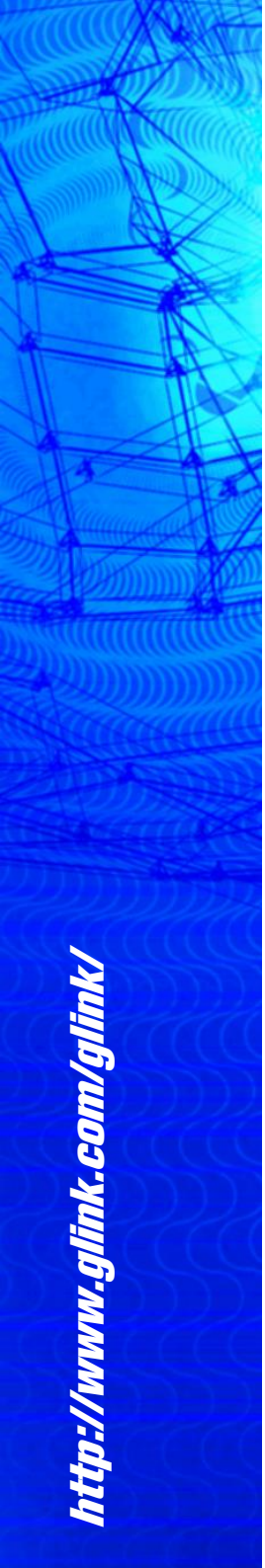

GLINK™

PROFESSIONAL EDITION
ENTERPRISE EDITION

API Reference Manual

http://www.glink.com/glink/

GALLAGHER ROBERTSON

Glink Enterprise Edition, version 8.6
Glink Professional Edition, version 8.6

GALLAGHER & ROBERTSON A/S, Grini Næringspark 3, N-1361 Oslo, Norway
Tel: +47 23357800
www: http://www.glink.com/
e-mail: support@glink.com

Contents

Contents

General

GLINK provides both the standard windows APIs (Application Programmable Interface) and the more specific emulator APIs from IBM and Bull.

GlinkApi COM+ component

COM+ and OLE Automation allows you to manipulate objects from other applications using the Component Object Model (COM). COM defines how objects interact between applications.

The Glink COM+ component is a complete and comprehensive API designed for a wide range of uses from E-business servers such as Microsoft's IIS or MTS, for client based applications written in languages such as C, Delphi, Visual Basic (VB), Visual Basic for Applications (VBA), to simple macros used by office applications such as the Office 97 suite from Microsoft.

Glink.Auto OLE Automation

Glinks OLE Automation object (Glink.Auto) was designed more for simple macros written for office applications.

DDE (Dynamic Data Exchange)

The GLINK DDE interface will transfer data to other applications using the standard Windows text format, OEM format or using the extensions to the Windows character set used internally by GLINK. (The GLINK character set may be inspected using Alt+F9 when running GLINK - it is a superset of the standard set). To use this last character set you must obtain a clipboard format identifier by using the RegisterClipboardFormat call with the name 'GlinkFont'. The format identifier may then be used in DDE requests in place of cf_Text.

HLLAPI (High-Level Language API)
UVTI (Unified Virtual Terminal Interface)

The HLLAPI and UVTI APIs are specified by IBM and Bull and give emulation access via DLLs supplied with Glink

GlinkApi COM+ component

Overview

COM+ is an evolutionary extension to the Component Object Model making it even easier for developers to create software components in any language, using any visual development tool.

The Glink COM+ component is a complete and comprehensive API designed for a wide range of uses from E-business servers such as Microsoft's IIS or MTS, for client based applications written in languages such as C, Delphi, Visual Basic (VB), Visual Basic for Applications (VBA), to simple macros used by office applications such as the Office 97 suite from Microsoft.

This GlinkApi is 100% compatible with the API supplied with the Glink for Java. E-business application developers who want a cross-platform application can easily migrate between the Glink for Java and Glink for Windows versions of the GlinkApi.

'Fat-client' (`Glink.GlinkApi`)

The fat-client GlinkApi COM+ component included in Glink is a single-instance out-of-process server. This means that each time a new GlinkApi object is created a new instance of GL.EXE is started. Each new instance is initially invisible and completely independent from any other instances. It is ideal for workstation client applications, it is not intended for server applications due to its larger footprint size and it's mono-threaded, single process architecture.

The fat-client GlinkApi class name is `Glink.GlinkApi` and is included in both the Glink Professional edition and Glink Enterprise edition.

Before you can use the GlinkApi component, Glink needs to register the GlinkApi classes in the workstation. Glink automatically registers the GlinkApi COM+ components when it is installed but you can explicitly register or unregister the `Glink.GlinkApi` class by executing:

```
gl.exe /regserver
gl.exe /unregserver
```

When the above commands are used, Glink will automatically exit after registering the GlinkApi class.

Once registered, you can immediately start using the `Glink.GlinkApi` class from other applications, or from visual development tools that can import COM+ components.

'Thin-client' (`GlinkApi.GlinkApi`)

The thin-client GlinkApi COM+ component included in Glink is a multi-threaded multi-object out-of-process server. This means that each time a new GlinkApi object is created it is created within the same GLINKAPI.EXE instance and only requires memory allocation for the new object. Each new instance is always invisible and completely independent from any other instances. It is ideal for workstation client applications and perfect for server applications due to its small footprint size, its multi-threaded and multi-object architecture.

The GLINKAPI.EXE server is started automatically on demand by the Windows system and is completely invisible to the user. To help developers, it can also be started upfront with the /J command line option to enable a debug window used for trace output.

The thin-client GlinkApi class name is `GlinkApi.GlinkApi` and is included in both the Glink Professional edition and Glink Enterprise edition.

Before you can use the GlinkApi component, the GlinkApi server needs to register the GlinkApi classes in the workstation. Glink automatically registers the GlinkApi COM+ components when it is installed but you can explicitly register or unregister the `GlinkApi.GlinkApi` class by executing:

```
glinkapi.exe /regserver
glinkapi.exe /unregserver
```

When the above commands are used, the GlinkApi server will automatically exit after registering the GlinkApi class.

Once registered, you can immediately start using the `GlinkApi.GlinkApi` class from other applications, or from visual development tools that can import COM+ components.

The thin-client GlinkApi COM+ component can execute within the MTS (Microsoft Transaction Server) architecture as an out-of-process server, making Glink the perfect host access component for your E-business applications on the Microsoft platform.

The GlinkApi component also includes the standard SESSION and APPLICATION objects available in Active Server Pages (ASP) within Microsoft's IIS.

Differences between the fat and thin clients

The main difference between the fat and thin-client versions of the GlinkApi Com+ object is that the fat-clients Glink.GlinApi object can be displayed by calling the setVisible method. When displayed, the fat-client is GL.EXE. Each new fat-client `Glink.GlinkApi` object starts the execution of a new instance of Glink. A thin-client `GlinkApi.GlinkApi` object resides as a single object within the multi-threaded GLINKAPI.EXE out-of-process server and cannot be displayed.

Both GL.EXE and GLINKAPI.EXE have approximately the same sized footprint in memory at start-up, however the creation of multiple thin-client `GlinkApi.GlinkApi` objects only requires additional memory for the objects data which is negligible compared to the memory requirements for starting up a new process.

As mentioned above, the object class names are different, `Glink.GlinkApi` for the fat-client and `GlinkApi.GlinkApi` for the thin-client. The interfaces are identical but the following methods have no effect in the thin-client API:

noScreen
noToolbar
scriptCommand
scriptFile
scriptTerminate
setSplashScreenVisible
setVisible

As there is no user interface to the object, the following GlinkEvent will not be generated:

GlinkEvent_CommandKeyTyped
GlinkEvent_KeyTyped

If a GlinkApi program is written without the need for Glink to be visible, then normally the only difference between using the fat or thin-client versions of the API is the actual class name used when the object is create, `Glink.GlinkApi` or `GlinkApi.GlinkApi`.

The Glink Professional edition license restricts to 5 simultaneous GlinkApi sessions per user. The Enterprise edition will be limited by license session's limit whether it is Glink or GlinkApi sessions.

Differences from the Glink for Java version

There are very few differences between the Java version of the GlinkApi and this Windows COM+ component version. Source code that has been written for one version should be easy to port to the other.

The main difference has to do with GlinkApi session names and configurations, the reason being that the configuration concept is different in Glink for Java to that used by Glink for Windows. Glink for Java session names correspond to a set of 'profile' options configured by the Glink for Java administration program which are configured and stored on the Glink for Java server. Glink for Windows session names correspond to configuration files stored locally in the Glink User directory. This difference affects the following GlinkApi methods:

GlinkApi.sessionName
GlinkApi.getAvailableSessions
GlinkApi.GlinkConfiguration

GlinkConfiguration.getName
GlinkConfiguration.setProfileName
GlinkConfiguration.getProfileName
GlinkConfiguration.setParameter

The session name corresponds to the name of the Glink configuration file with the ".glinkconfig" part stripped off, for example if you have a configuration file called "TSS at Phoenix.glinkconfig", then you can tell the GlinkApi to load this config file by using ***GlinkApi.sessionName("TSS at Phoenix")***. In the same way, the ***GlinkApi.getAvailableSessions*** method will return the list of configuration files, without the ".glinkconfig" part. This make the names more 'user-friendly' if the list is to be displayed in a list box.

When you use the ***GlinkApi.GlinkConfiguration*** constructor to create a new configuration object, the first parameter (the session name) may be an existing Glink configuration file. If so, then that file will be used as the initial configuration parameters. If not, then Glink default values will be used.

GlinkConfiguration.getProfileName and ***GlinkConfiguration.setProfileName*** have no effect in the Windows version as Glink for Windows has no concept of profiles.

Constant definitions are slightly differently in the Glink for Java and Windows versions due to differences in Java and COM+. In the Windows COM+ version, the constants are global values rather than part of the actual class, so Java syntax with a dot such as ***GlinkEvent.CONNECTED*** or ***GlinkKey.HOME*** become ***GlinkEvent_CONNECTED*** and ***GlinkKey_HOME***. This applies to all the defined constants, e.g.:

GlinkEvent_STARTED
GlinkConfiguration_PROTOCOL_TNVIP
GlinkField_ATR_HIDDEN
GlinkApi_SEARCH_FORWARD
GlinkKey_TRANSMIT

The following methods are not supported or applicable to the Windows versions

ConfigServer
serverPort
userName
userPassword
traceMethod

The Glink for Windows version of the API also contains special 'constructor' methods which allow you to create Glink objects that some GlinkApi methods require as parameters. These are:

GlinkApi.GlinkPoint	cursor position, X and y
GlinkApi.GlinkScreenArea	used mainly for setting ***GlinkEvent_SCREEN_AREA_MATCH*** notifications
GlinkApi.GlinkConfiguration	for creating new configuration objects
GlinkApi.GlinkKey	for creating a utility key object

Programming techniques

Below are a few examples in programming the GlinkApi. Full examples were installed in the "Samples" directory under the Glink installation directory.

Dual-interface, Type Library definition

The VTable definition is contained in GlinkApi's Type Library (TLB) included in the GL.EXE file or the supplied GLINK.TLB resource file. Including the TLB in

your visual development tool will normally allow the tool to display the GlinkApi methods, properties and events as you program. It also allows syntax checking at compile time. How you include the TLB definitions in your program is tool-dependent.

Using GlinkApi with MS Visual Basic

VB needs to load GlinkApi VTable Type Library (TLB) definition to be able to display the GlinkApi methods, properties and events as you program.

Including the GlinkApi definition is done via the Project/References... menu, which displays a list of references that are available. GlinkApi's type library reference is called "Glink Professional, COM+ and Automation Library" and is located in the GL.EXE file and "GlinkApi, COM+ Library" located in the GLINKAPI.EXE file. Once one of them has been included, you should see that the "Glink" object is added to the list of selectable objects when you try to define a variable, for example type:

```
Dim glapi as
```

and the list appears. Once you have selected "Glink", type a dot and a list of interfaces will be displayed:

```
Dim glapi as Glink.
```

Select "GlinkApi".

```
Dim glapi as Glink.GlinkApi
```

If you want the Glink.GlinkApi object created automatically as soon as it's used, then use:

```
Dim glapi as new Glink.GlinkApi
```

To create a thin-client version of the object, then use:

```
Dim glapi as new GlinkApi.GlinkApi
```

The below set of examples will use `Glink.GlinkApi` rather than `GlinkApi.GlinkApi` as the Glink object, but in most cases they are completely interchangeable.

If you are going to use the GlinkApi's events, then you must use the WithEvents keyword. However, when using the WithEvents keyword you cannot use the New keyword and will need to create the object at runtime:

```
Dim WithEvents glapi as Glink.GlinkApi

Private Sub Form_Load()
  Set glapi = New Glink.GlinkApi
  'or Set glapi = CreateObject("Glink.GlinkApi")
End Sub
```

Once the object is created you can use any method or property directly.

```
Set glapi = New Glink.GlinkApi
glapi.SessionName ("my config.glinkconfig")
glapi.start
glapi.notifyString "LOGICAL ID--", False, 1, True
glapi.setVisible (True)
```

Lists of the relevant properties and methods will appear as soon as you type the dot (or bracket).

```
Private Sub DumpField(fld As Glink.GlinkField)
    Dim t As String

    If TypeOf fld Is Glink.GlinkField Then
        t = fld.getString
        If fld.isProtected Then
            List1.AddItem ("Protected Field : " + t)
        Else
            List1.AddItem ("Variable Field: " + t)
        End If
        List1.AddItem ("X: " + Str(fld.getStart.X))
        List1.AddItem ("Y: " + Str(fld.getStart.Y))
        List1.AddItem ("L: " + Str(fld.getLength))
        List1.AddItem ("A: " + Hex(fld.getAttribute))
    End If
End Sub

Private Sub Command6_Click()
    Dim I, n As Integer
    Dim t As String
    Dim flds As GlinkFields

    Dim fld As GlinkField
    Dim f As GlinkField

    Set flds = glapi.getFields
    If TypeOf flds Is Glink.GlinkFields Then
        I = 0
        n = flds.getCount
        List1.AddItem ("Number of Field = " + Str(n))
        While I < n
            Set fld = flds.Item(I + 1)
            If TypeOf fld Is Glink.GlinkField Then
                List1.AddItem ("Field text = " +
fld.getString)
                DumpField fld
            End If
            I = I + 1
        Wend
    End If
End Sub
```

If the WithEvents keyword was used when declaring your object variable, then your GlinkApi object name will be displayed in the Object list. In the above example it would be called "glapi". Select it and the Procedure will be filled with the available Glink.GlinkApi event, OnGlinkEvent. Selecting this Event will generate the function which will be called when Glink signals an event, the parameters indicate which event occured.

```
Private Sub glapi_onGlinkEvent(ByVal glevent As
Glink.IGlinkEvent)
    Dim Val As Integer
    Dim Ev As Integer

'    List1.AddItem ("Glevent text is: ")
'    List1.AddItem (glevent.getValueText)
    Ev = glevent.getEventCode
    Val = glevent.getValue
    If Ev = GlinkEvent_STARTED Then
      List1.AddItem ("glevent: STARTED")
      GlStarted = True
    ElseIf Ev = GlinkEvent_STOPPED Then
      List1.AddItem ("glevent: STOPPED")
      GlStarted = False
    ElseIf Ev = GlinkEvent_CONNECTED Then
      List1.AddItem ("glevent: CONNECTED")
      GlConnected = True
    ElseIf Ev = GlinkEvent_DISCONNECTED Then
      List1.AddItem ("glevent: DISCONNECTED")
      GlConnected = False
    ElseIf Ev = GlinkEvent_TURN_RECEIVED Then
      List1.AddItem ("glevent: TURN_RECEIVED")
      ReadScreenContent
      GlKeybLocked = False
    ElseIf Ev = GlinkEvent_TURN_LOST Then
      List1.AddItem ("glevent: TURN_LOST")
      GlKeybLocked = True
    ElseIf Ev = GlinkEvent_STRING_RECEIVED Then
      List1.AddItem ("glevent: STRING_RECEIVED: " +
                     str(glevent.getValue))
      If Val = 1 Then
        glapi.SendKeys ("myuserid")
        glapi.sendCommandKey (Transmit)
        glapi.SendKeys ("logo")

        glapi.sendCommandKey (Transmit)
      End If
    End If

End Sub
```

Below is an example of reading the full Glink screen line by line:

```
Private Sub UpdateScreenContent()
    Dim R, I As Integer
    Dim startline As GlinkPoint
    Dim endline As GlinkPoint
    Dim S As String

    Set startline = glapi.GlinkPoint(1, 1)
    Set endline = glapi.GlinkPoint(80, 1)
    R = glapi.getRows
    I = 1
    List2.Clear
    While I <= R
        startline.Y = I
        endline.Y = I
        S = glapi.getString(startline, endline)
        List2.AddItem S
        I = I + 1
    Wend
End Sub
```

Using GlinkApi with Borland Delphi

Delphi needs to load the GlinkApi VTable Type Library (TLB) definition to be able to display the GlinkApi methods, properties and events as you program.

Including the GlinkApi definition is done via the Project/Import type library... menu, which displays a list of libraries that are available. The GlinkApi type library reference is called "Glink Professional, COM+ and Automation Library" and is located in the GL.EXE file. If it is not displayed in the list, click the Add button and select the GL.EXE file in the Glink directory. Once youve selected the "Glink Professional, Automation and COM+ Library" click install which will add it to the DCLUSER50.DPK file and rebuilt.

The TGlinkApi object will then be added to the ActiveX Component palette and you can drag the component into your applications form. The following variable will be added to the Form.

```
GlinkApi1: TGlinkApi;
```

Add Glink_TLB.PAS to the uses section.

Create and use the Glink.GlinkApi object in the following way:

```
GlinkApi1 := TGlinkApi.Create(Self);
GlinkApi1.SetupSession(1); (* my '7800local.glinkconfig'
*)
GlinkApi1.start; (* connection will be started
automatically *)
GlinkApi1.notifyString('LOGICAL ID--', false, 1, true);
GlinkApi1.setVisible(true);
```

Lists of the relevant properties and methods will appear as soon as you type the dot (or bracket). Below are some examples of using the GlinkApi interface:

```
procedure TForm1.DumpCursor (const cursor : IGlinkPoint);
begin
  if Cursor = nil then Exit;
  Listbox1.Items.Add('X: '+IntToStr(Cursor.X));
  Listbox1.Items.Add('Y: '+IntToStr(Cursor.Y));
end;

procedure TForm1.DumpField (const Field : IGlinkField);
var
  S : string;
  cursor : IGlinkPoint;
begin
  if Field = nil then Exit;
  cursor := Field.getStart;
  DumpCursor(cursor);
  S := Field.getString;
  Listbox1.Items.Add('S: ' + S);
end;

procedure TForm1.DumpScreenArea (const sArea :
IGlinkScreenArea);
var
    startPt : IGlinkPoint;
    endPt : IGlinkPoint;
begin
    If sArea = nil Then

        Listbox1.Items.Add ('No ScreenArea')
    Else
      Begin
        If sArea.isRectangle Then
            Listbox1.Items.Add
                          ('Rectangle ScreenArea mode')
        Else
            Listbox1.Items.Add ('Line ScreenArea mode');
        startPt := sArea.getStart;
        endPt := sArea.getEnd;
        Listbox1.Items.Add ('Start: ' +
                             IntToStr(startPt.X)+','+
                             IntToStr(startPt.Y));
        Listbox1.Items.Add ('End:   ' +
                             IntToStr(endPt.X)+','+
                             IntToStr(endPt.Y));
        Listbox1.Items.Add ('Text: ['+
                             sArea.getString+']');
      end;
end;

procedure TForm1.DumpFields (const Fields :
IGlinkFields);
var
  I, Count : integer;
  S : string;
```

```
  Field : IGlinkField;
begin
  if Fields = nil then Exit;
  Count := Fields.getCount;
  Listbox1.Items.Add('Number of fields:
'+IntToStr(Count));
  if Count = 0 then Exit;
  for I := 1 to Count do
    begin
      Listbox1.Items.Add('Field index: '+IntToStr(I));
      Field := Fields.item(I);
      DumpField(Field);
    end;
end;

procedure TForm1.Button6Click(Sender: TObject);
var
  I, Count : integer;
  Fields : IGlinkFields;

  Field : IGlinkField;
  cursor : IGlinkPoint;
  S : string;
begin
  Fields := GlinkApi1.getFields;
  cursor := GlinkApi1.GlinkPoint(5, 15);
  Listbox1.Items.Add('ByPosition');
  Field := Fields.findByPosition(cursor);
  DumpField(Field);
  cursor.X := 1;
  cursor.Y := 1;
  Listbox1.Items.Add('ByString');
  Field := Fields.findByString('password', cursor, 24*80,
GlinkApi_SEARCH_FORWARD, false);
  DumpField(Field);
end;

procedure TForm1.Button7Click(Sender: TObject);
var
  I, Count : integer;
  Fields : IGlinkFields;
  Field : IGlinkField;
  cursor : IGlinkPoint;
  S : string;
begin
  Fields := GlinkApi1.getFields;

  if Fields = nil then Exit;
  cursor := GlinkApi1.GlinkPoint(1, 1);
  Field := Fields.findByString('Enter user', cursor,
                 24*80, GlinkApi_SEARCH_FORWARD, false);
  if Field = nil then Exit;
  I := Fields.getFieldIndex (Field);
```

```
  Field := Fields.item(I + 1);
  if Field =  nil then Exit;
  Field.setString ('Phil');
  Field := Fields.findByString('Enter password', cursor,
                 24*80, GlinkApi_SEARCH_FORWARD, false);
  if Field = nil then Exit;
  I := Fields.getFieldIndex (Field);
  Field := Fields.item(I + 1);
  if Field =  nil then Exit;
  Field.setString ('Password');
  Fields.refresh;
  DumpFields(Fields);
end;

procedure TForm1.Button8Click(Sender: TObject);

var
  I, Count : integer;
  Fields : IGlinkFields;
  Field : IGlinkField;
  cursor : IGlinkPoint;
  S : string;
begin
  Fields := GlinkApi1.getVariableFields;
  if Fields = nil then Exit;
  Field := Fields.item(1);
  if Field = nil then Exit;
  Field.setString ('Phil');
  Field := Fields.item(2);
  Field.setString ('Password');
  Fields.refresh;
  DumpFields(Fields);
end;

procedure TForm1.Button9Click(Sender: TObject);
var
    startPt : IGlinkPoint;
    endPt : IGlinkPoint;
    sArea : IGlinkScreenArea;
begin
    sArea := GlinkApi1.getMarkedScreenArea;
    DumpScreenArea(sArea);
    if sArea <> nil then

      begin
        glinkapi1.notifyScreenArea(sArea,
                                   sAindex, false);
        inc(sAindex);
        sArea := GlinkApi1.getScreenArea (sArea.getStart,
                     sArea.getEnd, sArea.isRectangle);
        DumpScreenArea(sArea);
      end
    else
```

```
        begin
          startPt := glinkapi1.GlinkPoint(17, 6);
          endPt := glinkapi1.GlinkPoint(30, 6);
          sArea := glinkapi1.GlinkScreenArea(startPt,
                          endPt, true, 'identification');
          glinkapi1.notifyScreenArea(sArea, sAindex,
                                              false);
          DumpScreenArea(sArea);
          inc(sAindex);
        end;
end;
```

Below is an example of loading an existing Glink configuration file, getting a list of available sessions or creating a new temporary session configuration:

```
procedure TForm1.SetupSession(sesstype: integer);
var
  V : Variant;
  I, N : Integer;
  Conf : IGlinkConfiguration;
begin
  case sesstype of
  (* We select a predefined Glink config file 'my
config.glinkconfig' *)
  1 : GlinkApi1.SessionName ('my config');
  (* We select the first in the available list *)
  2 :
    begin
      V := GlinkApi1.getAvailableSessions;
      if not VarIsNull(V) and not VarIsEmpty(V) then
        begin
          N := VarArrayHighBound(V, 1);
          (* just list all we get *)
          for I := 0 to N do
            Listbox1.Items.Add('Session: ' + V[I]);
          GlinkApi1.SessionName (V[0]);
        end;
    end;
  (* we create our own from scratch *)
  3:
    begin
      Conf := GlinkApi1.GlinkConfiguration
                  ('My very own host config',
                  GlinkConfiguration_EMULATION_VIP7804,
                  GlinkConfiguration_PROTOCOL_TNVIP,
                  'localhost');
      if Conf = nil then
        Listbox1.Items.Add
         ('Unable to create a new configuration object')
      else
        begin
          GlinkApi1.addConfiguration (Conf);
          GlinkApi1.sessionName (Conf.getName);
        end;
    end;
  else (* Glink displays open config dialogbox *)
    GlinkApi1.SessionName ('');
  end;
end;
```

