic and G2.

For more information on the Cycle Lights program, see How to Communicate with G2 on page 14.

The following example is for non-.NET versions of Visual Basic. Users of VB .NET should refer to Using G2 ActiveXLink with Visual Basic .NET on page 20.

To run the demonstration program:

1. Load `axldemo.kb` into G2.

   The KB is located in the `demos` subdirectory of the `activexlink` directory of your G2 product directory.

2. Start G2.

3. Run `VBDemo.exe`.

   The program is located in the `vbdemo` subdirectory of the `demos` subdirectory.

The following code specifies the connection in Visual Basic:

```
Dim NextMode As String
Dim RedOn, RedOff, YellowOn, YellowOff, GreenOn,
   GreenOff As Long

Private Sub CallRPC_Click()
   rannum = G2Gateway1.Call("G2RANDOMGENERATOR",
      Val(CallItem.Text))
   CallItemRetVal = str(rannum)
End Sub

Private Sub Update_Light(Mode As String)
   If Mode = "PROCEED" Then
      Redlight.FillColor = RedOff
      Yellowlight.FillColor = YellowOff
      Greenlight.FillColor = GreenOn
   ElseIf Mode = "STOP" Then
      Redlight.FillColor = RedOn
      Yellowlight.FillColor = YellowOff
      Greenlight.FillColor = GreenOff
   Else
      Redlight.FillColor = RedOff
      Yellowlight.FillColor = YellowOn
      Greenlight.FillColor = GreenOff
   End If
End Sub

Private Sub CycleLights_Click()
   Call G2Gateway1.Start("CHANGE-SIGNAL", NextMode)
   If NextMode = "stop" Then
      NextMode = "slow"
   ElseIf NextMode = "slow" Then
      NextMode = "proceed"
   Else
      NextMode = "stop"
   End If
End Sub
```

```
Private Sub Form_Load()
   RedOn = &HFF&
   RedOff = &H40&
   YellowOn = &HFFFF&
   YellowOff = &H4040&
   GreenOn = &HFF00&
   GreenOff = &H4000&

   NextMode = "slow"
   Redlight.FillColor = RedOn
   Yellowlight.FillColor = YellowOff
   Greenlight.FillColor = GreenOff
End Sub

Private Sub G2Gateway1_Error(ByVal ErrorMessage As String,
      ByVal ErrorCode As Long,
      DeferredCallIdentifier As Variant)
   MsgBox ErrorMessage
End Sub

Private Sub G2Gateway1_RpcStarted(ByVal Name As String,
      InArgs As Variant)
   Dim str As String
   str = InArgs
   If Name = "CYCLELIGHTS" Then Call Update_Light(str)
End Sub

Private Sub StartRPC_Click()
   G2Gateway1.PostMessage StartItem.Text
End Sub
```

Using G2 ActiveXLink in Microsoft Excel

The Excel demonstration program enables you to update a graph from data supplied by the G2 server.

For more information on the demonstration program, see Calling a Procedure in G2 and Excel on page 33.

To run the demonstration program:

1. Load `axldemo.kb` into G2.

   The KB is located in the `demos` subdirectory of the `activexlink` directory of your G2 product directory.

2. Open `Gateway.xls`.

   The file is located in the `exceldemo` subdirectory of the `demos` subdirectory.

The following code specifies the connection between G2 and Excel in Visual Basic for Applications:

```
Private Sub CommandButton1_Click()
   Ret = Range("myString")
   Call G2Gateway1.PostMessage(Ret)
End Sub

Private Sub CommandButton2_Click()
   Range("myString") = G2Gateway1.Call("G2StringGenerator")
End Sub

Private Sub CommandButton3_Click()
   Range("A19:C24") = G2Gateway1.Call("G2ChartGenerator",
      100) ' Get values from G2
End Sub

Private Sub G2Gateway1_Error(ByVal ErrorMessage As String,
      ByVal ErrorCode As Long,
      DeferredCallIdentifier As Variant)
   MsgBox ErrorMessage
End Sub

Private Sub G2Gateway1_G2Paused()
   MsgBox ("G2 Paused.")
End Sub

Private Sub G2Gateway1_G2Resumed()
   MsgBox ("G2 Resumed.")
End Sub

Private Sub Worksheet_SelectionChange(ByVal Target
   As Excel.Range)
End Sub
```