Add a Glink.GlinkApi event handler by double clicking the OnGlinkEvent in the Events form of the Object inspector.

```
procedure TForm1.GlinkApi1onGlinkEvent(Sender: TObject;
  var glevent: OleVariant);
begin

end;
```

To enable the event handler, add the following after the GlinkApi object creation and before starting the session:

```
GlinkApi1 := TGlinkApi.Create(Self);
GlinkApi1.OnonGlinkEvent := GlinkApi1onGlinkEvent;
GlinkApi1.start; (* connection will be started
automatically *)
```

Here is an example of a simple GlinkApi event handler:

```
procedure TForm1.GlinkApi1onGlinkEvent(Sender: TObject;
  var glevent: OleVariant);
var
  S : string;
  C : string;
  E : integer;
  V : integer;
  GlApi : IGlinkApi;
  Ev : IGlinkEvent;
begin
  Ev := IDispatch(event) as IGlinkEvent;
  E := Ev.getEventCode;
  V := Ev.getValue;
  GlApi := IDispatch(Ev.getSource) as IGlinkApi;
  if E = GlinkEvent_ERROR_DETECTED then
    S := 'ERROR_DETECTED: '+event.getValueText;
  if E = GlinkEvent_COMMAND_KEY_TYPED then
    begin
      S := 'COMMAND_KEY_TYPED: '+IntToStr(V);

      GlApi.sendCommandKey(V);
    end;
  if E = GlinkEvent_KEY_TYPED then
    begin
      C := Ev.getValueText;
      S := 'KEY_TYPED: '+ C;
      GlApi.sendKeys(C, nil);
    end;
  if E = GlinkEvent_MESSAGE_MODE_DATA then
    S := 'MESSAGE_MODE_DATA';
  if E = GlinkEvent_PRINT_DATA then
    S := 'PRINT_DATA';
  if E = GlinkEvent_STRING_RECEIVED then
    begin
      S := 'STRING_RECEIVED: id: '+IntToStr(V);
      case V of
        1 :
          begin (* 'LOGICAL ID--' *)
            GlApi.sendKeys ('myuserid', nil);
            GlApi.sendCommandKey (GlinkKey_TRANSMIT);
            GlApi.sendKeys ('logo', nil);
            GlApi.sendCommandKey (GlinkKey_TRANSMIT);
            GlApi.notifyString('password', true, 2,
                                           false);
            GlApi.notifyKey(-1);
            GlApi.notifyCommandKey(-1);
          end;
        2 :
          begin (* 'password' *)
            S := S + ': received password';
```

```
            end;
        end;
      end;
    if E = GlinkEvent_STARTED then
      S := 'STARTED'
    else if E = GlinkEvent_STOPPED then
      begin
        S := 'STOPPED';
        GlinkApi1.Free;
        GlinkApi1:= nil;

      end
    else if E = GlinkEvent_CONNECTED then
      S := 'CONNECTED'
    else if E = GlinkEvent_DISCONNECTED then
      S := 'DISCONNECTED'
    else if E = GlinkEvent_TURN_RECEIVED then
      S := 'TURN_RECEIVED'
    else if E = GlinkEvent_TURN_LOST then
      S := 'TURN_LOST'
    else if E = GlinkEvent_SCREEN_AREA_MATCH then
      S := 'E = SCREEN_AREA_MATCH: id: '+IntToStr(V);
    if S <> '' then
      Listbox1.Items.Add('Got GlinkEvent: ' + S)
    else
      Listbox1.Items.Add('Got untreated GlinkEvent: '+
                         IntToStr(E));
End;
```

Connecting with an existing Glink configuration file

Connecting using an existing configuration file is done by simply specifying the Glink configurations file name in the *GlinkApi.sessionName* property before calling the *GlinkApi.start* method. Normally you should not supply the ".glinkconfig" part of the config file name, if you do, then the GlinkApi will strip it. This means that any subsequent reads of the sesison name with the getSessionName method will return the name without the ".glinkconfig" part.

```
Dim WithEvents glapi As Glink.GlinkApi

  Set glapi = New Glink.GlinkApi

' We'll load the "TSS at Phoenix.glinkconfig" config file
  glapi.sessionName ("TSS at Pheonix")
  glapi.start
  glapi.notifyString "LOGICAL ID--", False, 1, True
  glapi.setVisible (True)
  etc...
```

Creating or modifying configuration parameters

Modifying Glink configuration options, must be done before calling the GlinkApi.start method. You must first create a GlinkConfiguration object, either from an existing Glink configuration file or a default, depending on the name you supply as the first parameter. Once the object is created, you can modify any of the documented parameters using the *GlinkConfiguration.setParameter* method. You then add the configuration object to the list of available sessions, and set the *GlinkApi.sessionName* to the name you supplied when you created the object.

```
Dim WithEvents glapi As Glink.GlinkApi
Dim glconf As Glink.GlinkConfiguration

  Set glapi = New Glink.GlinkApi

  Set glconf = glapi.GlinkConfiguration
           ("My new TP8 config",
           GlinkConfiguration_EMULATION_VIP7804,
           GlinkConfiguration_PROTOCOL_GGATE_DSA_CXI,
           "ggate.gar.no")
  glconf.setParameter ("emu.destructbs=true")
  glapi.addConfiguration glconf
  glapi.sessionName  (glconf.getName)
  glapi.start
  glapi.notifyString "LOGICAL ID--", False, 1, True
  glapi.setVisible (True)
  etc...
```

Intercepting print data

Print data can only be received by the GlinkApi program if the Glink configuration has been setup for GlinkApi printing. This can be done in the Glink Settings/Printer/Options dialog box by selecting GlinkApi in the "Host printing" and/or "Local printing" options, or it can be done with the *GlinkConfiguration.setParameter ("print.type=GlinkApi")* method at run-time. Print is then received as GlinkEvent_PRINT_DATA notifications in the *onGlinkEvent* event listener function.

```
Dim WithEvents glapi As Glink.GlinkApi
Dim glconf As Glink.GlinkConfiguration

  Set glapi = New Glink.GlinkApi

  Set glconf = glapi.GlinkConfiguration
          ("My new TP8 config",
           GlinkConfiguration_EMULATION_VIP7804,
           GlinkConfiguration_PROTOCOL_GGATE_DSA_CXI,
          "ggate.gar.no")
  glconf.setParameter("print.type=GlinkApi")
  glapi.addConfiguration glconf
  glapi.sessionName (glconf.getName)
  glapi.start
  glapi.notifyString "LOGICAL ID--", False, 1, True
  glapi.setVisible (True)
  etc...

Private Sub glapi_onGlinkEvent(ByVal glevent As
Glink.IGlinkEvent)
    Dim Val As Integer
    Dim Ev As Integer
    Dim S As String

    Ev = glevent.getEventCode
    Val = glevent.getValue
    If Ev = GlinkEvent_STARTED Then
      List1.AddItem ("glevent: STARTED")
      GlStarted = True
    ElseIf Ev = GlinkEvent_STOPPED Then
      List1.AddItem ("glevent: STOPPED")
      GlStarted = False
    ElseIf Ev = GlinkEvent_CONNECTED Then
      List1.AddItem ("glevent: CONNECTED")
      GlConnected = True
    ElseIf Ev = DISCONNECTED Then
      List1.AddItem ("glevent: DISCONNECTED")
      GlConnected = False
    ElseIf Ev = GlinkEvent_PRINT_DATA Then
      S = "glevent: PRINT_DATA: "
      If Ev.getValue = 1 Then
        S = S + "Last block: "
        S = S + "Data:" + Ev.getValueText
      EndIf
      List1.AddItem (S)
    EndIf
```

Glink.GlinkApi summary

Class Summary

GlinkApi	This is the main class for the API interface to Glink.
GlinkConfiguration	This class enables you to create a configuration used to set the necessary parameters for connecting to a given host application.
GlinkEvent	Identifies the type of events that a GlinkEventListener may receive.
GlinkField	A field is the fundamental element of a virtual screen.
GlinkFields	GlinkFields contains a collection of the fields in the virtual screen.
GlinkKey	Glink command keys constants.
GlinkScreenArea	Use GlinkScreenArea objects to facilitate screen/form identification.

Interface Summary

GlinkEventListener	Implement this interface to allow the class to be registered as a GlinkEventListener.

GlinkApi class

Class GlinkApi

GlinkApi
GlinkApi
class GlinkApi

This is the main class for the API interface to Glink. The class is contained in the glink.jar file delivered. The Java class path must be adjusted so this jar file can be found.

To start Glink from an application, create a GlinkApi object and supply the user name, password and session name with the userName, userPassword and sessionName methods. Then call the start method to launch Glink.

Restricted mode

When Glink is running as a J2EE connector, the J2EE application server will place the API in Restricted mode before passing it to client applications. When in Restricted mode, some methods will silently do nothing. These methods are marked as Restricted in this documentation.

Message mode and character sets

In message mode, Glink bypasses the emulation and sends and receives data directly between client and host. In emulation mode, clients always work in ASCII/UNICODE regardless of the host character set, as data is always assumed to be textual.

In message mode, binary data transmission is possible. This is implemented by two sets of message mode functions. One set works much like emulation mode in that data is always in ASCII/UNICODE as seen from the client, regardless of the host native character set. The other set allows transmission of raw data buffers in the host native character set. The difference between these two sets is that the former operates on String data, the latter operates on byte arrays. See the relevant messageModeXXX() methods below for details

Java only

Methods marked as Java only, indicated that they only appliy to Glink for Java and are irrelevant, unsupported of ignored by the Windows versions of Glink.

Field Summary

static int GlinkApi_ASCII_TO_EBCDIC

Specifies translation direction for translateBuffer(): ASCII to EBCDIC.

static int GlinkApi_EBCDIC_TO_ASCII

Specifies translation direction for translateBuffer(): EBCDIC to ASCII.

static int GlinkApi_SCREEN_FORMATTED

Specifies that one ore more fields are defined for the current screen.

static int GlinkApi_SCREEN_UNFORMATTED

Specifies that no fields are defined for the current screen.

static int GlinkApi_SCREEN_UNFORMATTED_CHAR

Specifies that no fields are defined for the current screen and data is sent to the host character by character and simultaneously displayed on the local screen.

static int GlinkApi_SEARCH_BACKWARD

Specifies the search direction for the findString method.

static int GlinkApi_SEARCH_FORWARD

Specifies the search direction for the findString method.

Method Summary

void addConfiguration(GlinkConfiguration glinkConfiguration)

Adds the configuration to the list of available sessions.

void addGlinkEventListener(GlinkEventListener listener)

Specifies the listener which is to receive Glink events.

void configBase(String configBase)

Sets the Glink configuration directory when running without a Glink Config Server.

void configLanguage(String lkey)

Specifies the language file to use for fixed texts.

void configServer(String address)

Specifies the IP address or host name of the computer where the Glink server is installed and running.

void connect()

Connects to the session specified with the sessionName() method.

void dedugStart(String debugFile)

Start or restart Glink debugging, send output to a file.

void debugStop()

Stop Glink debugging.

void disconnect()

Disconnects the current session.

void emulate(String hostData)

This method emulates a string locally.

String fieldRead(int fieldNum)

Returns the data for the given variable field.

void fileBase(String fileBase)

Sets the Glink file base directory.

GlinkPoint findString(String string, GlinkPoint start, int length, int direction, boolean caseSensitive)

Searches the text plane for the given string.

void flushGlinkEvents

Flushes the GlinkEvent queue.

int[ ] getAttributes(GlinkPoint start, int len)

Returns the attributes for the target area

int[ ] getAttributes(GlinkPoint start, GlinkPoint end)

Returns the attributes for the target area

Collection getAvailableSessions()

Returns the sessions configured.

int getBackgroundColor(int attribute)

Returns the RGB value for the given field or character attribute.

int getColumns()

Returns the number of columns in the presentation space.

int geCommunicationsProtocol()

Returns the current communication protocol.

GlinkPoint getCursor()

Returns the current cursor position.

byte[ ] getDataBytes(GlinkPoint start, int len)

Returns the screen characters for the target area in the character set used internally by Glink

byte[ ] getDataBytes(GlinkPoint start, GlinkPoint end)

Returns the screen characters for the target area in the character set used internally by Glink

int getDataLength(GlinkPoint start, GlinkPoint end)

Returns the length for the target area.

int getDefaultColorAttribute()

Returns the default color attribute settings.

int getEmulationType()

Returns the current emulation type.

GlinkFields getFields()

Returns the GlinkFields object associated with the current screen.

GlinkFields getFieldsEx()

Returns the GlinkFields object associated with the current screen.

int getForegroundColor(int attribute)

Returns the RGB value for the given field or character attribute.

GlinkEvent getGlinkEvent (int eventCode, long timeout)

Returns the first GlinkEvent available.

Collection getInteractiveComParams ()

Returns the interactive communication parameters.

GlinkScreenArea getMarkedScreenArea()

Returns a screen area object for the current screen selection.

int getRows()

Returns the number of rows in the presentation space.

GlinkScreenArea getScreenArea(GlinkPoint start, GlinkPoint end, boolean rectangle)

Returns an object for the given screen area or null if the positions given are not within the current screen.

String getScreenCharacterSet ()

Returns the name of the screen character set configured.

int getScreenMode()

If the host application has used field attributes to define fields on the screen, then the current screen is formatted.

String getSessionName()

Returns the name of the current session. **Restricted**.

String getString()

Returns the entire text plane of the presentation space as a string.

String getString(GlinkPoint start, GlinkPoint end)

Returns the text at the given location in the presentation space as a string.

GlinkFields getVariableFields()

Returns the GlinkFields object associated with the current screen with variable fields only.

GlinkFields getVariableFieldsEx()

Returns the GlinkFields object associated with the current screen with variable fields only.

boolean isConnected()

Returns true if Glink is connected to a session.

boolean isScreenAreaMatch(GlinkScreenArea sa)

Returns true if the supplied GlinkScreenArea object matches the current screen.

boolean isSessionAvailable(String name, Collection sessions)

Returns true if the given session is available.

boolean isSplashScreenVisible()

Returns true if the splash screen is visible.

boolean isStarted()

Returns true if Glink has started.

boolean isTurnKnown()

Returns true if Glink is connected to a host and the communication module in use knows about which side has the turn to send, either the host side or the Glink side.

boolean isTurnReceived()

Returns true if this side has the turn to send data.

boolean isVisible()

Returns true if the Glink screen is visible.

void markScreenArea(GlinkPoint start, GlinkPoint end)

This method marks the given screen area.

void messageModeBufferOverwrite(boolean overwrite)

Specifies if the message mode receive buffer may be overwritten if the limit of the buffer is reached.

void messageModeOff()

This method turns message mode off.

void messageModeOn(int bufsize, boolean processHostData)

This method instructs Glink to start message mode operation.

String messageModeReceive()

This method returns the string of characters received from the host since the last time this method was called.

byte[ ] messageModeReceiveBytes()

This method returns the byte array received from the host since the last time this method or messageModeReceive() was called.

String messageModeReceiveBytesSync (int maxlen)

This method waits until the requested number of characters has been received from the host before it returns these.

String messageModeReceiveSync (int maxlen)

This method waits until the requested number of characters has been received from the host before it returns these.

void messageModeSend(String s, boolean passTurn)

This method bypasses the Glink emulation and sends the string directly to the host.

void messageModeSendBytes(byte[ ] s, int offset, int len, boolean passTurn)

This method bypasses the Glink emulation and sends a byte array directly to the host.

void noScreen()

This method instructs Glink to start without the Glink screen display class.

void notifyCommandKey(int commandKey)

Instructs Glink to fire a GlinkEvent_COMMAND_KEY_TYPED event when the given command key is typed.

void notifyKey(int key)

Instructs Glink to fire a GlinkEvent_KEY_TYPED event when the given key is typed.

void notifyScreenArea(GlinkScreenArea screenArea, int identity, boolean removeAfterwards)

Instructs Glink to fire a GlinkEvent_SCREEN_AREA_MATCH event when the given object matches the current screen.

void notifyString(String string, boolean caseSensitive, int identity, boolean removeAfterwards)

Instructs Glink to fire a GlinkEvent_STRING_RECEIVED event when the given string is received from host.

void queueGlinkEvents(boolean enable)

Enables or disables the queuing of GlinkEvents.

void noToolbar()

This method instructs Glink to start without a toolbar and keybar.

void removeConfiguration(GlinkConfiguration glinkConfiguration)

Removes the configuration previously set with the addConfiguration method.

void removeGlinkEventListener(GlinkEventListener listener)

Removes the specified listener set with the addGlinkEventListener method.

void removeNotifyCommandKey(int commandKey)

Removes the notification for the command key previously set with the notifyCommandKey method.

void removeNotifyKey(int key)

Removes the notification for the key previously set with the notifyKey method.

boolean removeNotifyScreenArea(int identity)

Removes the GlinkScreenArea object associated with the given identity from the list of objects being checked for a match.

boolean removeNotifyString(int identity)

Removes the string associated with the given identity from the list of strings being checked for.

boolean restoreScreenContent (String fileName)

Restores the screen previously saved.

boolean saveScreenContent (String fileName)

Saves the current screen content to the file specified.

void scriptCommand(String command)

Activates the given script command.

void scriptFile(String fileName)

Activates the given script file.

void scriptTerminate()

Terminates the current script.

void sendCommandKey(int key)

The sendCommandKey method sends a "command" keystroke to the virtual screen.

void sendKeys(String text)

Sends a string to the emulator starting at the current cursor location.

void sendKeys(String text, GlinkPoint location)

Sends a string to the emulator starting at the given cursor location.

void serverPort(String port)

Set the IP port on which to connect to the Glink server.

boolean sessionName(String name)

Specifies the name of the session to which Glink should connect.

void setCursor(int x, int y)

Moves the cursor to the given position.

void setCursor(GlinkPoint location)

Moves the cursor to the given position.

void setGlinkDirectory(String glinkDir)

Set the Glink directory.

void setInteractiveComParams (Collection params)

Sets the interactive communication parameters.

void setRestrictedMode(String key, int level)

Switch restricted mode on or off.

void setScreenSize(int columns, int rows)

For most emulations the screen size is 80 columns times 24 rows.

void setSplashScreenVisible(boolean visible)

Specifies whether the splash screen should be visible.

void setString(String text, GlinkPoint location)

The setString method sends a string to the virtual screen at the specified location.

void setVisible(boolean visible)

Specifies whether Glink should be visible.

void start()

Starts Glink.

void stop()

Stops Glink.

void traceMethods(boolean trace)

Instructs the API to log the methods being called.

String translateBuffer(String buffer, int direction)

Translate a string between ASCII and EBCDIC.

void userName(String name)

Specifies the user name.

void userPassword(String password)

Specifies the password for the user.

Field Detail

GlinkApi_ASCII_TO_EBCDIC

static int GlinkApi_ASCII_TO_EBCDIC

Specifies translation direction for translateBuffer(): ASCII to EBCDIC.

GlinkApi_EBCDIC_TO_ASCII

static int GlinkApi_EBCDIC_TO_ASCII

Specifies translation direction for translateBuffer(): EBCDIC to ASCII.

GlinkApi_SCREEN_FORMATTED

static int GlinkApi_SCREEN_FORMATTED

Specifies that one ore more fields are defined for the current screen.

See also:
getScreenMode()

GlinkApi_SCREEN_UNFORMATTED

static int GlinkApi_SCREEN_UNFORMATTED

Specifies that no fields are defined for the current screen.

GlinkApi_SCREEN_UNFORMATTED_CHAR

static int GlinkApi_SCREEN_UNFORMATTED_CHAR

Specifies that no fields are defined for the current screen and data is sent to the host character by character and simultaneously displayed on the local screen.

GlinkApi_SEARCH_FORWARD

static int GlinkApi_SEARCH_FORWARD

Specifies the search direction for the findString method.

See also:
findString(String, GlinkPoint, int, int, boolean)

GlinkApi_SEARCH_BACKWARD

static int GlinkApi_SEARCH_BACKWARD

Specifies the search direction for the findString method.

See also:
findString(String, GlinkPoint, int, int, boolean)

Method Detail

addConfiguration

void addConfiguration(GlinkConfiguration glinkConfiguration)

Adds the configuration to the list of available sessions.

Restricted

See also:
sessionName(String), GlinkConfiguration

addGlinkEventListener

void addGlinkEventListener(GlinkEventListener listener)

Specifies the listener which is to receive Glink events.

See also:
GlinkEvent

configBase

void configBase(String configBase)

Sets the Glink configuration directory when running without a Glink Config Server. The configuration directory is located in a subdirectory to the directory where 'glink.jar' is installed. The default name of this configuration directory is 'config'. Normally there should be no need to use this method. Just make sure that 'glink.jar' is specified in the class path directive with a path name, for example 'd:\glinkj\glink.jar'.

Restricted

Parameters:

configBase - The path to the location where the Glink configuration directory is placed, for example: ***GlinkApi.configBase***("D:\\glinkj\\config");

configLanguage

void configLanguage(String lkey)

Specifies the language file to use for fixed texts. This determines the language used in menus, dialogs, message boxes and status line. Normally the value should be a two-letter abbreviation specifying the language; the default value is us. Glink looks for a file glink_xx.txt, where xx is the value of this parameter, in the .../no/gar/data directory.

Restricted

Parameters:

lkey - The 2-letter language key to use. Default is "us".

configServer

void configServer(String address)

Specifies the IP address or host name of the computer where the Glink server is installed and running. If configServer is not set, Glink assumes that the configuration server is running on the same computer as Glink. To run Glink without a Config Server, specify "none" as the server address.

Restricted; Java only

Parameters:

address - The address may be specified as: hostname.yourcompany.com or 123.234.123.1 or none

connect

void connect()

Connects to the session specified with the sessionName() method. At startup Glink automatically connects to the session specified with the sessionName method.

Restricted

See also:
sessionName(String)

debugStart

void debugStart(String debugFile)

Start or restart Glink debugging, send output to a file. If debug was already running it is restarted, and subsequent debugging output is sent to debugFile. If debugging wasn't running it is started.

Restricted

Parameters:

debugStream - destination file for debug output**.**

See also:
debugStop*()*

debugStop

void debugStop()

Stop Glink debugging.

Restricted

See also:
debugStart*(String)*

disconnect

void disconnect()

Disconnects the current session.

Restricted

emulate

void emulate(String hostData)

This method emulates a string locally. The string may be simple text or contain emulation escape sequences. Normally this command will be used to process host data obtained with the messageModeReceive method when the "processHostData" parameter in the messageModeOn method is set to FALSE.

See also:
messageModeOn(int, boolean)

fieldRead

String fieldRead(int fieldNum)

Returns the data for the given variable field.

Parameters:

fieldNum - The variable field number where 1 is the first variable field and 0 means the current field.

Returns:
The field value.

fileBase

void fileBase(String fileBase)

Sets the Glink file base directory. Glink needs some files located in the 'no' directory, normally located as a subdirectory of the directory where the 'glink.jar' file is found. There should be no need to use this method, the default file base directory is where the 'glink.jar' file is found. Just make sure that the 'glink.jar' is specified in the class path directive with a path name, for example 'd:\glinkj\glink.jar'.

Restricted

Parameters:

fileBase - The path to the location where the Glink 'no' directory is placed, for example: ***GlinkApi.fileBase***("D:\\glinkj");
GlinkApi.fileBase("http://www.gar.no/glinkj");

findString

GlinkPoint findString(String string, GlinkPoint start, int length, int direction, boolean caseSensitive)

Searches the text plane for the given string. Null characters in the text plane are treated as spaces during search processing.

Parameters:

string - The string for which to search.

startPos - The row and column in which to start. The position is inclusive (for example, row 1, col 1 means that position 1,1 will be used as the starting location and 1,1 will be included in the search).

length - The length from startPos to include in the search.

dir - Search direction value:

Constant	Value	Description
GlinkApi_SEARCH_FORWARD	0	Forward (beginning towards end)
GlinkApi_SEARCH_BACKWARD	1	Backward (end towards beginning)

caseSensitive - Indicates whether the search is to be case sensitive. True means the search will be case sensitive.

Returns:
If found, returns a Point object containing the location. If not found, returns a null. The string must be completely contained by the target area for the search to be successful.

flushGlinkEvents

void flushGlinkEvents

Flushes the GlinkEvent queue. Any GlinkEvents queued are removed from the queue.

See also:
getGlinkEvent(int, long), queueGlinkEvent(boolean), GlinkEvent addGlinkEventListener(GlinkEventListener)

getAttributes

int[ ] getAttributes(GlinkPoint start, int len)

Returns the attributes for the target area

Parameters:

startPos - The row and column in which to start.

length - The length (number of attributes integer) from ***startPos***.

See also:
Attribute format.

getAttributes

int[ ] getAttributes(GlinkPoint start, GlinkPoint end)

Returns the attributes for the target area

See also:
getAttributes(GlinkPoint, int), Attribute format.

getAvailableSessions

Collection getAvailableSessions()

Returns the sessions configured. Please note that if this method is called before Glink has started, the user name, password and configserver must be set prior this call or must have been configured, otherwise "null" will be returned and a ***GlinkEvent_ERROR_DETECTED*** will be posted.

Restricted

See also:
getAvailableSessions(), userName(String), userPassword(String), configServer(String), serverPort(String), GlinkEvent_VE_SESSIONS_ERROR

getBackgroundColor

int getBackgroundColor(int attribute)

Returns the RGB value for the given field or character attribute. The color returned is affected by the Glink screen color settings for the current session.

Returns:
The RGB value (Bits 24-31 are 0xff, 16-23 are red, 8-15 are green, 0-7 are blue)

See also:
GlinkField

getColumns

int getColumns()

Returns the number of columns in the presentation space.

Returns:
The number of columns.

getCommunicationProtocol

int getCommunicationProtocol()

Returns the current communication protocol.

Returns:
The communication protocol value defined int the GlinkConfiguration class or -1 if Glink is not connected.

See also:
GlinkConfiguration

getCursor

GlinkPoint getCursor()

Returns the current cursor position.

Returns:
The cursor position.

getDataBytes

byte[ ] getDataBytes(GlinkPoint start, int len)

Returns the screen characters for the target area in the character set used internally by Glink.

getDataBytes

byte[ ] getDataBytes(GlinkPoint start, GlinkPoint end)

Returns the screen characters for the target area in the character set used internally by Glink.

getDataLength

int getDataLength(GlinkPoint start, GlinkPoint end)

Returns the length for the target area.

getDefaultColorAttribute

int getDefaultColorAttribute()

Returns the default color attribute settings.

For example, the following call gets the RGB value for the default background color:
getBackgroundColor(getDefaultColorAttribute());

Returns:
The default color attribute settings. The foreground color is given in bits 0-3 the and background color is given in bits 4-6.

See also:
Attribute format. getForegroundColor(int) getBackgroundColor(int)

getEmulationType

int getEmulationType()

Returns the current emulation type.

Returns:
The emulation type value defined int the GlinkConfiguration class or -1 if Glink is not connected

See also:
GlinkConfiguration

getFields

GlinkFields getFields()

Returns the ***GlinkFields*** object associated with the current screen. ***GlinkFields*** contains all the fields in the current screen. For unformatted screens, the returned object contains only one input field that contains the whole presentation space.

See also:
GlinkField, getFields()

getFieldsEx

GlinkFields getFieldsEx()

Returns the ***GlinkFields*** object associated with the current screen. ***GlinkFields*** contains all the fields in the current screen. For formatted screens, this method will return the very same as the *getFields()* method.

For unformatted screens where no fields are defined, this method will simulate that one or more fields are defined. For VIP and DKU emulations set in text mode, an input field will be defined from the position of the cursor to the end of the line. For emulations running in character mode like VT, an input field will be defined from the cursor position. The end of this field will be at the same line but will depend of the line contents and attributes.

The rest of the screen will be presented as a fixed field. The result will often be that three fields are defined for the screen (one fixed field, one variable field and another fixed field at the end).

See also:
GlinkField, getFields ()

getForegroundColor

int getForegroundColor(int attribute)

Returns the RGB value for the given field or character attribute. The color returned is affected by the Glink screen color settings for the current session.

Returns:
The RGB value (Bits 24-31 are 0xff, 16-23 are red, 8-15 are green, 0-7 are blue)

See also:
GlinkField

getGlinkEvent

GlinkEvent getGlinkEvent(int eventCode, long timeout)

Returns the first ***GlinkEvent*** available from the GlinkEvent queue. Use the queueGlinkEvents method to enable GlinkEvents queuing. The GlinkEvents queuing can be used instead of a GlinkEvent listener to get hold of GlinkEvents.

If the eventCode parameter is set to 0, the first available GlinkEvent is returned. If the eventCode is set to a valid GlinkEvent code, then other GlinkEvents queued are flushed and the specified GlinkEvent is returned if available.

The GlinkEvent_TURN_LOST event is ignored and not reported by this method.

Parameters:

eventCode - The GlinkEvent code to wait for or 0 to wait for any.

timeout - Specifies the number of milliseconds to wait for GlinkEvents if not available or 0 if no waiting.

Returns:
First available GlinkEvent or null if no GlinkEvent is available.

See also:
queueGlinkEvent(boolean), flushGlinkEvent(), GlinkEvent
addGlinkEventListener(GlinkEventListener)

getInteractiveComParams

Collection getInteractiveComParams ()

Returns the interactive communication parameters. This method should be called only if the following event is posted:

GlinkEvent_INTERACTIVE_COM_PARAMS_REQUESTED

The Ggate and DSA communication modules may be configured with parameters that should be supplied or modified by the user when connecting to the host application. These parameters are referred to as interactive communication parameters. If the current communication module is configured with such interactive parameters and Glink itself is not visible (glink.setVisible(false)), then the GlinkEvent_INTERACTIVE_COM_PARAMS_REQUESTED is posted. The GlinkApi application should then call the getInteractiveComParams method to get the interactive parameters in question, modify the parameter values and set the interactive parameters with the setInteractiveComParams method call.

The getInteractiveComParams returns a Collection with one element per interactive parameter. Each element is a string array of 3. The first string contains the short name of the parameter, the second the long name and the third the actual parameter value. For example a parameter may have "pw" as the short name, "Password" as the long name and an empty string as the parameter value. Below is a code example where one of the parameter ("pw") is modified if present.

```
Collection v = glink.getInteractiveComParams();
String param[];
for (int i = 0; i < v.size(); i++) {
    param = (String[])v.elementAt(i);
    if (param[0].equals("pw")) {
        params[2] = "mypassword";
        v.set(i,param);
    }
}
glink.setInteractiveComParams(v);
```

Returns:
A Collection with one element per interactive config parameter.

See also:
setInteractiveComParams(Collection),
GlinkEvent_INTERACTIVE_COM_PARAMS_REQUESTED

getMarkedScreenArea

GlinkScreenArea getMarkedScreenArea()

Returns a screen area object for the current screen selection. The method checks the current screen and if any text is marked, returns a ***GlinkScreenArea*** object for the marked text. Please note that Glink supports two modes of marking; marking line-by-line and rectangular marking. Use ***GlinkScreenArea.isRectangle()*** to check the marking mode used.

Returns:
A ***GlinkScreenArea*** object describing the marked area or null if no text has been marked.

See also:
mark*ScreenArea(GlinkPoint, GlinkPoint, boolean),*
getScreenArea(GlinkPoint, GlinkPoint, boolean),
isScreenAreaMatch(GlinkScreenArea)

getRows

int getRows()

Returns the number of rows in the presentation space.

Returns:
The number of rows.

getScreenArea

GlinkScreenArea getScreenArea(GlinkPoint start, GlinkPoint end, boolean rectangle)

Returns an object for the given screen area or null if the positions given are not within the current screen.

Parameters:

start - The row and column in which to start. The position is inclusive (for example, row 1, col 1 means that position 1,1 will be used as the starting location and 1,1 will be included in the data).

end - The row and column in which to end. The position is inclusive (for example, row 10, col 12 means that position 10,12 will be used as the ending location and 10,12 will be included in the data).

rectangle - Supply true to get a rectangle area and false to get an area that is on a line-by-line basis from the given start position.

Returns:
A ***GlinkScreenArea*** object for the given area

See also:
getMarkedScreenArea(), notifyScreenArea(GlinkScreenArea, int, boolean), isScreenAreaMatch(GlinkScreenArea)

getScreenCharacterSet

String getScreenCharacterSet ()

Returns the name of the screen character set configured. The screen character set is used to transform byte arrays internally used by Glink to Unicode strings. The default value returned is **null** which means that no character set is configured. The **ISO-8859-1** standard will be used in this case. Semi-graphics characters are not defined and will always be returned as Unicode defined semi-graphics regardless of the screen character set configured.

getScreenMode

int getScreenMode()

If the host application has used field attributes to define fields on the screen, then the current screen is formatted. If there are no fields defined, then the current screen is unformatted.

Returns:
The screen mode for the current screen (***GlinkApi_SCREEN_FORMATTED***, ***GlinkApi_SCREEN_UNFORMATTED*** or ***GlinkApi_SCREEN_UNFORMATTED_CHAR***).

See also:
GlinkApi_SCREEN_FORMATTED, GlinkApi_SCREEN_UNFORMATTED, GlinkApi_SCREEN_UNFORMATTED_CHAR

getSessionName

String getSessionName()

Returns the name of the current session

Restricted

getString

String getString()

Returns the entire text plane of the presentation space as a string. All null characters and field attribute characters are returned as space characters.

Returns:
The entire text plane as a string

getString

String getString(GlinkPoint start, GlinkPoint end)

Returns the text at the given location in the presentation space as a string. All null characters and field attribute characters are returned as space characters.

Parameters:

start - The row and column in which to start. The position is inclusive (for example, row 1, col 1 means that position 1,1 will be used as the starting location and 1,1 will be included in the data).

end - The row and column in which to end. The position is inclusive (for example, row 10, col 12 means that position 10,12 will be used as the ending location and 10,12 will be included in the data).

Returns:
The text as a string at the given location

getVariableFields

GlinkFields getVariableFields()

Returns the ***GlinkFields*** object associated with the current screen with variable fields only. In other words only unprotected fields are contained in the returned ***GlinkFields*** object. For unformatted screens, the returned object contains only one ***GlinkField*** item that contains the whole presentation space.

See also:
GlinkField, getVariableFieldsEx()

getVariableFieldsEx

GlinkFields getVariableFieldsEx()

Returns the ***GlinkFields*** object associated with the current screen with variable fields only. In other words only unprotected fields are contained in the returned ***GlinkFields*** object. For formatted screens, this method will return the very same as the *getVariableFields()* method.

For unformatted screens where no fields are defined, this method will simulate that one input field is defined. For VIP and DKU emulations set in text mode, an input field will be defined from the position of the cursor to the end of the line. For emulations running in character mode like VT, an input field will be defined from the cursor position. The end of this field will be on the same line but will depend of the line contents and attributes.

See also:
GlinkField, getVariableFields()

isConnected

boolean isConnected()

Returns true if Glink is connected to a session.

isScreenAreaMatch

boolean isScreenAreaMatch(GlinkScreenArea sa)

Returns true if the supplied ***GlinkScreenArea*** object matches the current screen.

See also:
getScreenArea(GlinkPoint, GlinkPoint, boolean)

isSessionAvailable

boolean isSessionAvailable(String name, Collection sessions)

Returns true if the given session is available.

Restricted

See also:
getAvailableSessions()

isSplashScreenVisible

boolean isSplashScreenVisible()

Returns true if the splash screen is visible.

See also:
setSplashScreenVisible(boolean)

isStarted

boolean isStarted()

Returns true if Glink has started.

See also:
start()

isTurnKnown

boolean isTurnKnown()

Returns true if Glink is connected to a host and the communication module in use knows about which side has the turn to send, either the host side or the Glink side. When ***isTurnKnown()*** returns true, the Glink communication module signals the turn to the GlinkApi application with the ***GlinkEvent_TURN_LOST*** and ***GlinkEvent_TURN_RECEIVED*** event flags.

Communication modules like ComTN3270, ComTN5250, ComGgate ComDSA and ComTNVIP all know about the turn, whereas the ComTelenet and ComMinitel do not.

See also:
GlinkEvent_TURN_LOST, GlinkEvent_TURN_RECEIVED, isTurnReceived()

isTurnReceived

boolean isTurnReceived()

Returns true if this side has the turn to send data.

See also:
isTurnKnown()

isVisible

boolean isVisible()

Returns true if the Glink screen is visible.

See also:
setVisible(boolean)

markScreenArea

void markScreenArea(GlinkPoint start, GlinkPoint end, boolean rectangle)

This method marks the given screen area. Glink supports two modes of marking; marking done on a line-by-line basis and rectangular marking.

Parameters:

start - The starting position (1,1 is the upper left corner). Screen marking is removed by supplying an invalid screen position , for example (0,0).

end - The ending position (80,24 is the end of the screen).

rectangle - Sets the marking mode to be used.

See also:
getMarkedScreenArea()

messageModeBufferOverwrite

void messageModeBufferOverwrite(boolean overwrite)

Specifies if the message mode receive buffer may be overwritten if the limit of the buffer is reached. When the message mode receive buffer has reached its limit, the line module will by default be suspended until a message mode receive call has been issued to clear the buffer so more data can be accepted. If you allow the message mode receive buffer to be overwritten, the line module will clear the buffer and continue to receive the rest of the data without any delay.

Parameters:

overwrite - if true, Glink overwrites the message mode receive buffer when it has reached its limit, if false, Glink waits until one of the messageModeReceive methods are called to clear the buffer.

See also:
messageModeReceive(),messageModeReceiveBytes(), messageModeReceiveBytesSync(), messageModeReceiveSync(), messageModeBufferOverwrite(boolean), emulate(String)

messageModeOff

void messageModeOff()

This method turns message mode off.

Restricted

messageModeOn

void messageModeOn(int bufsize, boolean processHostData)

This method instructs Glink to start message mode operation. In message mode, all characters received from the host will be stored as "raw" data and made available to the application with either the messageModeReceive or messageModeReceiveSync calls.

Restricted

Parameters:

bufsize - Size of the buffer where the incoming characters are stored. The buffer is cleared with the messageModeReceive or messageModeReceiveSync calls. If the buffer reaches its limit, the line modul will be suspended until a receive call is issued to clear the buffer.

processHostData - Tells Glink to pass the received characters to the Glink emulation class when set to true. When set to false, the characters are stored in the message mode buffer only and the Glink emulation is bypassed.

See also:
messageModeReceive(),messageModeReceiveSync(int maxlen), messageModeBufferOverwrite(boolean), emulate(String)

messageModeReceive

String messageModeReceive()

This method returns the string of characters received from the host since the last time this method was called. Glink must be in message mode operation for this call to return any data.

Returns:
The string of character received or null if no data is available

See also:
messageModeReceiveBytes(), messageModeReceiveBytesSync(), messageModeReceiveSync(), messageModeBufferOverwrite(boolean), emulate(String)GlinkEvent_MESSAGE_MODE_DATA, emulate(String)

messageModeReceiveBytes

byte[ ] messageModeReceiveBytes()

This method returns the byte array received from the host since the last time this method or messageModeReceive() was called. Glink must be in message mode operation for this call to return any data. The returned byte array is always in the host native character set.

Returns:
an array of bytes received from the host, or null if no data available

See also:
messageModeReceive() ,messageModeReceiveBytesSync(), messageModeReceiveSync(), messageModeBufferOverwrite(boolean), emulate(String)GlinkEvent_MESSAGE_MODE_DATA, emulate(String)

messageModeReceiveBytesSync

byte[ ] messageModeReceiveBytesSync(int maxlen)

This method waits until the requested number of characters has been received from the host before it returns the byte array of characters received, in the host native character set.

The method will check the turn first and if the turn is at the host side, it will wait until the requested number of characters has been received from the host and return these as a byte array. If less characters have been received when turn is received, then these characters are returned. When no more data is available, a null pointer is returned.

If the message mode receive buffer has reached its limit before the number of requested characters has been received, the message mode receive buffer is automatically adjusted.

Parameters:

maxlen - Maximum of incoming characters to wait.

Returns:
array of host data bytes, or null if no data is available

See also:
messageModeReceive(),messageModeReceiveBytes(), messageModeReceiveSync(), messageModeBufferOverwrite(boolean), emulate(String)GlinkEvent_MESSAGE_MODE_DATA, messageModeOn(int, boolean),isTurnKnown(), isTurnReceived()

messageModeReceiveSync

String messageModeReceiveSync(int maxlen)

This method waits until the requested number of characters has been received from the host before it returns these.

The method will check the turn first and if the turn is at the host side, it will wait until the requested number of characters has been received from the host and return these as a string. If less characters have been received when turn is received, then these characters are returned. When no more data is available, a null pointer is returned.

If the message mode receive buffer has reached its limit before the number of requested characters has been received, the message mode receive buffer is automatically adjusted.

Parameters:

maxlen - Maximum of incoming characters to wait.

Returns:
The string of character received or null if no data is available

See also:
messageModeReceive(),messageModeReceiveBytes(), messageModeReceiveBytesSync(), messageModeBufferOverwrite(boolean), emulate(String)GlinkEvent_MESSAGE_MODE_DATA, messageModeOn(int, boolean),isTurnKnown(), isTurnReceived()

messageModeSend

void messageModeSend(String s, boolean passTurn)

This method bypasses the Glink emulation and sends the string directly to the host. Several strings of data may be sent without giving up the turn. The passTurn flag is used to signal that this is the end of the data and that the turn should be passed to the host side.

The data is assumed to be an ASCII/UNICODE string. It is translated to the host native character set before transmission.

Turn-issues:
Please note that the concept of **turn** is only well defined for the following communications protocols:

G&R Ggate
G&R DGA
Bull GTEA
TNVIP
TN3270
TN5250

For other communications protocols (such as Telnet, raw TCP/IP and Minitel) the **passTurn** argument is ignored. In these cases you may have to add a go-ahead indicator to your data if you want to pass the turn to the host, such as ETX (0x03) or Carriage Return (0x0D).

Parameters:

s - String of characters to sent to the host.

passTurn - Send the data to the host with the turn.

See also:
messageModeSendBytes()

messageModeSendBytes

void messageModeSendBytes(byte[ ] s, int offset, int len, boolean passTurn)

This method bypasses the Glink emulation and sends a byte array directly to the host. Several data buffers may be sent without giving up the turn. The passTurn flag is used to signal that this is the end of the data and that the turn should be passed to the host side.

The data is assumed to be in the host native character set.

Turn-issues:
Please note that the concept of **turn** is only well defined for the following communications protocols:

G&R Ggate
G&R DGA
Bull GTEA
TNVIP
TN3270
TN5250

For other communications protocols (such as Telnet, raw TCP/IP and Minitel) the **passTurn** argument is ignored. In these cases you may have to add a go-ahead indicator to your data if you want to pass the turn to the host, such as ETX (0x03) or Carriage Return (0x0D).

Parameters:

s - String of characters to sent to the host.

offset - starting index to send from.

len - number of bytes to send.

passTurn - Send the data to the host with the turn.

See also:
messageModeSend()

noScreen

void noScreen()

This method instructs Glink to start without the Glink screen display class. If a number of Glink sessions are started without using the Glink display at all, performance will be improved by setting this parameter. Please note that if you just want to hide the screen display, use the ***setVisible*** method instead to switch the Glink display on and off.

notifyCommandKey

void notifyCommandKey(int commandKey)

Instructs Glink to fire a ***GlinkEvent_COMMAND_KEY_TYPED*** event when the given command key is typed. This command key will then be ignored by Glink. For the command key to be processed by Glink, send the command key back to Glink with the ***sendCommandKey*** method when notified.

Several command keys may be marked for notification. To be notified for all command keys, set the commandKey parameter to -1.

Parameters:

commandKey - The command key for which notifications are to be sent.

See also:
GlinkKey, removeNotifyCommandKey(int),
GlinkEvent_COMMAND_KEY_TYPED

notifyKey

void notifyKey(int key)

Instructs Glink to fire a ***GlinkEvent.KEY_TYPED*** event when the given key is typed. This key will then be ignored by Glink. For the key to be processed by Glink, send the key back to Glink with the ***sendKeys*** method when the notification is received.

Several keys may be marked for notification. To receive notification for every key typed, set the commandKey parameter to -1.

Parameters:

key - The key to be notified.

See also:
removeNotifyKey(int), GlinkEvent.KEY_TYPED

notifyScreenArea

void notifyScreenArea(GlinkScreenArea screenArea, int identity, boolean removeAfterwards)

Instructs Glink to fire a ***GlinkEvent.SCREEN_AREA_MATCH*** event when the given object matches the current screen. Several objects may be waited for, each with its own identity value. The objects are checked each time Glink receives the turn. For some host connections turn signals are not generated. Use the ***isTurnKnown()*** method to check if the current connection gives turn signals.

Parameters:

screenArea - The object that describes the screen area to be checked. When the object matches the current screen the caller will be notified.

identity - Select a positive number to identify this object when being reported. This value will be supplied with the ***GlinkEvent*** object (***getValue*** method) to identify which ***screenArea*** object matches the current screen. If several objects match the current screen, a notification will be given for each of them.

removeAfterwards - Specifies whether the object should be removed from the notification list or not after the notification has been given for the object. For peformance reasons remove objects that are not in use anymore.

See also:
removeNotifyScreenArea(int), isTurnKnown(), GlinkEvent.getValue()

notifyString

void notifyString(String string, boolean caseSensitive, int identity, boolean removeAfterwards)

Instructs Glink to fire a ***GlinkEvent_STRING_RECEIVED*** event when the given string is received from host. Several strings may be waited for, each with its own identity value. The check is performed before the received data is processed by the emulation module of Glink. Use the ***notifyScreenArea*** method instead if you want the check to be performed after the received data has been processed by Glink.

Parameters:

string - The string for which a notification is to be sent.

caseSensitive - Specifies if the match is to be case sensitive.

identity - Select a positive number to identify this string. This value will be supplied with the ***GlinkEvent*** object (***getValue*** method) to identify which string the notification is for.

removeAfterwards - Specifies whether the string should be removed from the notification list after the string has been received from host and the notification has been sent. The string may also be removed from the list with the ***removeNotifyString*** call.

See also:
removeNotifyString(int), GlinkEvent.getValue(), notifyScreenArea(GlinkScreenArea, int, boolean)

noToolbar

void noToolbar()

This method instructs Glink to start without a toolbar and keybar. Also, when this option is set, no images are loaded at all to save memory and reduce startup time.

queueGlinkEvents

void queueGlinkEvents (boolean enable)

Enables or disables the queuing of GlinkEvents. GlinkEvents queuing can be used instead of a GlinkEvent listener to get hold of GlinkEvents. Use the getGlinkEvent method to get hold of GlinkEvents queued.

Restricted

See also:
getGlinkEvent(int, long), flushGlinkEvent(), GlinkEvent
addGlinkEventListener(GlinkEventListener)

removeConfiguration

void removeConfiguration(GlinkConfiguration glinkConfiguration)

Removes the configuration previously set with the ***addConfiguration*** method.

Restricted

See also:
addConfiguration(GlinkConfiguration)

removeGlinkEventListener

void removeGlinkEventListener(GlinkEventListener listener)

Removes the specified listener set with the ***addGlinkEventListener*** method.