Index

A

ActiveX control 1
adding G2 ActiveXLink from Control Toolbox in Excel 30
array 76
arrays, using in G2 ActiveXLink 110
authorizing G2 ActiveXLink with g2com.kb 7
Available References dialog 10
AxG2Gateway 26
  adding 24
  programs that use 26
axldemo.kb 7

B

blocking call
  `call` method 59
  `CallTimeout` property 56
blocking connection
  automatically created 58
  creating with `Connect()` 65
blocking RPC calls 56
Boolean data type 46
BSTR data type 46
Byte data type 46

C

C++, using G2 ActiveXLink with 40
`Call` method 59
  using in Excel 34
`CallDeferred( )` method 63
calling thread 64
`CallTimeout`
  basic property 12
  property 56
cells 33
change-signal G2 procedure 18
chart in Excel 33
client connection to G2 56
Clients 2
COM automation data type
  BSTR 46
  byte 46
  currency 47
  date 47
  double 46
  long 46
  null 47
  SAFEARRAY 46
  short 46
  VARIANT 46
  VARIANT_BOOL 46
com-array-type member 49
com-byte member 47
COM-compliant container application 2
com-currency member 49
com-day member 48
com-day-of-week member 48
com-dimensions member 49
com-double member 47
com-elements member 49
com-hour member 48
com-integer member 47
com-long member 47
com-lower-bounds member 49
com-minute member 48
com-month member 48
Component Object Model (COM)
  and ActiveX and OLE 1
  and development languages 45
  distributing events 77
components 1
Components option
  Visual Basic 8
  Visual Basic .NET 25
com-second member 48
com-single member 47
com-year member 48
configuration information 2
`Connect` method
  definition of 65
  using 14

connecting to G2 14
connection
identifying 56
status 56
container 2
Control Toolbox in Excel 30
currency
mapping 49
type 47
currency data type 46
customer support services xiv
Cycle Lights program
complete code 137
running 17
`CycleLights_Click()` Visual Basic function 18

D

data types
introduction to 45
mapping 46
date and time, mapping 48
date data type 47
Design Mode in Excel 30
`Disconnect` method 66
disconnect, explicitly 14
`DisconnectOnReset` basic property 12
`DisconnectOnReset` property 57
displaying text and data on G2? Message Board 67
double data type 46

E

error
message text 94
nonblocking calls 94
numeric representation 94
`Error` event 94
event handlers 68
events 2
`Error` 94
`G2Connected` 78
`G2Disconnected` 79
`G2Paused` 81
`G2Resumed` 82
`RpcCalled` 73
`RpcReturned` 80
`RpcStarted` 76
example form 13
example programs, running 7
example projects 6
Excel demonstration program
complete code 139
Excel, using G2 ActiveXLink with 29
exceptions 60

F

form
building 13
example 13
placing control on
Visual Basic 8
`Form_Load( )` function 3

G

G2 ActiveXLink
creating
`G2Item` 120
`G2Window` 115
`G2Workspace` 114
deleting a `G2Item` 124
example, using C++ 40
G2 type names 116
getting attribute names, values, and types 127
getting information about subscriptions 133
getting the icon for a `G2Item` 123
refreshing a `G2Item` 126
subscribing to item events
attribute changes 130
custom events 132
icon color changes 131
introduction to 129
item deletion 131
variable and parameter value changes 131
subscribing to window events 116
subscribing to workspace events 114
subscription types 117
unlinking items 126
unsubscribing from item events
all 133
attribute changes 132
custom events 132