See also:
addGlinkEventListener(GlinkEventListener)

removeNotifyCommandKey

void removeNotifyCommandKey(int commandKey)

Removes the notification for the command key previously set with the ***notifyCommandKey*** method. If the ***commandKey*** param is -1, all command key notifications are removed.

See also:
notifyCommandKey(int), GlinkKey

removeNotifyKey

void removeNotifyKey(int key)

Removes the notification for the key previously set with the ***notifyKey*** method. If the ***key*** param is -1, all key notifications are removed.

See also:
notifyKey(int)

removeNotifyScreenArea

boolean removeNotifyScreenArea(int identity)

Removes the ***GlinkScreenArea*** object associated with the given identity from the list of objects being checked for a match. If you only want to be notified the first time the object matches the current screen, you may set the "***removeAfterwards***" parameter in the ***notifyScreenArea*** call itself rather than using this call.

Parameters:

identity - Identifies the ***GlinkScreenArea*** object to be removed. To remove all of the ***GlinkScreenArea*** notification objects set, supply -1 for the identity.

Returns:
True if the object with the given identity is found, otherwise false.

See also:
notifyScreenArea(GlinkScreenArea, int, boolean)

removeNotifyString

boolean removeNotifyString(int identity)

Removes the string associated with the given identity from the list of strings being checked for. If you only want to be notified the first time the string is received from the host, you may set the "***removeAfterwards***" parameter in the ***notifyString*** call itself rather than using this call.

Parameters:

identity - Identifies the string to be removed. To remove all of the string notifications set, supply -1 for the identity.

Returns:
True if the string with the given identity is found, otherwise false.

See also:
notifyString(String, boolean, int, boolean)

restoreScreenContent

boolean restoreScreenContent (String fileName)

Restores the screen previously saved. The operation will fail if the given file does not exists, the file has not been created by the ***saveContentScreen*** method or the current emulation type does not match the saved one.

Returns:
If the operation fails, a ***GlinkEvent_ERROR_DETECTED*** will be posted with the error type code ***GlinkEvent_VE_EMULATION_ERROR*** and additional error type text.

See also:
saveScreenContent(String)

saveScreenContent

boolean saveScreenContent (String fileName)

Restores the screen previously saved. The operation will fail if the given file does not exists, the file has not been created by the ***saveContentScreen*** method or the current emulation type does not match the saved one.

Returns:
If the operation fails, a ***GlinkEvent_ERROR_DETECTED*** will be posted with the error type code ***GlinkEvent_VE_EMULATION_ERROR*** and additional error type text.

See also:
restoreScreenContent(String)

scriptCommand

void scriptCommand(String command)

Activates the given script command.

scriptFile

void scriptFile(String fileName)

Activates the given script file.

Parameters:

fileName - If the script is stored with the Glink Config Server specify: "server://filename" otherwise supply a full path name.

scriptTerminate

void scriptTerminate()

Terminates the current script.

sendCommandKey

void sendCommandKey(int key)

The ***sendCommandKey*** method sends a "command" keystroke to the virtual screen. These command keys can be thought of as special keystrokes, like the Enter key, the Tab key or the Page Up key. All valid special key values are contained in the GlinkKey class.

See also:
GlinkKey

sendKeys

void sendKeys(String text)

Sends a string to the emulator starting at the current cursor location. The ***sendKeys*** method sends a string of keys to the virtual screen. This method acts as though keystrokes were being typed from the keyboard.

See also:
setString(String, GlinkPoint)

sendKeys

void sendKeys(String text, GlinkPoint location)

Sends a string to the emulator starting at the given cursor location. The ***sendKeys*** method sends a string of keys to the virtual screen. This method acts as though keystrokes were being typed from the keyboard.

See also:
setString(String text)

serverPort

void serverPort(String port)

Set the IP port on which to connect to the Glink server.

Restricted; Java only

Parameters:

port - decimal port number.

sessionName

boolean sessionName(String name)

Specifies the name of the session to which Glink should connect. If the session is not specified, Glink will bring up the Glink Session dialog box for the user to make a selection.

Restricted

Parameters:

name - name of configuration profile.

Returns:
true if session name was found, otherwise false.

setCursor

void setCursor(GlinkPoint location)

Moves the cursor to the given position.

See also:
getCursor()

setCursor

void setCursor(int x, int y)

Moves the cursor to the given position.

See also:
getCursor()

setGlinkDirectory

void setGlinkDirectory(String glinkDir)

Set the Glink directory.

Restricted; Java only

Parameters:

glinkDir - Glink directory. glink.ini.glinkdat (formerly glink.ini) should be stored here.

setInteractiveComParams

void setInteractiveComParams (Collection params)

Sets the interactive communication parameters. This method should be used only if the following event is posted:

GlinkEvent_INTERACTIVE_COM_PARAMS_REQUESTED

The setInteractiveComParams must be used together with the getInteractiveComParams, see the getInteractiveComParams for the description on how to use these calls.

Parameters:

params - The Collection containing the interactive communication parameters. The Collection should have been initialized with the getInteractiveComParams method.

See also:
getInteractiveComParams(),
GlinkEvent_INTERACTIVE_COM_PARAMS_REQUESTED

setRestrictedMode

boolean setRestrictedMode(String key, int level)

Switch restricted mode on or off. In Restricted mode, methods marked as Restricted will silently do nothing. A key is required to switch restricted mode on, and the same key must be provided to exit restricted mode.

Parameters:

key - this key is memorized when switching restricted mode on, and must match to exit restricted mode.

level - switch restricted mode off if 0, on if != 0.

Returns:
true if operation succeeded, otherwise false.

setScreenSize

void setScreenSize(int columns, int rows)

For most emulations the screen size is 80 columns times 24 rows. However some emulation modes may use a different size. For Glink to be able to switch between most of the different screen sizes, the default screen size is set to 132 columns times 32 rows, which means that the actual screen size may be less than or equal to this size. For larger screens to be suported you must increase the size. On the other hand, if you plan to run a number of Glink sessions simultaneously and you know the screen size is less than the default, use this method to reduce the memory allocated. The method must be called before the starting the initial Glink session.

setString

void setString(String text, GlinkPoint location)

The ***setString*** method sends a string to the virtual screen at the specified location. The string will only overwrite unprotected fields, and any parts of the string which fall over protected fields will be discarded.

Parameters:

text - String to place in the virtual screen.

location - Position where the string should be written.

See also:
sendKeys(String)

setSplashScreenVisible

void setSplashScreenVisible(boolean visible)

Specifies whether the splash screen should be visible.

Java only

setVisible

void setVisible(boolean visible)

Specifies whether Glink should be visible.

start

void start()

Starts Glink. The ***GlinkEvent_STARTED*** event code is posted when Glink is started.

Restricted

stop

void stop()

Stops Glink. The ***GlinkEvent.STOPPED*** event code is posted when Glink is stopped.

Restricted

traceMethods

void traceMethods(boolean trace)

Instructs the API to log the methods being called.

Java only

See also:
getTracedMethod()

translateBuffer

String translateBuffer(String buffer, int direction)

Translate a string between ASCII and EBCDIC. This is a utility method for general-purpose use.

Parameters:

buffer - String to translate.

direction - how to translate.

Returns:
translated string.

See also:
ASCII_TO_EBCDIC, EBCDIC_TO_ASCII

userName

void userName(String name)

Specifies the user name. If the name is not specified or is not valid, Glink will bring up the Glink Logon dialog box.

Note that this user name is only used to access the Glink configuration database and is not used to log on to the host.

Restricted; Java only

Parameters:

name - user's name.

See also:
userPassword*(String)*

userPassword

void userPassword(String password)

Specifies the password for the user.

Note that this password is only used to access the Glink configuration database and is not used to log on to the host. If the password is not specified or is not valid, Glink will bring up the Glink Logon dialog box.

COM+ reference

Restricted; Java only

Parameters:

password - user's password.

See also:
userName*(String)*

GlinkConfiguration class

Class GlinkConfiguration

GlinkApi
↳GlinkConfiguration
class GlinkConfiguration

This class enables you to create a configuration used to set the necessary parameters for connecting to a given host application. Normally the administration program is used to create and manage various configurations (session descriptions). But configurations can also be created on the fly with this class. They are made available to GlinkApi objects with its ***addConfiguration*** method.

A "configuration" is a session description with enough information for Glink to be able to connect to a given host application. In other words there is one "configuration" for each host application made available to the users.

Each configuration is given a name, normally referred to as the "session name". When starting Glink, the user is presented a list with session names. When the user selects an entry from this list, he selects a configuration with that name.

From a GlinkApi object a configuration (session) is selected with the ***GlinkApi.sessionName()*** method.

The parameters in a configuration are grouped into six different types of profiles: host connection, emulation, screen display, keyboard layout, printer configuration, and toolbar setup. Each profile has a name. All the profiles in a ***GlinkConfiguration*** object are given the same name as the object itself and all the parameters are given default values. Sometimes it may be convenient to instruct the ***GlinkConfiguration*** object to use an existing profile created by the Glink Administration utility than changing the default value for a number paramters. This is done by the ***setProfileName*** method. The ***GlinkConfiguration*** object will then use this profile with its parameters instead of the default parameters.

The default value for any configuration parameter can be changed with the ***setParameter*** method regardless of which profile the parameter belongs to.

See also:
GlinkApi.addConfiguration(GlinkConfiguration),
GlinkApi.sessionName(String)

Field Summary

static int GlinkConfiguration_EMULATION_3270
static int GlinkConfiguration_EMULATION_5250
static int GlinkConfiguration_EMULATION_ANSI
static int GlinkConfiguration_EMULATION_DKU
static int GlinkConfiguration_EMULATION_MINITEL
static int GlinkConfiguration_EMULATION_TTY
static int GlinkConfiguration_EMULATION_VIEWDATA
static int GlinkConfiguration_EMULATION_VIP7700
static int GlinkConfiguration_EMULATION_VIP7801
static int GlinkConfiguration_EMULATION_VIP7804
static int GlinkConfiguration_EMULATION_VT
static string GlinkConfiguration_OVERRIDE_PARAMS
static int GlinkConfiguration_PROFILE_EMULATION
static int GlinkConfiguration_PROFILE_HOST
static int GlinkConfiguration_PROFILE_KEYBOARD
static int GlinkConfiguration_PROFILE_PRINTER
static int GlinkConfiguration_PROFILE_SCREEN
static int GlinkConfiguration_PROFILE_TOOLBAR
static int GlinkConfiguration_PROTOCOL_DSA
static int GlinkConfiguration_PROTOCOL_GGATE_DIWS
static int GlinkConfiguration_PROTOCOL_GGATE_DSA
static int GlinkConfiguration_PROTOCOL_GGATE_DSA_CXI
static int GlinkConfiguration_PROTOCOL_GTEA
static int GlinkConfiguration_PROTOCOL_MINITEL
static int GlinkConfiguration_PROTOCOL_TCP_RAW
static int GlinkConfiguration_PROTOCOL_TELNET
static int GlinkConfiguration_PROTOCOL_TELNET3270
static int GlinkConfiguration_PROTOCOL_TELNET5250
static int GlinkConfiguration_PROTOCOL_TNVIP

Constructor Summary

GlinkConfiguration(String name, int emulation, int protocol, String address)

This constructor creates a configuration with the given emulation and host protocol and the address to the host. If the name points to a valid Glink configuration file then it will be used for the default values. Any subsequent ***setParameter()*** calls will override these options.

Method Summary

String getName()

Returns the name of the configuration.

String getProfileName(int type)

Returns the name of the profile with the given type. This method has no effect in the Windows version.

void setParameter(String parameter)

Sets the specific parameter with the given value.

void setProfileName(int type, String name)

Sets the profile of the given type to be used with this configuration. This method has no effect in the Windows version.

Field Detail

GlinkConfiguration_EMULATION_3270

static int GlinkConfiguration_EMULATION_3270

GlinkConfiguration_EMULATION_5250

static int GlinkConfiguration_EMULATION_5250

GlinkConfiguration_EMULATION_ANSI

static int GlinkConfiguration_EMULATION_ANSI

GlinkConfiguration_EMULATION_DKU

static int GlinkConfiguration_EMULATION_DKU

GlinkConfiguration_EMULATION_MINITEL

static int GlinkConfiguration_EMULATION_MINITEL

GlinkConfiguration_EMULATION_TTY

static int GlinkConfiguration_EMULATION_TTY

GlinkConfiguration_EMULATION_VIEWDATA

static int GlinkConfiguration_EMULATION_VIEWDATA

GlinkConfiguration_EMULATION_VIP7700

static int GlinkConfiguration_EMULATION_VIP7700

GlinkConfiguration_EMULATION_VIP7801

static int GlinkConfiguration_EMULATION_VIP7801

GlinkConfiguration_EMULATION_VIP7804

static int GlinkConfiguration_EMULATION_VIP7804

GlinkConfiguration_EMULATION_VT

static int GlinkConfiguration_EMULATION_VT

GlinkConfiguration_OVERRIDE_PARAMS

static string GlinkConfiguration_OVERRIDE_PARAMS

Special GlinkConfiguration name used as a generic parameter override. If you add a GlinkConfiguration with this name to the API object, the parameters it contains will always override the corresponding parameters loaded from the standard Glink configuration database, regardless of the session name in use. It is exclusive in the sense that if you add it, do not add other GlinkConfiguration objects to the same API object, or else results will be unpredictable.

GlinkConfiguration_PROFILE_EMULATION

static int GlinkConfiguration_PROFILE_EMULATION

GlinkConfiguration_PROFILE_HOST

static int GlinkConfiguration_PROFILE_HOST

GlinkConfiguration_PROFILE_KEYBOARD

static int GlinkConfiguration_PROFILE_KEYBOARD

GlinkConfiguration_PROFILE_PRINTER

static int GlinkConfiguration_PROFILE_PRINTER

GlinkConfiguration_PROFILE_SCREEN

static int GlinkConfiguration_PROFILE_SCREEN

GlinkConfiguration_PROFILE_TOOLBAR

static int GlinkConfiguration_PROFILE_TOOLBAR

GlinkConfiguration_PROTOCOL_DSA

static int GlinkConfiguration_PROTOCOL_DSA

GlinkConfiguration_PROTOCOL_GGATE_DIWS

static int GlinkConfiguration_PROTOCOL_GGATE_DIWS

GlinkConfiguration_PROTOCOL_GGATE_DSA

static int GlinkConfiguration_PROTOCOL_GGATE_DSA

GlinkConfiguration_PROTOCOL_GGATE_DSA_CXI

static int GlinkConfiguration_PROTOCOL_GGATE_DSA_CXI

GlinkConfiguration_PROTOCOL_GTEA

static int GlinkConfiguration_PROTOCOL_GTEA

GlinkConfiguration_PROTOCOL_MINITEL

static int GlinkConfiguration_PROTOCOL_MINITEL

GlinkConfiguration_PROTOCOL_TCP_RAW

static int GlinkConfiguration_PROTOCOL_TCP_RAW

GlinkConfiguration_PROTOCOL_TELNET

static int GlinkConfiguration_PROTOCOL_TELNET

GlinkConfiguration_PROTOCOL_TELNET3270

static int GlinkConfiguration_PROTOCOL_TELNET3270

GlinkConfiguration_PROTOCOL_TELNET5250

static int GlinkConfiguration_PROTOCOL_TELNET5250

GlinkConfiguration_PROTOCOL_TNVIP

static int GlinkConfiguration_PROTOCOL_TNVIP

Constructor Detail

GlinkConfiguration

GlinkConfiguration(String name, int emulation, int protocol, String address)

This constructor creates a configuration with the given emulation and host protocol and the address to the host. Normally this configuration can be used asis, no extra configuration is needed.

See also:
setProfileName, setParameter

Method Detail

getName

String getName()

Returns the name of the configuration.

getProfileName

String getProfileName(int type)

Returns the name of the profile with the given type. Normally this name will be the same as the name of the configuration itself.

setParameter

void setParameter(String parameter)

Sets the specific parameter with the given value. For example; ***setParameter("screen.fontsize=16")*** or ***setParameter("emu.vipinittext=false")***.

There are two groups of parameters available for use with the API:

- comms parameters, prefixed with com.
- emulation parameters, prefixed with emu.

Within each group there are several sets of parameters:

- generic parameters that apply to all emulations and comms protocols
- emulation and protocol specific parameters

Generic protocol parameters

Name	Type	Description
com.parity	Integer	parity of the line. Values are 0=7 bits even, 1=8 bits none, 2=7 bits odd, 3=8 bits even, 4=8 bits odd.
com.strippar	Boolean	if true, parity is stripped.
com.terminateondisconnect	Boolean	if true, terminates Glink when disconnected from host.
com.servertarget	String	Ggate CoName/TNVIP resource

		name.

G&R Ggate protocol parameters.

See Ggate documentation for details.

Name	Type	Description
com.ggateda	String	Host application mailbox name
com.ggatedn	String	Host node name
com.ggatedu	String	User ID
com.ggatepw	String	Password
com.ggatedp	String	Project
com.ggatedb	String	Billing
com.ggatehm	String	Ggate host mode
com.ggatetm	String	Ggate terminal mode
com.ggateur	String	GRTS/LID user string
com.ggatedx	String	Mailbox extension
com.ggatemn	String	Local mailbox name
com.ggatelm	String	IBM log mode
com.ggateextra	String	Additional Ggate parameters
com.ggatekeepaliveint	Integer	Keepalive interval, in seconds

Generic Telnet parameters.

Name	Type	Description
com.telnetcrnul	Boolean	if true, replace CR LF pairs with CR NUL
com.telnetuseip	Boolean	if true, map Break to the IP command
com.telnetbinary	Boolean	if true, sets the Telnet session to binary mode
com.telnetsimulateparity	Boolean	if true, simulate parity in software

TN3270 parameters.

Name	Type	Description
com.tn3270luname	String	Logical Unit name
com.tn3270extended	Boolean	if true, use extended TN3270 (TN3270E)
com.tn3270associatelu	Boolean	if true, this is a printer session associated with the LU of the terminal

TN5250 parameters.

Name	Type	Description
com.tn5250maindevice	String	Primary 5250 device name, as specified in the "5250 Telnet Enhancements" IETF draft
com.tn5250altdevice	String	Alternate 5250 device name, as specified in the "5250 Telnet Enhancements" IETF draft
com.tn5250usedevname	Boolean	if true, send device name as specified in the "5250 Telnet Enhancements" IETF draft

Generic emulation parameters

Name	Type	Description
emu.autotabl	Boolean	switch local autotab on or off
emu.autotabh	Boolean	switch host autotab on or off
emu.destructbs	Boolean	if true, backspaces are destructive
emu.nostatusline	Boolean	if true, the status line is hidden
emu.host8bit	Boolean	true for 8 bit host connections
emu.xlitfile7, emu.xlitfile8	String	Transliteration file for 7 and 8 bit connections. Values are 437, 865, dp7, fin, fra, frc, ger, ita, kpc, mac, nor, pnc, sf, spa, swe, swi, t43, t85, t86, uk, us. See online help for further information.
emu.usesiso	Boolean	if true, use SI/SO for printing
emu.allowlcase	Boolean	if true, send lowercase to host asis

Locally stored forms parameters

Name	Type	Description
emu.formsenable	Boolean	if true, locally stored forms are enabled in VIP mode
emu.tcsenable	Integer	locally stored forms support in DKU mode. Values are 0=inactive, 1=enabled, 2=ignored.
emu.formsdir, emu.formsdir2	String	names of directories for storing primary and backup forms. Must

		conform to the file system syntax of the platform.
emu.etxsend	Boolean	if true, send ETX using ETX character (0x03), otherwise use EOT (0x04)
emu.ftcsack, emu.ftcspgof, emu.ftcsnak	String	Ack, Page Overflow and Nak responses in TCS forms mode. Should be 'a', 'b' and 'c' respectively.

DKU emulation parameters

Name	Type	Description
emu.dkublinkav	Boolean	if true, sets blink/blank attribute on reception of caret/tilde
emu.dkublinkshow	Boolean	if true, blink/blank characters are shown
emu.dkucud	Boolean	if true, allows cursor straight up/down using arrow keys
emu.dkucuf	Boolean	if true, allow cursor out-of-field
emu.dkunewline	Boolean	if true, send Newline when Transmit key is pressed
emu.dkusdpatt	Boolean	if true, enables SDP attribute mode
emu.dkuwrap	Boolean	if true, page overflow wraps to top of screen
emu.dkuwraptab	Boolean	if true, tabbing off the end of a form wraps to the beginning
emu.dkuinitroll	Boolean	if true, start in Roll mode
emu.dkutermid	Integer	Terminal type, 0=7107, 1=7211
emu.dkucolor	Integer	Color mode, values are 0=1M, 1=4A, 2=4B, 3=7Q, 4=7G
emu.dkuprtid	String	Printer ID returned on device attribute enquiry
emu.dkuprtnbc	Integer	Number of print columns returned on device attribute enquiry
emu.dkuprtnbl	Integer	Printer lines-per-page returned on device attribute enquiry
emu.dkuprtcps	Integer	Printer CPS rate returned on device attribute enquiry

VIP emulation parameters

Name	Type	Description
emu.vipinittext	Boolean	if true, start in Text mode
emu.vipenqstring	String	VIP enquiry reply string
emu.vipextstatus	Boolean	if true, enable extended status on enquiry
emu.vipscrewyforms	Boolean	if true, non-linear forms mode is enabled
emu.vipstickyroll	Boolean	if true, preserve Roll Mode setting across forms
emu.vipaddlfs	Boolean	if true, add LF to CR (7700)
emu.vipnostatusline	Boolean	if true, suppress status line
emu.viptxretmode	Boolean	if true, TX on return mode is enabled
emu.vipinitecho	Boolean	if true, start in Echo mode
emu.vipinitroll	Boolean	if true, start in Roll mode
emu.vipinitblock	Boolean	if true, start in Block mode
emu.vipnoprintff	Boolean	if true, suppress Form Feeds in print output
emu.vipsuppresslock	Boolean	if true, suppresses status line lock
emu.vipspacesuppress	Boolean	if true, suppress trailing spaces to host
emu.vipaddcrlfs	Boolean	if true, add CRLF delimiters in SSM mode
emu.vipnlafterxmit	Boolean	if true, send NL on Transmit key
emu.vipsuppressapcmd	Boolean	if true, suppress APR commands from host
emu.vipv7200a	Integer	7200 attribute mode, 0=none, 1=enabled, 2=extended
emu.vipfkeyswithcr	Boolean	if true, sends CR instead of ETX with function keys
emu.vip72linemode	Boolean	if true, enable 72 line mode
emu.vip77spacereplace	Boolean	if true, replace spaces with dots in 7700 mode
emu.vipcompat	Integer	Vip compatibility, 0=Normal, 1=ITT Courier, 2=Thomas box

VT/ANSI emulation parameters

Name	Type	Description

emu.vtaddlfs	Boolean	if true, add LF to CR
emu.vtinitecho	Boolean	if true, start in Echo mode
emu.vtinitroll	Boolean	if true, start in Roll mode
emu.vtno81	Boolean	if true, suppress column 81
emu.vtanswerb	String	ANSI answerback string

IBM 3270 emulation parameters

Name	Type	Description
emu.ibm3270nxlit	Boolean	if true, use Norwegian ASCII translation
emu.ibm3270noprint	Boolean	if true, host print is disabled
emu.ibm3270model	Integer	3270 model. Values are: 0=3279-2, 1=3279-3, 2=3278-1, 3=3278-2, 4=3278-3, 5=3278-4, 6=3278-5, 7=3287-1 (printer), 8=3279-2E, 9=3279-3E, 10=3278-1E, 11=3278-2E, 12=3278-3E, 13=3278-4E, 14=3278-5E
emu.ibm3270xlit	Integer	Language to use for EBCDIC/ASCII transliteration. Values are: 0=International, 1=UK English, 2=US English, 3=Finnish/Swedish, 4=French, 5=French Canadian, 6=Austrian/German, 7=Italian, 8=Danish/Norwegian, 9=Spanish, 10=Finnish/Swedish (alt), 11=Belgian, 12=Danish/Norwegian (alt), 13=Japanese English, 14=Brazilian, 15=Portuguese, 16=Spanish (alt), 17=Spanish-speaking, 18=Austrian/German (alt), 19=Icelandic

IBM 5250 emulation parameters

Name	Type	Description
emu.ibm5250nxlit	Boolean	if true, use Norwegian ASCII transliteration
emu.ibm5250model	Integer	5250 model. Values are: 0=3179-2, 1=3180-2, 2=3196-A1, 3=3477-

		FC, 4=3477_FG, 5=5251_11, 6=5291_1, 7=5292_2, 8=5555_B01, 9=5555_C01, 10=3812_1, 11=5553_B01
emu.ibm5250xlit	Integer	Language to use for EBCDIC/ASCII transliteration. Values are: 0=International, 1=UK English, 2=US English, 3=Finnish/Swedish, 4=French, 5=French Canadian, 6=Austrian/German, 7=Italian, 8=Danish/Norwegian, 9=Spanish, 10=Finnish/Swedish (alt), 11=Belgian, 12=Danish/Norwegian (alt), 13=Japanese English, 4=Brazilian, 15=Portuguese, 16=Spanish (alt), 17=Spanish-speaking, 18=Austrian/German (alt), 19=Icelandic

Extended 5250 parameters

The parameters below implement the variable/user variable scheme defined in the IETF draft draft-ietf-tn3270e-tn5250e-05 : "5250 Telnet Enhancements". Refer to that document for an explanation of these parameters. Each parameter comes in a string/boolean pair: the parameter itself, as a string, and a boolean activation parameter. If the activation parameter is set to true, the corresponding parameter will be sent to the TN5250 server on request.

The IETF draft can be found here.

Parameter	Activated by
emu.ibm5250user	***emu.ibm5250sendusr***
emu.ibm5250keyboard	***emu.ibm5250sendkb***
emu.ibm5250codepage	***emu.ibm5250sendcp***
emu.ibm5250charset	***emu.ibm5250sendcs***
emu.ibm5250msgqname	***emu.ibm5250sendmsgq***
emu.ibm5250formfeed	***emu.ibm5250sendff***
emu.ibm5250transform	***emu.ibm5250sendxform***
emu.ibm5250printermodel	***emu.ibm5250sendmdl***
emu.ibm5250papersource1	***emu.ibm5250sendps1***
emu.ibm5250papersource2	***emu.ibm5250sendps2***

emu.ibm5250envelope	***emu.ibm5250sendenv***
emu.ibm5250wscsctname	***emu.ibm5250sendwsname***
emu.ibm5250wsctlib	***emu.ibm5250sendwslib***

Print parameters

Name	Type	Description
print.type	String	Type of printer. Values are Local, LPD, File, GlinkApi, None.

Parameters:

parameter - The string contains both the name of the parameter and its value.

setProfileName

void setProfileName(int type, String name)

Sets the profile of the given type to be used with this configuration. Sometimes it may be more convenient to refer to a profile already created by the Glink Administration utility than setting a number of parameters. The GlinkConfiguration object will then use this profile if found instead of the default one.

Parameters:

type - The profile type, for example GlinkConfiguration_PROFILE_HOST or GlinkConfiguration_PROFILE_SCREEN.

name - The name of an existing profile with the given type.

GlinkEvent class

GlinkApi
↳GlinkEvent
class GlinkEvent

Identifies the type of events that a ***GlinkEventListener*** may receive. A ***GlinkEvent*** gets delivered whenever a Glink event has occured. The ***eventCode*** contains the event type and the source of the GlinkApi object that generated the event.

See also:
GlinkEventListener

Field Summary

static int GlinkEvent_COMMAND_KEY_TYPED

A command key has been typed.

static int GlinkEvent_CONNECTED

static int GlinkEvent_DISCONNECTED

static int GlinkEvent_ERROR_DETECTED

The ***getValue*** method returns the error code.

static int GlinkEvent_INTERACTIVE_COM_PARAMS_REQUESTED

Notifies that the Glink communication module needs additional parameters.

static int GlinkEvent_KEY_TYPED

A key has been typed.

static int GlinkEvent_MESSAGE_MODE_DATA

In message mode operation, notifies that data received from host is available and may be picked up with the ***messageModeReceive*** method.

static int GlinkEvent_PRINT_DATA

Notifies that print data is available.

static int GlinkEvent_PRINT_STARTED

Notifies that print has started but not available yet.

static int GlinkEvent_SCREEN_AREA_MATCH

The ***getValue*** method returns the GlinkScreenArea identity value.

static int GlinkEvent_STARTED

static int GlinkEvent_STOPPED

static int GlinkEvent_STRING_RECEIVED

The ***getValue*** method returns the string identity value.

static int GlinkEvent_TURN_LOST

Notifies that this side does not have the turn (permission) to send data and should instead expect to receive data from the other side (host).

static int GlinkEvent_TURN_RECEIVED

Notifies that this side have the turn (permission) to send data.

static int GlinkEvent_VE_CONFIG_ERROR

Error type: An error with the configuration is detected.

static int GlinkEvent_VE_CONNECTION_ERROR

Error type: An error with the connection is detected.

static int GlinkEvent_VE_EMULATION_ERROR

Error type: An error with the emulation is detected.

static int GlinkEvent_VE_LOGON_FAILED

Error type: User name or user password is missing or wrong.

static int GlinkEvent_VE_NO_API_LICENSE

Error type: License key allowing use of the API not found.

static int GlinkEvent_VE_NO_CNX_LICENSE

Error type: License key allowing use of connector not found.

static int GlinkEvent_VE_OUT_OF_MEMORY

Error type: The Java VM reported out of memory when loading Glink.

static int GlinkEvent_VE_SESSIONS_ERROR

Error type: The ***GlinkApi.getAvailableSessions*** failed.

static int GlinkEvent_VE_STARTUP_ERROR

Error type: Glink failed to start.

static int GlinkEvent_VE_TOOMANY_INSTANCES

Error type: Too many simultaneous instances for Pro version.

Method Summary

int getEventCode()

The application should check the Glink event code to determine which Glink event did occur, for example that the Glink session was connected or disconnected.

GlinkApi getSource()

The method returns the GlinkApi object that generated the event.

int getValue()

Some Glink events have a value set that gives more information about the given event.

String getValueText()

Some Glink events have additional text information about the given event.

Field Detail

GlinkEvent_COMMAND_KEY_TYPED

static int GlinkEvent_COMMAND_KEY_TYPED

A command key has been typed. The ***getValue*** method returns the value of the command key.

See also:
GlinkApi.notifyCommandKey(int), GlinkKey, getValue()

GlinkEvent_CONNECTED

static int GlinkEvent_CONNECTED

GlinkEvent_DISCONNECTED

static int GlinkEvent_DISCONNECTED

GlinkEvent_ERROR_DETECTED

static int GlinkEvent_ERROR_DETECTED

The ***getValue*** method returns the error code. See the ***GlinkEvent_VE_xxx*** values for the type of error.

See also:
getValue()

GlinkEvent_INTERACTIVE_COM_PARAMS_REQUESTED

static int GlinkEvent_INTERACTIVE_COM_PARAMS_REQUESTED

Notifies that the Glink communication module needs additional parameters. The Ggate and DSA communication modules may be configured with parameters that should be supplied or modified by the user when connecting to the host application. These parameters are referred to as interactive communication parameters.

If the ***GlinkEvent_INTERACTIVE_COM_PARAMS_REQUESTED*** event is posted, then the GlinkApi application must at least do the following GlinkApi method calls:

glink.setInteractiveComParams(glink.getInteractiveComParams());

for Glink to continue the connecting process.

See also:
GlinkApi.getInteractiveComParams()
GlinkApi.setInteractiveComParams(Collection)

GlinkEvent_KEY_TYPED

static int GlinkEvent_KEY_TYPED

A key has been typed. The ***getValue*** method returns the value of the key as an integer. The ***getValueText*** method returns the value of the key as a one character long string.

See also:
GlinkApi.notifyKey(int), getValue()

GlinkEvent_MESSAGE_MODE_DATA

static int GlinkEvent_MESSAGE_MODE_DATA

In message mode operation, notifies that data received from host is available and may be picked up with the ***messageModeReceive*** method. Please note that it may be more convienient to let other events trigger the ***messageModeReceive*** call, for example the ***GlinkEvent.KEYBOARD_UNLOCKED*** if generated (two way alternate sessions), or set up a string notification event for CrLf or just Etx.

See also:
GlinkApi.messageModeReceive(), GlinkApi.notifyString(String, boolean, int, boolean)

GlinkEvent_PRINT_DATA

static int GlinkEvent_PRINT_DATA

Notifies that print data is available. If the printer type configured is GlinkApi, then a ***GlinkEvent_PRINT_DATA*** event will be posted whenever a print data block is available. The ***getValueText*** method returns the posted print data block and the ***getValues*** method returns a last print block flag; 1 is returned for the last print block and 0 is returned for intermediate blocks.

See also:
GlinkConfiguration.setParameter(String)

GlinkEvent_PRINT_STARTED

static int GlinkEvent_PRINT_STARTED

Notifies that print has started but not available yet. Print data is not available until a ***GlinkEvent_PRINT_DATA*** event is posted. ***PRINT_STARTED*** events will be posted only if the printer type configured is GlinkApi..

See also:
GlinkEvent_PRINT_DATA

GlinkEvent_SCREEN_AREA_MATCH

static int GlinkEvent_SCREEN_AREA_MATCH

The ***getValue*** method returns the ***GlinkScreenArea*** identity value.

See also:
GlinkApi.notifyScreenArea(GlinkScreenArea, int, boolean), getValue()

GlinkEvent_STARTED

static int GlinkEvent_STARTED

GlinkEvent_STOPPED

static int GlinkEvent_STOPPED

GlinkEvent_STRING_RECEIVED

static int GlinkEvent_STRING_RECEIVED

The ***getValue*** method returns the string identity value.

See also:
getValue()

GlinkEvent_TURN_LOST

static int GlinkEvent_TURN_LOST

Notifies that this side does not have the turn (permission) to send data and should instead expect to receive data from the other side (host).

See also:
GlinkApi.isTurnKnown()

GlinkEvent_TURN_RECEIVED

static int GlinkEvent_TURN_RECEIVED

Notifies that this side have the turn (permission) to send data.

See also:
GlinkApi.isTurnKnown()

GlinkEvent_VE_CONFIG_ERROR

static int GlinkEvent_VE_CONFIG_ERROR

Error type: An error with the configuration is detected. The session name is not set or a config with the session name specified is not found or an error has occured while loading the config.

The error is only reported if Glink is not visible, otherwise the session dialog box is displayed or a default config is loaded.

See also:
getValue(), GlinkApi.setVisible(boolean), GlinkApi.sessionName(String)

GlinkEvent_VE_CONNECTION_ERROR

static int GlinkEvent_VE_CONNECTION_ERROR

Error type: An error with the connection is detected. Use the ***getValueText()*** method to get the error message.

GlinkEvent_VE_EMULATION_ERROR

static int GlinkEvent_VE_EMULATION_ERROR

Error type: An error with the emulation is detected. Use the ***getValueText()*** method to get the error message.

GlinkEvent_VE_LOGON_FAILED

static int GlinkEvent_VE_LOGON_FAILED

Error type: User name or user password is missing or wrong.

The error is reported only if Glink is not visible, otherwise the logon dialog box is displayed.

See also:
getValue(), GlinkApi.setVisible(boolean)

GlinkEvent_VE_NO_API_LICENSE

static int GlinkEvent_VE_NO_API_LICENSE

Error type: License key allowing use of the API not found.

GlinkEvent_VE_NO_CNX_LICENSE

static int GlinkEvent_VE_NO_CNX_LICENSE

Error type: License key allowing use of connector not found.

GlinkEvent_VE_OUT_OF_MEMORY

static int GlinkEvent_VE_OUT_OF_MEMORY

Error type: The Java VM reported out of memory when loading Glink. Use the ***getValueText()*** method to get the error message.

GlinkEvent_VE_SESSIONS_ERROR

static int GlinkEvent_VE_SESSIONS_ERROR

Error type: The GlinkApi.getAvailableSessions failed. Use the ***getValueText()*** method to get the error message.

GlinkEvent_VE_STARTUP_ERROR

static int GlinkEvent_VE_STARTUP_ERROR

Error type: Glink failed to start. Use the ***getValueText()*** method to get the error message.

GlinkEvent_VE_TOOMANY_INSTANCES

static int GlinkEvent_VE_TOOMANY_INSTANCES

Error type: Too many simultaneous instances for Pro version.

Method Detail

getEventCode

int getEventCode()

The application should check the Glink event code to determine which Glink event did occur, for example that the Glink session was connected or disconnected.

getSource

GlinkApi getSource()

The method returns the GlinkApi object that generated the event.

getValue

int getValue()

Some Glink events have a value set that gives more information about the given event.

getValueText

String getValueText()

Some Glink events have additional text information about the given event.

GlinkField class

Class GlinkField

GlinkApi
GlinkField
class GlinkField

A field is the fundamental element of a virtual screen. A field includes both data and attributes describing the field. The ***GlinkField*** class encapsulates a virtual screen field and provides methods for accessing and manipulating field attributes and data.

GlinkField objects can be accessed only through the ***GlinkFields*** object.

See also:
GlinkFields

Field Summary

static int GlinkField_ATR_BLINK
static int GlinkField_ATR_FIELD_ALPHABETIC
static int GlinkField_ATR_FIELD_ATTRIBUTE
static int GlinkField_ATR_FIELD_DIGITAL
static int GlinkField_ATR_FIELD_MUST_ENTER
static int GlinkField_ATR_FIELD_MUST_FILL
static int GlinkField_ATR_FIELD_NUMERIC
static int GlinkField_ATR_FIELD_RIGHT_JUSTIFY
static int GlinkField_ATR_HIDDEN
static int GlinkField_ATR_INVERSE
static int GlinkField_ATR_UNDERLINE
static int GlinkField_ATR_UNPROTECTED

Method Summary

int getAttribute()

Returns the attribute for the field.

int[ ] getAttributes()

Returns an attribute array for the characters in the field.

byte[ ] getDataBytes()

Returns the screen characters (bytes) for the field in the character set used internally by Glink.

GlinkPoint getEnd()

Returns the ending position of the field.

int getLength()

Returns the length of the field.

GlinkPoint getStart()

Returns the starting position of the field.

String getString()

Returns the field text.

boolean isHidden()

Indicates whether or not the field is hidden.

boolean isHighIntensity()

Indicates whether or not the field is high-intensity.

boolean isModified()

Indicates whether or not the field has been modified.

boolean isNumeric()

Indicates whether or not the field is numeric-only.

boolean isProtected()

Indicates whether or not the field is protected.

void setString(String text)

Sets the field text to the specified string.

Field Detail

GlinkField_ATR_BLINK

static int GlinkField_ATR_BLINK

GlinkField_ATR_FIELD_ALPHABETIC

static int GlinkField_ATR_FIELD_ALPHABETIC

GlinkField_ATR_FIELD_ATTRIBUTE

static int GlinkField_ATR_FIELD_ATTRIBUTE

GlinkField_ATR_FIELD_DIGITAL

static int GlinkField_ATR_FIELD_DIGITAL

GlinkField_ATR_FIELD_MUST_ENTER

static int GlinkField_ATR_FIELD_MUST_ENTER

GlinkField_ATR_FIELD_MUST_FILL

static int GlinkField_ATR_FIELD_MUST_FILL

GlinkField_ATR_FIELD_NUMERIC

static int GlinkField_ATR_FIELD_NUMERIC

GlinkField_ATR_FIELD_RIGHT_JUSTIFY

static int GlinkField_ATR_FIELD_RIGHT_JUSTIFY

GlinkField_ATR_HIDDEN

static int GlinkField_ATR_HIDDEN

GlinkField_ATR_INVERSE

static int GlinkField_ATR_INVERSE

GlinkField_ATR_UNDERLINE

static int GlinkField_ATR_UNDERLINE

GlinkField_ATR_UNPROTECTED

static int GlinkField_ATR_UNPROTECTED

Method Detail

getAttribute

int getAttribute()

Returns the attribute for the field.

Returns:
The raw field attribute.

Bit Position	Description
0-3	Foreground colors
4-6	Background colors
7	Underline
8	Reserved
9	Reserved
10	Blink
11	Reserved
12	Reserved
13	Reserved
14	Reserved
15	Inverse
16	Field, must enter
17	Field, must fill
18	Field, right justify
19	Field, Alpha character
20	Field, Numeric
21	Field, digital
22	Reserved
23	Hidden
24	Reserved
25	Reserved
26	Reserved
27	Reserved
28	Field, modified
29	Reserved
30	Unprotected
31	Field flag

getAttributes

int[ ] getAttributes()

Returns an attribute array for the characters in the field. For 3270 and 5250 the attribute byte itself preceding the field is not included. Use the ***getAttribute*** method to get that attribute.

Returns:
A integer array containing the attributes for the characters in the field

See also:
Attribute format.

getDataBytes

byte[ ] getDataBytes()

Returns the screen characters (bytes) for the field in the character set used internally by Glink.

getEnd

GlinkPoint getEnd()

Returns the ending position of the field. The position can range from 1 to the size of the presentation space. The ending position of a field is the position of the last character in the field.

Returns:
The ending position of the field.

getLength

int getLength()

Returns the length of the field. A field's length can range from 1 to the size of the presentation space.

Returns:
The length of the field.

getStart

GlinkPoint getStart()

Returns the starting position of the field. The position can range from 1 to the size of the virtual screen. The starting position of a field is the position of the first character in the field.

Returns:
The starting position of the field.

getString

String getString()

Returns the field text. This is similar to the ***getDataBytes()*** method, except the data is returned as a string instead of a byte array.

Returns:
The text content of the field returned as a string

isHidden

boolean isHidden()

Indicates whether or not the field is hidden.

Returns:
True if the field is hidden, otherwise false.

isHighIntensity

boolean isHighIntensity()

Indicates whether or not the field is high-intensity.

Returns:
True if the field is high intensity, otherwise false.

isModified

boolean isModified()

Indicates whether or not the field has been modified.

Returns:
True if the field has been modified, otherwise false.

isNumeric

boolean isNumeric()

Indicates whether or not the field is numeric-only.

Returns:
True if the field is numeric only, otherwise false.

isProtected

boolean isProtected()

Indicates whether or not the field is protected.

Returns:
True if the field is protected, otherwise false.

setString

void setString(String text)

Sets the field text to the specified string. If the string is shorter than the length of the field, the rest of the field is cleared. If the string is longer than the field, the text is truncated. A subsequent call to ***getText*** on this field will not show the changed text. To see the changed text, do a ***refresh*** on the ***GlinkFields*** collection and retrieve the refreshed field object.

Parameters:

text - The text to be placed in the field's text plane.

GlinkFields class

Class GlinkFields

GlinkApi
↳GlinkFields
class GlinkFields

GlinkFields contains a collection of the fields in the virtual screen. It provides methods to iterate through the fields, find fields based on location, and find fields containing a given string. Each element of the collection is an instance of ***GlinkField***.

GlinkFields can be accessed through GlinkApi using the ***getFields()*** or ***getVariableFields*** methods. ***GlinkFields*** is a static view of the virtual screen and does not reflect changes made to the virtual screen after its construction. The field list can be updated with a new view of the virtual screen using the refresh() method.

Note:All Field objects returned by methods in this class are invalidated when ***refresh()*** is called.

See also:
GlinkField, GlinkApi.getFields(), GlinkApi.getVariableFields()

Method Summary

GlinkField findByPosition(GlinkPoint targetPosition)

Searches the collection for the target position and returns the ***GlinkField*** object containing that position.

GlinkField findByString(String string, GlinkPoint startPos, int length, int dir, boolean caseSensitive)

Searches the collection for the string and returns the ***GlinkField*** object containing that string.

int getCount()

Returns the number of ***GlinkField*** objects contained in the current form.

int getFieldIndex(GlinkField glField)

Returns the index of a ***GlinkField*** object within the collection.

GlinkField item(int fieldIndex)

Returns the ***GlinkField*** object at the given index.

void refresh()

Updates the collection of ***GlinkField*** objects.

Method Detail

findByPosition

GlinkField findByPosition(GlinkPoint targetPosition)

Searches the collection for the target position and returns the ***GlinkField*** object containing that position.

Parameters:

targetPosition - The target row and column.

Returns:
If found, a ***GlinkField*** object containing the target position. If not found, returns a null.

findByString

GlinkField findByString(String string, GlinkPoint startPos, int length, int dir, boolean caseSensitive)

Searches the collection for the string and returns the ***GlinkField*** object containing that string. The string must be totally contained within the field to be considered a match.

Parameters:

string - The string to search for.

startPos - The row and column on which to start. The position is inclusive (for example, row 1, col 1 means that position 1,1 will be used as the starting location and 1,1 will be included in the search).

length - The length from ***startPos*** to include in the search.

dir - Search direction value:

Constant	Value	Description
GlinkApi_SEARCH_FORWARD	0	Forward (beginning towards end)
GlinkApi_SEARCH_BACKWARD	1	Backward (end towards beginning)

caseSensitive - Indicates whether the search is to be case sensitive. True means the search will be case sensitive.

Returns:
If found, a ***GlinkField*** object containing the search string. If not found, returns a null.

getCount

int getCount()

Returns the number of ***GlinkField*** objects contained in the current form.

Returns:
The number of ***GlinkField*** objects

getFieldIndex

int getFieldIndex(GlinkField glField)

Returns the index of a ***GlinkField*** object within the collection. Indexes start at 1.

Parameters:

glField - The field

Returns:
The index of the field, or 0 if the field is not in the collection.

item

GlinkField item(int fieldIndex)

Returns the ***GlinkField*** object at the given index. The first ***GlinkField*** is at index 1.

Parameters:

fieldIndex - The index of the target field

Returns:
The ***GlinkField*** object at the given index position.

See also:
GlinkField

refresh

void refresh()

Updates the collection of ***GlinkField*** objects. All ***GlinkField*** objects in the current virtual screen are added to the collection. Indexing of ***GlinkField*** objects will not be preserved across refreshes.

GlinkKey class

Class GlinkKey

GlinkApi
↳GlinkKey
class GlinkKey

Glink command keys constants. This class contains the list of the available Glink command keys that can be supplied as a parameter to the ***SendCommandKey*** method. For other characters, use the ***sendKeys*** method.