icon color changes 132
item deletions 132
variable and parameter value changes 133
updating an item in G2 124
using item references 119
using linked items as parameters to RPCs 128
using symbols 98
using the G2Item class 103
using the G2ListOrArray class 110
verifying linked items 126
G2 ActiveXLink control
calling by reference 10
dropping on Visual Basic form 9
inserting in Visual Basic 8
setting properties 11
G2 data type
integer 46
item-or-value 46
sequence 47
text 46
truth-value 46
G2 Gateway interface object 56
G2 remote procedures
g2com-call 69
g2com-start 70
g2com-start-over-interface 71
G2 servers
connecting to 14
linking to more than one 13
multiple 9
g2com.kb module 7
g2com-call remote procedure 69
g2com-interface class 55
g2com-start remote procedure 70
g2com-start-over-interface remote procedure 71
G2Connected event 78
g2-current-remote-interface( ) system procedure
finding interface object 72
using 56
G2Disconnected event 79
G2Gateway
class 9
instances 9
object 9
G2Gateway1 9
G2Item
creating in G2 ActiveXLink 120
deleting in G2 ActiveXLink 124
getting the icon for 123
refreshing in G2 ActiveXLink 126
subscribing to item events 129
unlinking in G2 ActiveXLink 126
updating G2 items from G2 ActiveXLink 124
using in G2 ActiveXLink 103
verifying linkage in G2 ActiveXLink 126
G2Location
basic property 12
property defined 55
G2Paused event 81
G2Reset event 83
G2Resumed event 82
G2RunState property 57
G2RunStateKnown event 85
G2Started event 84
G2Structure Visual Basic object type 47
G2Symbols
property defined 55
G2Symbols basic property 12
G2Window
creating in G2 ActiveXLink 115
getting the G2 user mode 115
subscribing to window events 116
G2Workspace
creating in G2 ActiveXLink 114
subscribing to workspace events 114
Gensym G2 Gateway
Visual Basic 8
Visual Basic .NET 23
GensymAxl 24

H

host machine name 12
HTML
complete markup for demonstration program 141

I

identifiers
for errors from deferred calls 94
for return arguments from a deferred call 80

to identify a deferred call 63
integer data type 46
interface object, finding 71
InterfaceClass
basic property 12
property defined 55
Internet Explorer, using G2 ActiveXLink with 35
IsG2Connected property 56
item-or-value G2 data type 46

L

lists, using in G2 ActiveXLink 110
long data type 46

M

mapping
currency 49
date and time 48
multidimensional arrays 49
simple data types 47
structures 99
matching the return values with the original call 63
members
com-array-type 49
com-byte 47
com-currency 49
com-day 48
com-day-of-week 48
com-dimensions 49
com-double 47
com-elements 49
com-hour 48
com-integer 47
com-long 47
com-lower-bounds 49
com-minute 48
com-month 48
com-second 48
com-single 47
com-year 48
merging g2com.kb 7
Message Board
displaying text and values in 67
posting to 15
methods
Call() 59
CallDeferred() 63
Connect() 65
Disconnect 66
introduction to 2
Start() 61
multidimensional arrays 50
multi-threaded applications 2

N

natural language 45
.NET
See Visual Basic .NET
.NET, using G2 ActiveXLink with 20
network address of a G2 server 55
nonblocking calls, errors 94
nonblocking connection 65
null data type 47

O

Object Linking and Embedding (OLE) 1
objects, communication using 1
one-dimensional arrays 50

P

populating cells 33
port number 12
PostMessage( ) method 67
procedures
calling 2
example of calling 16
properties 2
basic G2 ActiveXLink 11
CallTimeout 56
defined 54
DisconnectOnReset 57
G2Location 55
G2RunState 57
G2Symbols 55
InterfaceClass 55
IsG2Connected 56
modifying 54
RemoteInitializationString 56
setting programmatically 13
Properties Window
in Excel 31

setting properties in 11
Property Pages, displaying 11
publish/subscribe mechanism 77

R

random numbers, generating 16
references in Visual Basic 10
RemoteInitializationString
basic property 12
property defined 56
return values
and started G2 procedure 61
from G2 68
RpcCalled event
and subscribers 77
definition of 73
RpcReturned event
and CallDeferred() method 63
definition of 80
RPCs, passing linked items, using 128
RpcStarted event
and subscribers 77
definition of 76

S

SAFEARRAY 46
sequence G2 data type 47
sequence of event handling 68
sequence of values 73
setting properties programmatically 13
short data type 46
single data type 47
spreadsheet 29
Start( ) method 61
string data type 46
structure G2 data type 47
structures, mapping 99
subscriber 77
symbols, using in G2 ActiveXLink 98

T

TCP/IP network 14
text data type 46
toolbox
adding a control
Visual Basic 8
Visual Basic .NET 25
traffic signal 18
truth-value data type 46

U

unique identifier 63
Update_Light() Visual Basic function 19

V

Variant data type 46
VARIANT_BOOL 46
VBScript, complete code for demonstration program 141
Visual Basic .NET, using G2 ActiveXLink with 20
Visual Basic data type
Boolean 46
byte 46
currency 46
date 47
Double 46
Integer 46
Long 46
Null 47
single 47
String 46
Variant 46
Visual Basic, using G2 ActiveXLink with 7