See also:
GlinkApi.sendCommandKey(int), GlinkApi.sendKeys(String)

Field Summary

static int GlinkKey_ALPHA_OVERIDE

3270 specific: Enable alphanumeric characters for the current unprotected field

static int GlinkKey_BACK_TAB

Tab to previous variable field

static int GlinkKey_BACKSPACE

Delete the character at the previous position if the cursor is placed in an unprotected field

static int GlinkKey_BREAK

Break/Attention

static int GlinkKey_CLEAR

Clear the screen and remove the fields defined

static int GlinkKey_CURSOR_SELECT

3270 specific: Cursor select

static int GlinkKey_DELETE

Delete the character at the cursor position if the cursor is placed in an unprotected field

static int GlinkKey_DOWN

Cursor down

static int GlinkKey_DUP

3270 specific: Set duplicate operation for the rest of the field and do a TAB key operation

static int GlinkKey_END

Move the cursor to the position following the last none space character in the variable field

static int GlinkKey_ENTER

Transmit to host if the keyboard option "Enter sends Transmit" is configured, otherwise a new line is performed

static int GlinkKey_EOL

Erease to end of field or end of line

static int GlinkKey_EOP

Erase all unproteced fields

static int GlinkKey_F1

Program function key 1

static int GlinkKey_F10

Program function key 10

static int GlinkKey_F11

Program function key 11

static int GlinkKey_F12

Program function key 12

static int GlinkKey_F2

Program function key 2

static int GlinkKey_F3

Program function key 3

static int GlinkKey_F4

Program function key 4

static int GlinkKey_F5

Program function key 5

static int GlinkKey_F6

Program function key 6

static int GlinkKey_F7

Program function key 7

static int GlinkKey_F8

Program function key 8

static int GlinkKey_F9

Program function key 9

static int GlinkKey_FIELD_EXIT

5250 specific: Exits and validates the field, erasing from current cursor position to the end of the field

static int GlinkKey_FIELD_MARK

3270 specific: Set a field mark character to inform the application of the end of the field for an unformatted screen

static int GlinkKey_FIELD_MINUS

5250 specific: Exits and validates the field as a negative value

static int GlinkKey_FIELD_PLUS

5250 specific: Exits and validates the field as a positive value

static int GlinkKey_HOME

Go to first variable field

static int GlinkKey_HOST_HELP

5250 specific: Request host help

static int GlinkKey_INSERT

Set the emulator into insert mode

static int GlinkKey_LEFT

Cursor left

static int GlinkKey_NEW_LINE

New line

static int GlinkKey_PA1

3270 specific: Program access key 1

static int GlinkKey_PA2

3270 specific: Program access key 2

static int GlinkKey_PA3

3270 specific: Program access key 3

static int GlinkKey_PA4

3270 specific: Program access key 4

static int GlinkKey_PGDN

Page down, 5250 Roll up

static int GlinkKey_PGUP

Page up, 5250 Roll down

static int GlinkKey_PRINT_SCREEN

Print screen

static int GlinkKey_RESET_KEYBOARD

3270 and 5250 specific: Reset keyboard

static int GlinkKey_RIGHT

Cursor right

static int GlinkKey_SF1

Program function key shift/F1 or program function key 13 for 3270

static int GlinkKey_SF10

Program function key shift/F10 or program function key 22 for 3270

static int GlinkKey_SF11

Program function key shift/F11 or program function key 23 for 3270

static int GlinkKey_SF12

Program function key shift/F12 or program function key 24 for 3270

static int GlinkKey_SF2

Program function key shift/F2 or program function key 14 for 3270

static int GlinkKey_SF3

Program function key shift/F3 or program function key 15 for 3270

static int GlinkKey_SF4

Program function key shift/F4 or program function key 16 for 3270

static int GlinkKey_SF5

Program function key shift/F5 or program function key 17 for 3270

static int GlinkKey_SF6

Program function key shift/F6 or program function key 18 for 3270

static int GlinkKey_SF7

Program function key shift/F7 or program function key 19 for 3270

static int GlinkKey_SF8

Program function key shift/F8 or program function key 20 for 3270

static int GlinkKey_SF9

Program function key shift/F9 or program function key 21 for 3270

static int GlinkKey_SYSREQ

3270 and 5250 specific: System request key

static int GlinkKey_TAB

Tab to next variable field

static int GlinkKey_TESTREQ

5250 specific: Test request

static int GlinkKey_TRANSMIT

Transmit to host

static int GlinkKey_UP

Cursor up

Constructor Summary

GlinkKey()

Method Summary

static String toString(int key)

static int toKey(String key)

Field Detail

GlinkKey_ALPHA_OVERIDE

static int GlinkKey_ALPHA_OVERIDE

3270 specific: Enable alphanumeric characters for the current unprotected field

GlinkKey_BACK_TAB

static int GlinkKey_BACK_TAB

Tab to previous variable field

GlinkKey_BACKSPACE

static int GlinkKey_BACKSPACE

Delete the character at the previous position if the cursor is placed in an unprotected field

GlinkKey_BREAK

static int GlinkKey_BREAK

Break/Attention

GlinkKey_CLEAR

static int GlinkKey_CLEAR

Clear the screen and remove the fields defined

GlinkKey_CURSOR_SELECT

static int GlinkKey_CURSOR_SELECT

3270 specific: Cursor select

GlinkKey_DELETE

static int GlinkKey_DELETE

Delete the character at the cursor position if the cursor is placed in an unprotected field

GlinkKey_DOWN

static int GlinkKey_DOWN

Cursor down

GlinkKey_DUP

static int GlinkKey_DUP

3270 specific: Set duplicate operation for the rest of the field and do a TAB key operation

GlinkKey_END

static int GlinkKey_END

Move the cursor to the position following the last none space character in the variable field

GlinkKey_ENTER

static int GlinkKey_ENTER

Transmit to host if the keyboard option "Enter sends Transmit" is configured, otherwise a new line is performed

GlinkKey_EOL

static int GlinkKey_EOL

Erease to end of field or end of line

GlinkKey_EOP

static int GlinkKey_EOP

Erase all unproteced fields

GlinkKey_F1

static int GlinkKey_F1

Program function key 1

GlinkKey_F2

static int GlinkKey_F2

Program function key 2

GlinkKey_F3

static int GlinkKey_F3

Program function key 3

GlinkKey_F4

static int GlinkKey_F4

Program function key 4

GlinkKey_F5

static int GlinkKey_F5

Program function key 5

GlinkKey_F6

static int GlinkKey_F6

Program function key 6

GlinkKey_F7

static int GlinkKey_F7

Program function key 7

GlinkKey_F8

static int GlinkKey_F8

Program function key 8

GlinkKey_F9

static int GlinkKey_F9

Program function key 9

GlinkKey_F10

static int GlinkKey_F10

Program function key 10

GlinkKey_F11

static int GlinkKey_F11

Program function key 11

GlinkKey_F12

static int GlinkKey_F12

Program function key 12

GlinkKey_FIELD_EXIT

static int GlinkKey_FIELD_EXIT

5250 specific: Exits and validates the field, erasing from current cursor position to the end of the field

GlinkKey_FIELD_MARK

static int GlinkKey_FIELD_MARK

3270 specific: Set a field mark character to inform the application of the end of the field for an unformatted screen

GlinkKey_FIELD_MINUS

static int GlinkKey_FIELD_MINUS

5250 specific: Exits and validates the field as a negative value

GlinkKey_FIELD_PLUS

static int GlinkKey_FIELD_PLUS

5250 specific: Exits and validates the field as a positive value

GlinkKey_HOME

static int GlinkKey_HOME

Go to first variable field

GlinkKey_HOST_HELP

static int GlinkKey_HOST_HELP

5250 specific: Request host help

GlinkKey_INSERT

static int GlinkKey_INSERT

Set the emulator into insert mode

GlinkKey_LEFT

static int GlinkKey_LEFT

Cursor left

GlinkKey_NEW_LINE

static int GlinkKey_NEW_LINE

New line

GlinkKey_PA1

static int GlinkKey_PA1

3270 specific: Program access key 1

GlinkKey_PA2

static int GlinkKey_PA2

3270 specific: Program access key 2

GlinkKey_PA3

static int GlinkKey_PA3

3270 specific: Program access key 3

GlinkKey_PA4

static int GlinkKey_PA4

3270 specific: Program access key 4

GlinkKey_PGDN

static int GlinkKey_PGDN

Page down, 5250 Roll up

GlinkKey_PGUP

static int GlinkKey_PGUP

Page up, 5250 Roll down

GlinkKey_PRINT_SCREEN

static int GlinkKey_PRINT_SCREEN

Print screen

GlinkKey_RESET_KEYBOARD

static int GlinkKey_RESET_KEYBOARD

3270 and 5250 specific: Reset keyboard

GlinkKey_RIGHT

static int GlinkKey_RIGHT

Cursor right

GlinkKey_SF1

static int GlinkKey_SF1

Program function key shift/F1 or program function key 13 for 3270

GlinkKey_SF2

static int GlinkKey_SF2

Program function key shift/F2 or program function key 14 for 3270

GlinkKey_SF3

static int GlinkKey_SF3

Program function key shift/F3 or program function key 15 for 3270

GlinkKey_SF4

static int GlinkKey_SF4

Program function key shift/F4 or program function key 16 for 3270

GlinkKey_SF5

static int GlinkKey_SF5

Program function key shift/F5 or program function key 17 for 3270

GlinkKey_SF6

static int GlinkKey_SF6

Program function key shift/F6 or program function key 18 for 3270

GlinkKey_SF7

static int GlinkKey_SF7

Program function key shift/F7 or program function key 19 for 3270

GlinkKey_SF8

static int GlinkKey_SF8

Program function key shift/F8 or program function key 20 for 3270

GlinkKey_SF9

static int GlinkKey_SF9

Program function key shift/F9 or program function key 21 for 3270

GlinkKey_SF10

static int GlinkKey_SF10

Program function key shift/F10 or program function key 22 for 3270

GlinkKey_SF11

static int GlinkKey_SF11

Program function key shift/F11 or program function key 23 for 3270

GlinkKey_SF12

static int GlinkKey_SF12

Program function key shift/F12 or program function key 24 for 3270

GlinkKey_SYSREQ

static int GlinkKey_SYSREQ

3270 and 5250 specific: System request key

GlinkKey_TAB

static int GlinkKey_TAB

Tab to next variable field

GlinkKey_TESTREQ

static int GlinkKey_TESTREQ

5250 specific: Test request

GlinkKey_TRANSMIT

static int GlinkKey_TRANSMIT

Transmit to host

GlinkKey_UP

static int GlinkKey_UP

Cursor up

Constructor Detail

GlinkKey

GlinkKey()

Method Detail

toString

static String toString(int key)

toKey

static int toKey(String key)

GlinkScreenArea class

Class GlinkScreenArea

GlinkApi
↳GlinkScreenArea
class GlinkScreenArea

Use ***GlinkScreenArea*** objects to facilitate screen/form identification. Often a given text string at a specific location is enough to identify the current screen. Use the ***GlinkApi.getScreenArea*** to get an object for a given screen area and use the ***GlinkApi.notifyScreenArea*** for later be notified when the current screen matches this object. You may also use the ***GlinkApi.isScreenAreaMatch*** method to check if the object matches the current screen.

Two types of screen area are supported: rectangle and standard area. The latter goes from the starting point and continues on a line-by-line basis to the end point.

See also:
GlinkApi.getScreenArea(GlinkPoint, GlinkPoint, boolean),
GlinkApi.getMarkedScreenArea(),
GlinkApi.notifyScreenArea(GlinkScreenArea, int, boolean),
GlinkApi.isScreenAreaMatch(GlinkScreenArea)

Constructor Summary

GlinkScreenArea(GlinkPoint start, GlinkPoint end, boolean rectangle, String string)

Method Summary

GlinkPoint getEnd()

Returns the end posistion for screen area.

GlinkPoint getStart()

Returns the start posistion for the screen area.

String getString()

Returns the text string for the given screen area.

boolean isRectangle()

Returns true if the object is for a rectangle area.

Constructor Detail

GlinkScreenArea

GlinkScreenArea(GlinkPoint start, GlinkPoint end, boolean rectangle, String string)

Method Detail

getEnd

GlinkPoint getEnd()

Returns the end posistion for screen area.

getStart

GlinkPoint getStart()

Returns the start posistion for the screen area.

getString

String getString()

Returns the text string for the given screen area. If the area is a rectangle, then all the text strings within the rectangle are concatenated into one string without any line separators.

isRectangle

boolean isRectangle()

Returns true if the object is for a rectangle area.

GlinkEventListener interface

Interface GlinkEventListener

interface GlinkEventListener

Implement this interface to allow the class to be registered as a ***GlinkEventListener***.

Method Summary

void onGlinkEvent(GlinkEvent event)

This method will be called when a Glink event is generated.

Method Detail

onGlinkEvent

void onGlinkEvent(GlinkEvent event)

This method will be called when a Glink event is generated.

See also:
GlinkEvent

Glink.Auto OLE Automation

Overview

OLE (Object Linking and Embedding) allows you to manipulate objects from other applications using the Component Object Model (COM). COM defines how objects interact between applications.

OLE Automation makes use of this model by not only allowing external applications to manipulate the server application in a predefined way but also allowing the external application (known as the Automation controller) to inquire dynamically as to what functionality is available. This is done through a standard mechanism called the Idispatch interface.

An Automation server is always identified by a unique 'class' which allows you to access a copy of the application that supports it. Each Automation server may define two types of members that you can expose from its object:

- Methods perform an action on an object. For example, Glink script commands or menu commands.
- Properties are characteristics of an object - For example, Glink internal or global variables.

An Automation server exposes its objects for use by Automation controller client applications. It is then up to the clients to request and manipulate the server's objects.

Automation Controller

The Glink script command OLE allows Glink to become an OLE Automation controller

```
OLE CREATE OleID "class.name"
OLE CONNECT OleID "class.name"
OLE GET %v OleID.Property
OLE SET OleID.Property value
OLE CALL OleID.Method
```

```
OLE CALR %v OleID.Method [params]
OLE FREE OleID
```

The `OLE CREATE` command allows you to create an instance of an OLE Automation object. To access a copy of Word for example you would use the name "Word.Basic".

`OLE CONNECT` is similar but lets you connect to an object that already exists.

To obtain the value of a property supported by the server you can use the `OLE GET` command. To set the value of a property you would use the `OLE SET` command.

To invoke a method that doesn't return a value (or that returns a value you don't need) you use `OLE CALL`, while to invoke one that does you would use `OLE CALR`.

Finally when you are done with the automation server you can use `OLE FREE` (this is done automatically when the script terminates if you should forget to do it yourself).

What this means in practice is best illustrated with an example. Let's imagine that we have a document in Word format and want to send it to our host machine. There is no point in sending the file itself in that our host has no way to extract the text from the document - we need Word for that.

```
Ole Create Word "Word.Basic"
Ole Call Word.FileOpen "C:\MYDOCS\DOC1.DOC"
Ole Call Word.EditSelectAll
Ole Call Word.EditCopy
Ole Free Word
Binary OFF
Set Transfer ANSI
SndLine "GKRM F"
PutFile FTRAN "*CLP;doc1"
```

What we are doing here is to ask Word to do the job. The example script first creates an object in the "Word.Basic" class called "Word" (this is just a name we choose to represent the object). Now that we have the object we ask it to open the file, select the entire text and copy it to the clipboard (there is no need to actually write to a file). We are now done with Word and can free the object. The rest of the script just uploads the contents of the clipboard to a file called "doc1" on the host.

Note that the copy of Word we start to do the conversion will not be visible (unless we ask it to be).

Automation Server

In the same way that Glink can operate as either a DDE server or a DDE client, it will also operate as an Automation server or an Automation controller.

The class has been called "Glink.Auto" and is registered automatically when you start Glink. Once that's been done you can immediately start using Glink from other applications that support OLE Automation. This includes, amongst others, Visual Basic for Applications (VBA) as provided in the Office 97 suite from Microsoft.

As well as providing OLE Automation events, the Glink.Auto class is a dual interface and allows run-time linking in the normal fashion, but also direct linking using the Glink.Auto VTable definition.

Dual-interface, Type Library definition

The VTable definition is contained in Glink's Type Library (TLB) which is included in the GL.EXE file or the supplied GLINK.TLB resource file. Including the TLB in your Visual Development tool will normally allow the tool to display the Glink.Auto methods, properties and events as you program. It also allows syntax checking at compile time. How you include the TLB definitions in your program is tool-dependent.

Using Glink.Auto with MS Visual Basic

VB needs to load Glink.Auto VTable Type Library (TLB) definition to be able to display the Glink.Auto methods, properties and events as you program.

Including the Glink.Auto definition is done via the Project/References... menu, which displays a list of references that are available. Glink's type library reference is called "Glink Auto Interface" and is located in the GL.EXE file. Once it's been included you should see that the "Glink" object is added to the list of selectable objects when you try to define a variable, for example type:

```
Dim Gl as
```

and the list appears. Once you have selected "Glink", type a dot and a list of interfaces will be displayed:

```
Dim Gl as Glink.
```

Select "Auto".

```
Dim Gl as Glink.Auto
```

If you want the Glink.Auto object created automatically as soon as it's used, then use:

```
Dim Gl as new Glink.Auto
```

If you are going to use the Glink's Automation events, then you must use the WithEvents keyword. However, when using the WithEvents keyword you cannot use the New keyword and will need to create the object at runtime:

```
Dim WithEvents Gl as Glink.Auto
```

Private Sub Form_Load()

```
 Set Gl = New Glink.Auto
 'or Set Gl = CreateObject("Glink.Auto")
 'or Set Gl = GetObject("", "Glink.Auto")
End Sub
```

Once the object is created you can use any method or property directly.

```
Text1.Text = Gl.Screen.ScreenText
```

Lists of the relevant properties and methods will appear as soon as you type the dot (or bracket).

```
If Gl.Screen.FormsMode then
 If Gl.Screen.Field(1).Fixed = false then
   Text1.Text = Gl.Screen.Field(1).FieldText
 End If
End If
```

If the WithEvents keyword was used when declaring your object variable, then your Glink.Auto object name will be displayed in the Object list. In the above example it would be called "Gl". Select it and the Procedures list will be filled with the available Glink.Auto Automation events, OnConnect, OnDisconnect, OnTurn, etc. Selecting one of the Events will generate the function which will be called when Glink signals that particular event.

```
Function Gl_OnTurn()
 Text1.Text = Gl.Screen.ScreenText
```

```
End Sub
```

Glink.Auto, a single instance interface

As Glink is not a Multi-Document Interface (MDI) in the same way as applications which open spread-sheets, documents, the Glink.Auto object is a single-instance interface. This means that only one Automation client can connect to it at a given time.

When you create a Glink.Auto object, the system will connect you to the first available copy of Glink. If there are no copies running, or they are all already in use as automation servers, then the system will start up a new instance of Glink. When you retrieve the Glink.Auto object, you can find out if it is a new instance by testing the gl.Automation property.

```
Set gl = GetObject("", "Glink.Auto")
If Not gl Is Nothing Then
     If gl.Automated = true Then
          gl.LoadConfig("def.glinkconfig")
          gl.Connect
     End If
     ....
     Set gl = Nothing
End If
```

Connecting to an running Glink session

As Glink is a single instance interface, you can not tell in advance which Glink you will get connected to. To connect to a specific Glink, you need to enumerate the Glinks until you find the Glink you require. See the section about enumerating Glink sessions.

Starting up a new Glink

The system will only start up a new instance of Glink when all copies of the program in memory are already in use as automation servers. If you need to start up a new Glink instance, then you need to enumerate all current copies (making each unavailable for further connections) until you get to the point where a new copy is started. See the section about enumerating Glink sessions.

Enumerating Glink sessions with VB

As you only get a connection to the next Glink session once the previous one is connected, you need to continue connecting to the next Glink.Auto without freeing any of the copies you have connected to already. This can be done in several ways -- below is a generic recursive method which connects to a specific Glink session using

Glink Title+"!"+Glink instance number

as an identification for a specific Glink. In Visual Basic this is:

```
gl.Caption + "!" + gl.Instance
```

An empty string can be used to start a new instance of Glink:

```
Dim gl As Glink.Auto

Private Function NextGlink(Name As String, Display As Boolean)
As Glink.Auto

    Dim gl As Glink.Auto
    Dim glnext As Glink.Auto
    Dim S As String
    Dim A As Boolean

    Set NextGlink = Nothing
    Set gl = GetObject("", "Glink.Auto")
    If Not gl Is Nothing Then
        A = gl.Automated
        S = gl.Caption + "!" + Str(gl.Instance)
        gl.ShowText (S)
        If Display And (A = False) Then
            List1.AddItem (S)
        End If
        If Name = "" Then
            If A = True Then
                Set NextGlink = gl
            Else
                Set NextGlink = NextGlink(Name, Display)
                Set gl = Nothing
            End If
        Else
            If Name = S Then
                Set NextGlink = gl
            Else
                If A = True Then   ' new session, close it
                    Set gl = Nothing
                Else
                    Set NextGlink = NextGlink(Name, Display)
                    Set gl = Nothing
                End If
            End If
        End If
    End If
End Function

Private Sub StartNewGlink()

    Set gl = NextGlink("", False)
    If Not gl Is Nothing Then
        MsgBox ("New Glink created, will terminate it now")
        Set gl = Nothing
    End If
End Sub

Private Sub ConnectToRunningGlink(Name As String)

    Set gl = NextGlink(Name, False)
    If Not gl Is Nothing Then
        MsgBox ("Connected to " + Name)
        Set gl = Nothing
    End If
End Sub
```

```
' New Glink

Private Sub Command1_Click()
    StartNewGlink
End Sub

Private Sub Command2_Click()

    Dim Name As String

    Name = List1.Text
    If Name = "" Then
        MsgBox ("You must select a Glink from the list first")
    Else
        ConnectToRunningGlink (Name)
    End If
End Sub

' Refresh list

Private Sub Command3_Click()

    List1.Clear
    Set gl = NextGlink("", True)
    If Not gl Is Nothing Then
        Set gl = Nothing
    End If
End Sub
```

Glink.Auto programmers reference

Events

The following events are available:

Event	Description
OnConnect	Connection event
OnDisconnect	Disconnection event
OnTurn	Turn or new screen received
OnData	Communications data received
OnPattern	User defined pattern received

Glink's Automation events allow you to supply callback functions which will be called by Glink when the specified event occurs.

The OnConnect Event

OnConnect()

This event occurs when Glink successfully connects.

VBA Example:

```
Private Sub Gl_OnConnect()
 List1.AddItem ("OnConnect received")
End Sub
```

The OnDisconnect Event

OnDisconnect()

This event is generated if the communication link goes down or gets disconnected either by the host or by the user.

VBA Example:

```
Private Sub Gl_OnDisconnect()
 List1.AddItem ("OnDisconnect received")
End Sub
```

The OnTurn Event

OnTurn()

This event signals that the host has finished sending data and is waiting for user input. It will normally indicate that the host has sent a new screen and that all the data has been processed by Glink. The detection of the turn depends on the communications interface being used and the emulation mode. If the emulation mode is FORMS or TEXT or the communications interface supports the notion of TURN (e.g. Ggate, TNVIP, TN3270, TN5250), then it will be immediate, otherwise the turn will be triggered 500 milliseconds after the host has stopped sending data. In both cases, the turn will only be generated if the keyboard is unlocked.

VBA Example:

```
Private Sub Gl_OnTurn()
 Dim i, n As Integer

 List1.AddItem ("OnTurn received")
 Text2.Text = Gl.Screen.ScreenText
 Text3.Text = Gl.Screen.Status.StatusText

 If Gl.Screen.FormsMode = True Then
   i = 0
   n = Gl.Screen.FieldCount
   While i < n
     AddField (i)
     i = i + 1
   Wend
 End If
End Sub
```

The OnData Event

OnData()

The OnData event is triggered when the host starts sending new data.

VBA Example:

```
Private Sub Gl_OnData()
 List1.AddItem ("OnData received")
End Sub
```

The OnPattern Event

OnPattern (PatternId)

PatternId: Integer Id value supplied on Pattern method

This event occurs when a user defined pattern occurs. Patterns are set by suing the Gl.Pattern method.

VBA Example:

```
Private Sub Gl_OnPattern(ByVal PatternId As Long)
If PatternId = 1 Then
  Gl.Pattern(1, "")
  Text1.Text = "Got the logon screen"
  DoMyLogon
End If
End Sub
```

Properties

The following properties are provided by Glink:

Property	Description
`Automated`	Provides Glink automation status
`Caption`	Current window title
`CursorX`	Current screen X coordinate
`CursorY`	Current screen Y coordinate
`DDEname`	Current DDEname
`Gparam`	Global script parameters
`Height`	Height of main window (pixels)
`Iconic`	Iconic state
`Info interface`	Emulation information interface
`Instance`	Glink instance identifier
`Left`	X-coordinate of main window (pixels)
`Param`	Script run-time parameters
`ProcessLine`	Process and emulate host input
`Screen interface`	Screen information interface
`Top`	Y-coordinate of main window (pixels)
`Visible`	Visible state
`Width`	Width of main window (pixels)
`Zoomed`	Zoomed state

These properties allow you to get or set the various aspects of Glink. All the Glink properties are readable and writable (get/set). For example, you can not only find out where the Glink window is positioned by reading the Left and Top properties, but you can change its position by assigning new values to the same properties.

The Automated property

Automated: Boolean Automation status

This property can be used to find out whether or not the copy of Glink to which you are connected is one that was started as a result of your CreateObject or that was already running at the time. Note that if you disconnect from Glink and leave the Glink window visible then that copy will remain in memory and be available for future sessions; the automated property will then change from true to false.

The Caption property

Caption: String Window title

This property can be used to get the current Glink Window title, or set it to a new title.

VBA Example:

```
MsgBox (GL.Caption)
GL.Caption = "New title"
MsgBox (GL.Caption)
```

The CursorX property

CursorX: Integer Cursor X co-ordinate

This property can be used to get or set the cursor's X position. The X position value goes from 1 to the width (in characters) of the emulation screen. This will depend on the emulation settings, but will normally be 80.

The CursorY property

CursorY: Integer Cursor Y co-ordinate

This property can be used to get or set the cursor's Y position. The Y position value goes from 1 to the number of lines in the emulation screen. This will depend on the emulation settings, but will normally be 24.

The DDEname property

DDEname: String Current DDEname

This property can be used to read or set Glink's DDE name, See the DDENAME script command for more details

The Gparam property

Gparam: String Global script parameter

This property can be used to get or set the global script parameters $GPARAM. Gparam is provided in the same way as it is for DDE, to allow you to interact with a script that might be running.

VBA Example:

```
GL.GParam = "0"
GL.ScriptCommand ("Idle 10;Gparam 1")
While GL.GParam = "0"
 DoEvents
Wend
```

The Height property

Height: Integer Window height

This property can be used to get or set the height of the main Glink window. This value is in pixels.

The Iconic property

Iconic: Boolean Iconic state

This property can be used to get or set the current iconic state of the Glink window. At start-up, this value will be FALSE.

The Info interface property

Info: IAutoInfo Emulation information interface

This property returns an interface to another object which contains general emulation information properties. See the Info interface section for more information on these properties.

The Instance property

Instance: Integer Instance identifier

This property returns an integer which uniquely identifies the current copy of Glink. This may be used to check that you are connected to the session you intended.

The Left property

Left: Integer Left window co-ordinate

This property can be used to get or set the left co-ordinate of the top-left corner of the main Glink window. This value is in pixels.

VBA Example:

```
Dim GL As Object
Set GL = CreateObject ("Glink.Auto")
GL.Visible = True
GL.Left = 0
```

The Param property

Param: String Current script parameter

This property can be used to get or set the current script parameters $PARAM. Param is provided in the same way as it is for DDE, to allow you to interact with a script that might be running.

The ProcessLine property

ProcessLine: Boolean Host data processing state

This property can be used to get or set whether Glink is to process host data. When TRUE, Glink will emulate received host data. By default Glink will not process line data when started as an automation server, rather it expects the automation controller to do this via the Receive and Emulate methods.

VBA Example:

```
Dim GL As Object
Dim Cont As Boolean
Dim S As String
Set GL = CreateObject("Glink.Auto")
GL.LoadConfig ("UNIX.glinkconfig")
GL.Connect("")
Cont = True
Do
 S = GL.Receive(100, String(1, Chr(13)), 100, True)
 If S > "" Then
   If InStrB(S, "ogin:") > 0 Then
     GL.Transmit ("name")
   End If
   If InStrB(S, "ssword") > 0 Then
     GL.Transmit ("pass")
     Cont = False
   End If
 End If
Loop Until Cont = False
GL.Visible = True
```

The Screen interface property

Screen: IAutoScreen Screen information interface

The Screen property returns an interface to the Screen allowing you to read and update the fields on the screen. It also allows you to read the status line. See the Screen interface section for more information.

The Top property

Top: Integer Top window co-ordinate

This property can be used to get or set the top co-ordinate of the top-left corner of the main Glink window. This value is in pixels.

VBA Example:

```
Dim GL As Object
Set GL = CreateObject ("Glink.Auto")
GL.Visible = True
GL.Top = 0
```

The Visible property

Visible: Boolean Visible state

This property can be used to get or set the visible state of the Glink window. When started as an automation server, this value will be FALSE which means that the Glink window will be hidden. You can change this by setting the Visible property to TRUE. If the Glink Visible property is still TRUE when you finish using the Glink.Auto object, Glink will be left on the screen rather than terminated.

VBA Example:

```
Dim GL As Object
Set GL = CreateObject ("Glink.Auto")
GL.LoadConfig ("WRK.glinkconfig")
GL.ProcessLine = True
GL.GWConnect ("ggate", "phoenix")
GL.GParam = "0"
GL.ScriptCommand ("Conv 'OGIN' 'NAME'; Conv 'SSWORD' 'PASS'; GPar '1'")
While GL.GParam = "0"
 DoEvents
Wend
GL.Visible = True
```

The Width property

Width: Integer Window width

This property can be used to get or set the width of the main Glink window. This value is in pixels.

The Zoomed property

Zoomed: Boolean Zoomed state

This property can be used to get or set the current zoomed state of the Glink window. At start-up, this value will be FALSE.

Methods

The following methods are provided by Glink:

Method	Description
`Connect`	Connects the communications line
`Disconnect`	Disconnects the communications line
`Emulate`	Emulates a string locally
`GWConnect`	Makes a connection through a gateway
`LoadConfig`	Changes to another configuration
`Pattern`	Enables OnPattern event
`Quit`	Stops the emulator
`Receive`	Waits for a defined string from the host
`ScriptCommand`	Executes a script command
`ScriptFile`	Executes a script file
`Send`	Sends a message to the host without a terminator
`Show`	Sends a message to the terminal
`ShowText`	Shows a message locally, no CRLF
`Transmit`	Sends a message without local emulation

These concentrate mostly on handling of line data, and mirror much of what you also see in the DDE interface. Especially important here is the ability to execute any script command or script file - given that you can do that then there isn't much you can't do from your automation controller in that the script language is so extensive.

Here again we can maybe get a better feel for what we are looking at by running through a real life example. We'll do more or less the same kind of thing as in the Automation controller client Word.basic example, but this time the file will go in the reverse direction; and this time we'll let Word be the controller and Glink be the server.

```
Dim GL As Object
Set GL = CreateObject ("Glink.Auto")
GL.LoadConfig ("WSK.glinkconfig")
GL.ProcessLine = True
GL.GWConnect ("ggate", "phoenix")
GL.GParam = "0"
GL.ScriptCommand ("Rece '-^$03';Sndl 'Gkrm F';GetF FTRA
'*CLP;msg';GPar '1'")
While GL.GParam = "0"
 DoEvents
Wend
Selection.Paste
```

The routine shown is the basis for a Word 7 macro, and as such is written in VBA.

It starts off by defining the Glink object. The first statement creates an instance of Glink. We then load a suitable configuration file with LoadConfig, and connect to our host with GWConnect (the example was run over a Ggate connection). Now we set the Gparam property to "0" (Gparam is equivalent to the script global parameter and is useful for script synchronization as it is in DDE applications). The script command we execute first waits for the host prompt, then sends a file transfer request and then brings down the file into the clipboard. Finally it sets the global script parameter to "1".

Invoking the ScriptCommand method will only start the process described, while the VBA routine can carry on doing whatever it likes in the meantime. However, we want to wait until the file is here, so the next three statements do just that.

Finally we quite simply paste in the contents of the clipboard (which now contains the contents of the GCOS file) at the current insertion point.

A point to note is that Glink does not initially connect to the host until the Automation controller client requests it via the Connect, GWConnect or script command methods.

The Connect method

Connects the communications line

Connect (Hostname)

Hostname: String host name to connect to

Return value: none

This method connects the communications line. If the configured communications interface supports a host name, such as TCP/IP connections, then the Hostname parameter will be use. If an empty string is provided, then the default config values will be used.

The Disconnect method

Disconnects the communications line

Disconnect( )

Return value: none

This method disconnects the communications line.

The Emulate method

Emulates a string locally

Emulate (Hostdata)

Hostdata: String host data to emulate

Return value: none

This method emulates a string locally. This string may be simple text or contain emulation escape sequences. Normally this command will be used to process host data obtained with the Receive method when the Process variable was passed as FALSE.

VBA Example:

```
Dim S As String
S = GL.Receive(100, String(1, Chr(03)), 100, FALSE)
If (S > "") Then
 Emulate (S)
 If InStrB(S, "MODEL") > 0 Then
   GL.Transmit ("VIP7804")
 Else If InStrB(S, "LOGON") > 0 Then
   GL.Transmit ("NAME")
 End If
End If
```

The GWConnect method

Makes a connection through a gateway

GWConnect (Gatewayname, Hostname)

Gatewayname: String	Gateway name to connect through
Hostname: String	host name to connect to
Return value:	none

This method makes a connection to a host through a gateway. This function can be used for G&R/Ggate connections or other communication interfaces requiring a gateway name. If empty strings are provided in either of the parameters, then the corresponding default config value will be used.

The LoadConfig method

Changes to another configuration

LoadConfig (Configfilepath)

Configfilepath: String	Config file to load
Return value: Boolean	TRUE if successful, otherwise FALSE

This method changes to another Glink configuration file. When started as an automation server, the default Glink config file, DEF.glinkconfig will be loaded.

The Pattern method

Enables the OnPattern event

Pattern (PatternId, Pattern)

PatternId: Long	PatternId supplied with OnPattern event
Pattern: String	Pattern to receive from the host
Return value: Boolean	TRUE if successful, otherwise FALSE

This method enables the OnPattern event. The PatternId is passed as a parameter to the OnPattern event callback function. Up to twenty different 'patterns' may be set using this command using the PatternId, numbered from 1 to 20. Glink scans the data received from the host for any patterns that may be set.

VBA Example:

```
Gl.Pattern(1, "Logon:")

Private Sub Gl_OnPattern(ByVal PatternId As Long)
If PatternId = 1 Then
  Gl.Pattern(1, "")
  Text1.Text = "Got the logon screen"
  DoMyLogon
End If
End Sub
```

Using an empty Pattern string cancels the event. Supplying ZERO as the PatternId will cancel all Pattern events.

The Quit method

Stops the emulator

Quit ( )

Return value:	none

This method tells Glink to terminate. The Glink object will no longer be valid after this call.

The Receive method

Waits for one of the defined characters from the host

Receive (Maxlen, Waitchars, Timeout, Process)

Maxlen: Integer	Max number of host data to receive
Waitchars: String	Characters to wait for
Timeout: Integer	Max millisecond to wait receive
Process: Boolean	Process host data received
Return value: String	Received data from the host

This method returns the string of characters received from the host. It will stop receiving when any of the characters in the Waitchars string is received from the host or if the Timeout has elapsed or if the MaxLen number of characters has been received. If the Process variable is TRUE, then the characters will be processed and displayed in the emulation window as they arrive from the host. If set to FALSE, then it will be up to the automation controller to return them by using the Emulate method if required. If the no data has arrived before the Timeout elapses, then the string will be empty

To have full control over the host data, you should set the ProcessLine property to FALSE otherwise data will be automatically emulated by Glink when there is no outstanding Receive method being called.

VBA Example:

```
Dim S As String
S = GL.Receive(100, String(1, Chr(03)), 100, True)
If InStrB(S, "MODEL") > 0 Then
 GL.Transmit ("VIP7804")
End If
```

The ScriptCommand method

Executes a script command

ScriptCommand (Command)

Command: String	Script command to execute
Return value:	none

This method executes a script command. The call returns immediately, so the script command may not have terminated when it returns.

VBA Example:

```
GL.ScriptCommand ("Set Mrect ON; Mark 8 5 72 22;Perform Copy")
```

The ScriptFile method

Executes a script command

ScriptFile (Scriptfilepath)

Scriptfilepath: String Script file to execute

Return value: none

This method executes a script file. The call returns immediately, so the script file may not have terminated when it returns.

The Send method

Sends data to the host without a terminator

Send (Data)

Data: String Data to send to the host

Return value: none

This method sends a sting of data to the host without local emulation and without a terminator. If a terminator is needed, then the Transmit method should be used.

VBA Example:

```
Dim S As String
S = GL.Receive(100, String(1, Chr(03)), 1000, True)
If InStrB(S, "MODEL") > 0 Then
 GL.Send ("VIP7804")
 GL.Transmit ("")
End If
```

The SendBreak method

Sends a break to the host

SendBreak ( )

Return value: none

This method sends a communications break to the host.

The Show method

Sends a message to the emulator

Show (Text)

Text: String Text to send

Return value: none

This method sends a message to the emulator as if it was typed in at the keyboard.

The ShowText method

Displays a message in the emulator window

ShowText (Text)

Text: String Text to display

Return value: none

This method displays a message in the emulator window. The text is not processed in anyway.

VBA Example:

```
GL.ShowText("Hello world!")
```

The Transmit method

Sends data to the host with a terminator

Transmit (Data)

Data: String Data to send to the host

Return value: none

This method sends a sting of data to the host without local emulation but with a terminator. The terminator will depend on the emulation state and the communications interface. Generally in FORMS or BLOCK modes, the terminator will be ETX, otherwise it will be CR.

VBA Example:

```
Dim S As String
S = GL.Receive(100, String(1, Chr(03)), 1000, True)
If InStrB(S, "MODEL") > 0 Then
 GL.Transmit ("VIP7804")
End If
```

The Info Interface

The following Glink.Info properties are available:

Property	Description
ApplicationKeypad	Application keypad mode
AutoLF	Auto line feed
AutoTabL	Auto tabbing
AutoWrap	Auto wrapping mode
Busy	Communications busy (read-only)
Capturing	Capture mode
CommName	Communications name (read-only)
CursorApplication	Cursor application mode
EmulMode	Emulation mode (read-only)
GrMode	Semi-graphics mode
InsertMode	Insert mode
KeybLocked	Keyboard locked state
LocalMode	Local mode
Mode	Emulation states (internal use)
PrintLogging	Print logging mode
RollMode	Roll mode
SpaceSuppress	Suppress spaces
SubMode	Sub-emulation mode (read-only)
TypeAheadMode	Type ahead mode

Most of these refer to the equivalent emulation toggles, see the configuration options for details.

All the properties are Booleans except for the following String properties.

The EmulMode property may have the following values:

7 = VIP7700
8 = VIP7800
A = Ansi 3.64
3 = IBM3270
5 = IBM5250
1 = IBM3151
P = Prestel (Viewdata)
Q = DKU7107/7211
2 = DKU7102
T = Minitel (Teletel)
V = VT100/VT220

The SubMode property may have the following values:

C = character mode
E = echo modeF = forms mode
T = text mode
X = TX-RET mode

The CommName may be one of the following:

AV8 Atlantis Bull TSA version 8
ETGX Eicon Bull TGX
FPX2 Cirel FPX / X.25
GATE G&R DNTD Gateway
GLAP G&R LDSA
NABI Eicon ECLAN
NETB NetBIOS (raw)
NETG NetBIOS (G&R)
NONE No interface defined
SH8 Atlantis X.25 V8
SPX SPX/IPX modem gateway
SPXG G&R DNTD Net Gateway
VTI Cirel FPX / VTI via VTI3.DLL
WIND Windows serial port access
WINS Windows socket API

The TCPprotocol is only relevant if the CommName is Windows sockets (WINS) and may be one of the following:

T = Telnet
S = SNI/OSI link
N = NCR/OSI link
6 = DNTD gateway
W = Raw TCP/IP
S = G&R DIWS gateway
D = G&R DSA gateway
R = Rlogin
V = TNVIP
3 = TN3270
5 = TN5250

The Screen Interface

The following Glink.Screen properties are available:

Property	Description
`Col`	Current cursor column position
`Cols`	Max columns (read-only)
`Field() interface`	Interface to field
`FormsMode`	Forms mode (read-only)
`FieldCount`	Number of fields in form (read-only)
`Row`	Current cursor row position
`Rows`	Max rows (read-only)
`ScreenText`	Screen text
`Status interface`	Interface to status line

The following Glink.Screen methods are available:

Methods	Description
`SendKeys`	Sends keys to emulator
`SetScreenText`	Set Screen text

Most of these refer to the equivalent emulation toggles, see the configuration options for details.

The SendKey method

Sends a string to the emulator as if it came from the keyboard

SendKeys (Keys)

Keys: String Keys to send

Return value: none

The string will also be interpreted if the following ^control-character syntax are present:

`^^`	inserts a single ^
`^X`	control character (bottom five bits of the specified character only)
`^#ddd`	decimal specification
`^&ooo`	octal specification
`^$hh`	hexadecimal specification

Using the different possibilities here means that the ASCII CR and ESCAPE control character may be specified in any of the following ways:

`"^M"`, or `"^#013"`, or `"^&015"` or `"^$0D"`
`"^[",` or `"^#027"`, or `"^&033"` or `"^$1B"`

In the same way, the Transmit key can be entered by "^[i", so sending "ABC" + Tab + "123" + Transmit would use the following Keys string:

VBA Example:

```
GL.Screen.SendKeys("ABC^i123^[i")
```

The SetScreenText method

Writes a string to the emulator screen.

SetScreenText (Col, Row, Text)

Col: Long Start Column position
Row: Long Start Row position
Keys: String Text to write on the screen

Return value: none

Text will only be written to the screen if it is valid text for that position of the screen. For none valid characters in the string such as none numeric in a numeric field or writing on a protected position, the character for that position will be ignored and will continue to the next position on the screen for the next character in the string.

The Field Interface

The following Glink.Screen.Field() properties are available:

Property	Description
Currency	Allow currency characters in numeric
Decimal	Allow decimal point in numeric
FieldAttr	Glink binary attribute mask
FieldDataLen	Length of data
FieldIndex	Index in form
FieldLen	Length
FieldOffset	Offset from beginning of screen
FieldText	Field text
FieldX	Start column
FieldY	Start row
Hidden	Invisible
HighIntensity	High-intensity
Justified	Right justified
Minus	Allow minus sign in numeric
Modified	Been modified
MustEnter	Must enter
MustFill	Must fill
Normal	Unprotected and not high-intensity
Numeric	Numeric
Printable	Printable
Fixed	Protected
Signed	Allow Sighed in numeric

`Variable`	Variable

All these properties are read-only apart from the FieldText string property. The attribute properties are booleans, the other field properties are longs.

The Field(i).FieldText property

FieldText: String Field text

writes a string to an unprotected field or reads a string from any field.

Text will only be written to unprotected fields. For invalid characters in the string such as non-numeric in a numeric field, the character for that position will be ignored and will continue to the next position on the field for the next character in the string. The string will be truncated or padded with spaces in necessary.

VBA Example:

```
If Gl.Screen.FormsMode then
 If Gl.Screen.Field(1).Fixed = false then
   Gl.Screen.Field(1).FieldText = "myuserid"
 End If
End If
```

The Status Interface

The following Glink.Screen.Status properties are available:

Property	Description
`StatusText`	Status line text (read-only)

This string property is read-only.

DDE reference

Overview

The Glink DDE interface will transfer data to other applications using the standard Windows text format, OEM format or using the extensions to the Windows character set used internally by Glink. (The Glink character set may be inspected using ALT+F9 when running Glink - it is a superset of the standard set). To use this last character set you must obtain a clipboard format identifier by using the `RegisterClipboardFormat()` call with the name `'GlinkFont'`. The format identifier may then be used in DDE requests in place of `cf_Text`.

An additional clipboard format is available for applications that need to transfer data in a completely transparent way, and is obtained by specifying `'GlinkBinary'` (this must first have been registered in the same way as `'GlinkFont'`). This format does not rely on a null termination character, but uses the first two bytes of the data to specify the length of the rest of the data. This should be specified with the low order byte followed by the high order byte. The `'GlinkBinary'` format may be used both for data requests and for pokes.

System Topic

Glink supports a `System` topic, set up in the normal way with `SysItems`, `Topics` and `Formats` as available items. Note that `SysItems` and `SysPokes` are available in the normal 'application' topic to give a list of currently supported functions.

DDE Initiate

The DDE conversation is initiated in the standard form, using application and topic names. The application name is of course `'GL'`, while the default topic name is `'GLINK'`. The topic name will in general be set to the main window title, which in turn consists of the fixed window title (`GLINK`) optionally followed by the current host name (a hyphen will separate these). The fixed window title may be changed for a specific instance of Glink using the script `TITLE` command. Alternatively you may specify a fixed DDE topic name using either the script `DDENAME` command or the `/DDE` parameter on the command line. In this case, the window title is not used for topic name setting.

In both cases, the topic will be considered to match so long as it matches to the entire length specified by the prospective client. For example, if the window title is `'GLINK - MyHost'` then the DDE conversation may be started using `'GLINK'` as a topic name (in which case more than one GLINK may acknowledge the request) or using the full window title.

In the same way, if `DDENAME` has been used to set a fixed `DDENAME` (say) of `'DDELINK'` then possible matching topics could be `'DDE'`, `'DDEL'` and `'DDELINK'`. A prospective client specifying `'DDELINK GLINK'` however would not be considered.

The topic is always case-insensitive.

Requestable items

Requestable items may be used for cold links, warm links and hot links with the exception of `LinGet` which may only be used for warm and/or hot links.

The following item specifications are supported:

`AtrGet X Y DX DY` Collects attributes from a rectangle on the emulation screen. If the length `DX` is not specified then attributes for the rest of the line will be returned. If the number of rows `DY` is not specified then attributes for one line will be delivered. If the position is not specified then the current cursor position will be used.

Attributes are returned in binary format with 32 bits per character position. The significance of the attribute bits is as for the returned attributes in the `GetFldX` request.

If you wish to read data from the status line, you can do this by specifying a row number of zero (in which case only one line will be delivered).

`AtrTxt X1 Y1 X2 Y2` Collects attributes from the emulation screen in a line-oriented fashion. The position of the first attribute to transfer and the last attribute to transfer should be specified with the command (all four parameters must be specified).

Attributes are returned in binary format with 32 bits per character position. The significance of the attribute bits is as for the returned attributes in the `GetFldX` request.

`CurGet` Returns the current row and column as an ASCII string `'X Y'`.

EtxGet

This request returns the current end of text position as an ASCII string `'X Y'`. If no end of text position is defined then zero will be returned for the Y position and the X value is undefined.

GetFldN field_no

Returns the field definition for the specified field (VIP7700 and VIP7800 modes only). A single parameter specifying the field number is needed; you may use a value of zero to specify the field currently containing the cursor. The data returned is in the same format as for `GetFldX` (see below).

GetFldX X Y

Returns the field definition for the field containing the screen location defined by X and Y (VIP7700 and VIP7800 modes only). The data returned is in the following format:

```
N,X,Y,L,attributes,F
```

where N is the number of the field (counting variable fields only), X is the column in which the field starts, Y is the row containing the field, L is the length of the field, and 'attributes' defines the field attributes for the field. F is a flag indicating whether the emulator is in forms mode (1) or not (0). If the emulator is not in an appropriate mode then only the field number will be delivered, with a value of -1. If a `GetFldN` call is made with a field number larger than the number of fields in the screen, then a field number of zero will be returned. If a `GetFldX` is made for a location inside a protected field then a field number of zero will also be returned but in this case, the field definition will be included.

Attributes are delivered as an eight-character hexadecimal field.

The following attribute bits are defined:

0x00000001	Blue
0x00000002	Green
0x00000004	Red
0x00000008	Low intensity
0x00000010	Blink
0x00000020	Hidden
0x00000040	Inverse
0x00000080	Underline
0x00000200	Double height
0x00000400	Double width
0x00000800	Omit print
0x00001000	Modified
0x00002000	Transmittable
0x00004000	Unprotected
0x00010000	Must Enter
0x00020000	Must Fill
0x00040000	Justify Right
0x00080000	Alphabetic
0x00100000	Numeric
0x00200000	Digit

GetInfo

Returns information about the current state of the emulator. The data is returned in the following format:

```
mode,submode,fields,columns,rows,
x,y,bits,speed,interface,M=flags,
field info
```

The different fields of this information field have the following possible values:

mode	7 = VIP7700
	8 = VIP7800
	A = Ansi 3.64
	3 = IBM3270
	5 = IBM5250
	1 = IBM3151
	P = Prestel (Viewdata)
	Q = DKU7107/7211
	2 = DKU7102

	`T` = Minitel (Teletel) `V` = VT102/VT220
`submode`	`C` = character mode `E` = echo mode `F` = forms mode `T` = text mode `X` = TX-RET mode
`fields`	number of fields in current form (forms mode only)
`columns`	current screen width
`rows`	current screen length
`x`	current cursor column
`y`	current cursor row
`bits`	`7` or `8` depending on whether the connection is 7 or 8-bit
`speed`	current connection speed (where relevant)
`interface`	communications interface (same mnemonics as in script CTYPE command, max four characters)
`M=flags`	String containing a subset of the following letters, each letter identifying a mode that is currently active: `A` = application keypad mode `C` = cursor application mode `F` = auto LF `G` = graphics mode `I` = insert mode `K` = keyboard locked `L` = local mode `N` = PF-keys in numeric pad `P` = print logging active `R` = roll mode `S` = space suppression mode `T` = auto-tab mode `V` = capture active `W` = auto wrap mode `Y` = typeahead mode
`field info`	same information (`N`,`X`,`Y`,`L`,`atts`, `F`) as is returned by `GetFldX`, for cursor position.

A typical string returned from GetInfo could look like this:

```
8,F,12,80,24,34,8,7,9600,WIND,
M=STWY,3,32,8,6,00016000,1
```

This would tell you that Glink was in VIP7800 forms mode, showing a 12-field form on a normal 80x24 screen. The cursor is in column 34 in row 8. The connection is a 7-bit connection at 9600 bps using the Windows serial interface. Space suppression, auto tabbing, auto wrapping and typeahead are in effect. The cursor is in the third field of the form, which starts at column 32 in row 8 and is 6 characters long. The field is unprotected, transmittable, and has the must enter attribute.

LinDat

Returns an indication as to whether there is unreceived data on the line ('1' means data is outstanding, '0' means no data outstanding). If you define a link to this item rather than just request the data, then in the same way as for `LinGet` and `LinProc` data will be marked for delivery to the DDE client and will not be processed by the emulator itself. `LinDat` will in such cases be posted whenever data is available, not when the actual value of `LinDat` changes. Advisements from `LinDat` will continue to be posted so long as there is still unread data available. Linking to `LinDat` is the recommended way of 'hooking' input from the line.

LinGet max_length

(Warm/hot links only; a particular instance of Glink will in addition only support **one** application using this link). Returns advisement messages whenever input is received from the line. When this link is in effect all data from the line is delivered to the DDE client and **not** to the emulator itself.

Unlike a normal link, this one does not deliver advisements when the data has changed but simply whenever there is any data to deliver. We recommend that for secure delivery of data, you define a warm link to the `LinDat` item rather than the `LinGet` item, and issue requests on `LinGet` when the advisement for `LinDat` is received.

`LinProc max_length` Same as `LinGet`, except that when the data is collected it will also be processed by the emulator (i.e. displayed on the emulator screen).

`LinSts` Returns the current connection state of the line. If the emulator is connected '1' is returned, otherwise '0'.

`ScrGet` Returns the current script parameter value if any (if a script is running). The value returned with this command is the same as is set by the script `PARAM` command and a warm/hot link to the script parameter may be used to monitor the current status of a script that may have been initiated via the DDE. Note that if the script terminates then the script parameter no longer has a defined value - in such cases, you should use the `ScrGget` request. You may set the value of the script parameter from your DDE application by using the `ScrPut` poke.

`ScrGget` Returns the current global script parameter value. This may be set in the script using the `GPARAM` command and you may therefore use this with a warm/hot link to return a status from the script being executed (in this case `ScrGet` will not work in that the script parameter disappears once the script has terminated). You may set the value of the global script parameter from your DDE application using the `ScrGput` poke.

`StxGet` This request returns the current start of text position as an ASCII string `'X Y'`. If no start of text position is defined then zero will be returned for the Y position and the X value is undefined.

SysItems This simply provides a list of items that are requestable. The list is separated with tab characters in the normal way.

SysPokes This provides a list of items that are 'pokable', see below.

UvtiGet Returns the current value of the UVTI status item. This item may be set either using the script `SET UVTI` command or using the `UvtiPut` DDE poke. This is provided specifically for the return of status codes from an executing script. To communicate with the script you set up a hot link to the `UvtiGet` item, and use `SET UVTI` commands in the script itself. The advisement will be posted immediately (although the script will continue executing).

Unlike normal items, the `UvtiGet` item will not result in an advisement simply because the item changes. It will only give an advisement when the `SET UVTI` command is executed in the script (even if the value does not change). In particular, a `UvtiPut` command will not result in an advisement.

VduFld N Collects data from the Nth field of the emulation screen. If N is not specified then data is delivered from the field containing the cursor. This command applies only in 7700 and/or 7800 modes.

VduGet X Y DX DY Collects data from a rectangle on the emulation screen. If the length `DX` is not specified then the rest of the line will be returned. If the number of rows `DY` is not specified then one line will be delivered. If the position is not specified then the current cursor position will be used.

If you wish to read data from the status line, you can do this by specifying a row number of zero (in which case only one line will be delivered).

`VduGet` may also be set up automatically using the 'Paste Link' menu item in those applications that support the Paste Link. In this case, to establish a link between a field in the client application you simply mark the area of the screen to be linked, making sure that the 'mark rectangles' option in the Edit menu is checked, select `Edit/Copy` in the Glink window, and then use the Paste Link command in the client application.

`VduTxt X1 Y1 X2 Y2` Collects data from the emulation screen in a line-oriented fashion. The position of the first character to transfer and the last character to transfer should be specified with the command (all four parameters must be specified).

`VduTxt` may also be set up automatically using the 'Paste Link' menu item in those applications that support the Paste Link. In this case, to establish a link between a field in the client application you simply mark the area of the screen to be linked, making sure that 'mark rectangles' is not checked, select `Edit/Copy` in the Glink window, and then use the Paste Link command in the client application.

VidGet X Y DX DY Collects video attributes from a rectangle on the emulation screen (video attributes relate only to the actual appearance of the field on the screen, as opposed to field-oriented attributes). If the length `DX` is not specified then attributes for the rest of the line will be returned. If the number of rows `DY` is not specified then attributes for one line will be delivered. If the position is not specified then the current cursor position will be used.

Attributes are returned in binary format with 16 bits per character position. The significance of the attribute bits is as follows:

0x8000	Inverse
0x4000	Wide line
0x2000	Reserved
0x1000	Sixel graphic
0x0800	Double height (lower)
0x0400	Blink
0x0200	Double width
0x0100	Double height
0x0080	Underline
0x0040	Red background
0x0020	Green background
0x0010	Blue background
0x0008	Bold
0x0004	Red foreground
0x0002	Green foreground
0x0001	Blue foreground

If you wish to read data from the status line, you can do this by specifying a row number of zero (in which case only one line will be delivered).

VidTxt X1 Y1 X2 Y2 Collects video attributes from the emulation screen in a line-oriented fashion (video attributes relate only to the actual appearance of the field on the screen, as opposed to field-oriented attributes). The position of the first attribute to transfer and the last attribute to transfer should be specified with the command (all four parameters must be specified).

Attributes are returned in binary format with 16 bits per character position. The significance of the attribute bits is as for the returned attributes in the `VidGet` request.

`XfrSts`

This request returns the current file transfer status and is typically used in the same way as `LinSts`. Applications may link to the `XfrSts` item before starting a transfer (this should provide a status of 'transfer pending' until the transfer is actually initiated). Advisements from `XfrSts` will be posted as the status of the transfer changes. In the case of a transfer failure additional error codes are provided after the status information; these follow the standard codes for UVTI as defined in the supplied `UVTI.H` header file. Note that if you have 'hooked' the communications line using one of the other DDE requests then you must post the relevant unadvise to unhook it before starting the transfer, or the file transfer will not be correctly processed.

The following values are used for status return:

0 initiating transfer
1 transfer terminated with error (see below)
2 transfer terminated normally
3 file transfer in progress

The additional error codes are only sent in the case when the state is 1. These error codes consist of two values, a major code and a minor code. The actual values of the major and minor code are contained in the `UVTI.H` file; for a full list consult that file. A summary of those that will normally be returned by Glink follows here:

Major	*Minor*	
1 (file)	0	(file or path not found)
	1	(file write-protected or read-only)
	5	(invalid drive specification)
	6	(access denied)
	9	(I/O error reading or writing file)
	10	(unable to open file)
3 (line)	0	(line was disconnected)
5 (protocol)	0	(initial handshake failed)
	1	(max retry count exceeded)
	2	(bad packet format)
10 (op)	0	(operator aborted transfer)
13 (none)	0	(unknown)

Pokable items

The following `POKE` commands are accepted by the Glink DDE:

`KeyPut` Feeds keystrokes to the emulator as though they were entered from the keyboard. This command may not be used to enter data into dialog boxes, only into the main emulator window. The format is also in the 'old' style where you may enter ASCII codes and/or scan codes. Each keystroke must be separated by a space, comma or semicolon. If you wish to enter a scancode as well as an ASCII code you should use the format 'scancode:ascii' or 'scancode/ascii'. The format 'scancode/' with no ASCII code may be used to enter a scancode with no accompanying ASCII code. Both the scancode and the ASCII code may be entered in any of the following formats:

`#n`	(decimal number)
`$n`	(hexadecimal number)
`^x`	(control character)
`x`	(single ASCII character)

For example, to enter `'AT'` plus a carriage return you could use `KeyPut` with `'A T ^M'`. Or to enter the function key `F1` you could use `'#58/'` (58 is the decimal value of the scancode for the `F1` key).

`LinBrk` Sends a break on the communications interface.

`LinCon` Initiates a connection using the current interface. The data accompanying the command specifies the host name. If there is a currently active connection then this will be terminated first.

`LinDis` Disconnects the current communications interface.

`LinPrg` Purges the communications read buffer, discarding any characters that may have been received.

LinPut The data accompanying this command will be sent directly to the communications interface irrespective of the current mode of the emulator. The data will not be 'terminated' (with CR, ETX or EOT) unless the relevant character is specified as part of the data. In that the termination character is affected by the emulator state, communications interface, and Glink configuration we recommend that you use the `LinXmt` command rather than the `LinPut` command when termination is required.

LinXmt Sends the accompanying data with the correct data terminator to the host machine.

PosCur Positions the cursor. The data accompanying the command should be the ASCII X and Y coordinates to position to, separated with a space.

ScrGput Inserts a value into the global script parameter; this may be retrieved in the script itself using the `$GPARAM` and/or `$Gn` built-in variables. The value of the global script parameter may be retrieved in your DDE application using the `ScrGget` request (either using a DDE request or by asking for an advisement).

Script Starts the script with the name specified in the accompanying data. Control is returned to the calling application immediately; if you require to monitor the progress of the script you should use a hot link to the `ScrGet` item and insert `PARAMETER` commands at appropriate places in the script file.

Simple script commands may be executed using the normal '=' prefix as in the DOS version (however, see below under 'Executing Commands').

ScrPut Inserts a value into the script parameter; this may be retrieved in the script itself using the `$PARAMETER` and/or `$Pn` built-in variables. The value of the script parameter may be retrieved in your DDE application using the `ScrGet` request (either using a DDE request or by asking for an advisement).

UvtiPut The data accompanying this command should be an ASCII numeric between zero and 255, and will be placed in the UVTI status (requestable with the `UvtiGet` request).

VduEmu This is exactly the same as `VduPut`, with the exception that whereas `VduPut` treats the data as though it had been typed, `VduEmu` treats the data as though it had been received over the line interface.

VduPut The data accompanying the command is processed according to the current emulator state. For example if Glink is currently in ECHO mode, the data will be sent to the communications interface only, while if Glink is in FORMS mode then the data will only be presented on the screen.

Xabort This poke simply aborts any file transfer or dialing operation that is in progress.

Executing commands

A command interface to support the DDE execute function is supplied. This does not however follow the normal syntax but specifies script commands for immediate execution. Caution should be exercised when using the HALT script command, as the DDE conversation will in this case disappear. If you wish to execute a complete script then the Script poke may be used.

Visual Basic Examples

The online help in Glink/Windows contains an example that you may use to help get you started with your own programs. Here we will include another example that may give you some ideas as to how you can use the DDE verbs. Many thanks to Ken Shepherd for giving us permission to use this code.

```
    '
    '   This VisualBasic program uses DDE to capture system information from
    '   the GCOS8 TSS VIDY system monitor.  VIDY can be invoked by any
    '   TSS user who enters "vidy" at the TSS prompt.
    '   This code contains other examples of using DDE to perform
    '   various functions with Glink.

    Global masslock As Integer

    Global vidyline$
    Global vidylnum% ' 1..24, initially zero


    Global sysname$  ' system name
    Global sysproc%  ' number of active processors
    Global sysmemk&  ' system available memory in Kwords

    Global jobmore%  ' 1=**MORE** in job column

    Global gcosmemk& ' GCOS memory used in Kwords
    Global gcoscp%   ' GCOS percent of processor used
    Global idlememk& ' idle memory in the system in Kwords

    Global idlecp%   ' percent of system idle processor


VERSION 2.00
Begin Form ddemain
    Caption          =  "Glink DDE Visual Video"
    ClientHeight     =  5415
    ClientLeft       =  1560
    ClientTop        =  1695
    ClientWidth      =  7425
    FontBold         =  -1  'True
    FontItalic       =  0   'False
    FontName         =  "MT Extra"
    FontSize         =  8.25
    FontStrikethru   =  0   'False
    FontUnderline    =  0   'False
    Height           =  6105
    Left             =  1500
    LinkTopic        =  "Form1"
    ScaleHeight      =  5415
    ScaleWidth       =  7425
    Top              =  1065
    Width            =  7545
    Begin CommonDialog CMDialog1
        CancelError      =  -1  'True
        Left             =  6120
        Top              =  120
    End
    Begin TextBox viewhost
        FontBold         =  0   'False
        FontItalic       =  0   'False
        FontName         =  "Terminal"
        FontSize         =  9
```

```
    FontStrikethru   =  0   'False
    FontUnderline    =  0   'False
    Height           =  735
    Left             =  240
    MultiLine        =  -1  'True
    ScrollBars       =  3   'Both
    TabIndex         =  6
    Top              =  2640
    Width            =  7095
End
Begin TextBox moretext
    Alignment        =  2   'Center
    Height           =  375
    Left             =  1560
    TabIndex         =  5
    Top              =  600
    Width            =  615
End
Begin CommandButton sendbutton
    Caption          =  "Send"
    Height           =  495
    Left             =  6120
    TabIndex         =  4
    Top              =  600
    Width            =  1215
End
Begin TextBox recvstuff
    Height           =  1215
    Left             =  240
    MultiLine        =  -1  'True
    ScrollBars       =  3   'Both
    TabIndex         =  3
    Top              =  1320
    Width            =  7095
End
Begin TextBox worktext
    Height           =  375
    Left             =  2280
    TabIndex         =  2
    Top              =  600
    Width            =  3735
End
Begin CommandButton breakbutton
    Caption          =  "Break"
    Height           =  495
    Left             =  240
    TabIndex         =  1
    Top              =  600
    Width            =  1215
End
Begin TextBox sendstuff
    Height           =  375
    Left             =  240
    TabIndex         =  0
    Top              =  120
    Width            =  7095
End
Begin Label Label5
    Alignment        =  2   'Center
    Caption          =  "GCOS8"
    Height           =  255
    Left             =  3960
    TabIndex         =  18
    Top              =  3720
    Width            =  1095
End
Begin Label Label4
    Alignment        =  2   'Center
```

```
    Caption          =  "Idle"
    Height           =  255
    Left             =  2760
    TabIndex         =  17
    Top              =  3720
    Width            =  1095
End
Begin Label Label3
    Alignment        =  1  'Right Justify
    Caption          =  "Proc"
    Height           =  495
    Left             =  240
    TabIndex         =  16
    Top              =  4680
    Width            =  1095
End
Begin Label Label2
    Alignment        =  1  'Right Justify
    Caption          =  "Memory"
    Height           =  495
    Left             =  240
    TabIndex         =  15
    Top              =  4080
    Width            =  1095
End
Begin Label Label1
    Alignment        =  1  'Right Justify
    Caption          =  "System"
    Height           =  495
    Left             =  240
    TabIndex         =  14
    Top              =  3480
    Width            =  1095
End
Begin Label gcp
    BorderStyle      =  1  'Fixed Single
    Height           =  495
    Left             =  3960
    TabIndex         =  13
    Top              =  4680
    Width            =  1095
End
Begin Label gmemk
    BorderStyle      =  1  'Fixed Single
    Height           =  495
    Left             =  3960
    TabIndex         =  12
    Top              =  4080
    Width            =  1095
End
Begin Label icp
    BorderStyle      =  1  'Fixed Single
    Height           =  495
    Left             =  2760
    TabIndex         =  11
    Top              =  4680
    Width            =  1095
End
Begin Label imemk
    BorderStyle      =  1  'Fixed Single
    Height           =  495
    Left             =  2760
    TabIndex         =  10
    Top              =  4080
    Width            =  1095
End
Begin Label sproc
    BorderStyle      =  1  'Fixed Single
```

```
        Height          =  495
        Left            =  1440
        TabIndex        =  9
        Top             =  4680
        Width           =  1215
    End
    Begin Label smemk
        BorderStyle     =  1  'Fixed Single
        Height          =  495
        Left            =  1440
        TabIndex        =  8
        Top             =  4080
        Width           =  1215
    End
    Begin Label sname
        BorderStyle     =  1  'Fixed Single
        Height          =  495
        Left            =  1440
        TabIndex        =  7
        Top             =  3480
        Width           =  1215
    End
    Begin Menu mnuFile
        Caption         =  "File"
        Begin Menu mnurunscr
            Caption         =  "Run script..."
        End
        Begin Menu mnuclose
            Caption         =  "Exit"
        End
    End
    Begin Menu mnuGlink
        Caption         =  "Glink"
        Begin Menu mnuConn
            Caption         =  "Connect"
        End
        Begin Menu mnuHalt
            Caption         =  "Halt"
        End
    End
End


Sub breakbutton_Click ()
    ' Come here when the commandbutton named "breakbutton" is pressed
    ' on the main form. This uses the DDE LinBrk LinkPoke function
    ' to send a break to the host.
    worktext.LinkTopic = "GL|GLINK"
    worktext.LinkItem = "LinBrk"
    worktext.LinkMode = COLD
    worktext.Text = "Send Break"
    worktext.LinkPoke
    sendstuff.Text = ""
    sendstuff.SetFocus
End Sub

Sub Form_Load ()
    ' This procedure gets invoked when the main form of the application
    ' is loaded.  It attempts to establish a hot link to Glink.  If this
    ' fails with the error DDE_NO_APP, meaning the target application was
    ' not currently running, then this procedure starts Glink and reattempts
    ' to establish the hot link.  See the subroutine recvstuff_change which
    ' asynchronously handles arrival of data from the host.
    Const DDE_NO_APP = 282, NONE = 0, COLD = 2, HOT = 1
    On Error GoTo startit
    vidyline$ = ""
    vidylnum% = 0
    jobmore% = 0
```

```
r1:
    recvstuff.LinkTopic = "GL|GLINK"
    recvstuff.LinkItem = "LinProc 5000"      ' Allow up to 5000 characters
    recvstuff.LinkMode = NONE                ' from the host per event.
    recvstuff.LinkMode = HOT
    recvstuff.LinkRequest
    Exit Sub
startit:
    If Err = DDE_NO_APP Then
        t% = Shell("\gl50e\gl.exe /DDE GLINK", 1)
        zapp% = DoEvents()
        GoTo r1
    Else
        MsgBox "Error starting DDE:  " + Str$(Err) + Error$
        Stop
    End If
End Sub

Sub mnuclose_Click ()
    ' Come here when menu File/Exit is selected.
    ' This ends this application.
    End
End Sub

Sub mnuConn_Click ()
    ' Come here when menu Glink/Connect is selected.
    ' This issues a connect via LinCon LinkPoke.
    Const NONE = 0, HOT = 1, COLD = 2
    worktext.LinkTopic = "GL|GLINK"
    worktext.LinkItem = "LinCon"
    worktext.LinkMode = NONE
    worktext.LinkMode = COLD
    worktext.Text = ""
    worktext.LinkPoke
End Sub

Sub mnuHalt_Click ()
    ' Come here when menu Glink/Halt is selected.
    ' This terminates the Glink emulator by executing the Halt
    ' script function via LinkExecute.
    On Error Resume Next
    If worktext.LinkMode <> NONE Then
        worktext.LinkExecute "HALT"
    End If
End Sub

Sub mnurunscr_Click ()
    ' Come here when menu File/Run script is selected.
    ' This uses the common dialog facility to obtain the path of a
    ' Glink script to launch via the DDE Script LinkPoke.
    cmdialog1.InitDir = "\"
    cmdialog1.Filter = "Script file (*.scr)|*.scr"
    cmdialog1.FilterIndex = 1
    cmdialog1.CancelError = -1
    On Error Resume Next
    cmdialog1.Action = 1 ' open file
    If Err <> 0 Then
        If Err = 32755 Then Exit Sub  ' cancel clicked
        MsgBox "Open Error " + Str$(Err) + Error$
        Exit Sub
    End If
    MsgBox "Opened " + cmdialog1.Filename, 64
    Close
    worktext.LinkMode = 0
    worktext.LinkTopic = "GL|GLINK"
    worktext.LinkItem = "Script"
    worktext.Text = cmdialog1.Filename
    worktext.LinkMode = 1
```

```
    worktext.LinkPoke
End Sub

Sub recvstuff_Change ()
    ' This subroutine is called each time new data arrives from the
    ' host.  The textbox recvstuff is hot-linked to Glink via LinProc.
    Static okdone%
    If okdone% = -1 Then
        MsgBox "oops!" + recvstuff.Text
        Exit Sub
    End If
    okdone% = -1  ' do not allow reentry while looking at dde
    xcn$ = viewhost.Text
    recvlen% = Len(recvstuff.Text)
    For i% = 1 To Len(recvstuff.Text)
        ch$ = Mid$(recvstuff.Text, i%, 1)
        If ch$ = Chr$(10) Then  ' linefeed
            '  force new line in our display textbox
            xcn$ = xcn$ + Chr$(13) + Chr$(10)
            lvideo% = InStr(1, vidyline$, "   VIDEO- ")
            If lvideo% > 0 Then
                v$ = Mid$(vidyline$, lvideo%)
                'vidyline$ = v$
                sname.Caption = Mid$(v$, 10, 6)
                sproc.Caption = Mid$(v$, 22, 1)
                sysproc% = Val(sproc.Caption)
                smemk.Caption = Mid$(v$, 31, 7)
                sysmemk& = Val(smemk.Caption)
                vidylnum% = 1
            Else
                vidylnum% = vidylnum% + 1
                If Mid$(vidyline$, 19, 8) = "  IDLE- " Then
                    imemk.Caption = Mid$(vidyline$, 29, 6)
                    icp.Caption = Mid$(vidyline$, 36, 3)
                End If
            End If
            vidyline$ = ""
        ElseIf ch$ = Chr$(13) Then ' carriage return
            ' xcn$ = xcn$ + ch$ ' "\015" + Chr$(13)
        ElseIf ch$ >= " " And ch$ < Chr$(127) Then
            xcn$ = xcn$ + ch$
            vidyline$ = vidyline$ + ch$
        ElseIf ch$ <> Chr$(127) Then
            xcn$ = xcn$ + "\" + Oct$(Asc(ch$)) + "&"
        End If
    Next i%
    If Mid$(vidyline$, 19, 8) = "  GCOS- " Then
        gmemk.Caption = Mid$(vidyline$, 29, 6)
        gcp.Caption = Mid$(vidyline$, 36, 3)
    End If
    'viewhost.text = xcn$ + "<" + Str$(recvlen%) + ">"
    okdone% = 0
End Sub

Sub sendbutton_Click ()
    ' Come here when the commandbutton named sendbutton is pressed on
    ' then main form.  This uses the DDE LinXmt LinkPoke function
    ' to send the text in the textbox sendstuff to the host.
    viewhost.Text = ""
    sendstuff.LinkTopic = "GL|GLINK"
    sendstuff.LinkItem = "LinXmt"
    sendstuff.LinkMode = COLD
    sendstuff.LinkPoke
    sendstuff.Text = ""  ' clear the input
    sendstuff.SetFocus
End Sub
```

HLLAPI reference

This section does *not* attempt to describe the actual HLLAPI interface; for that information refer to the official HLLAPI documentation supplied by IBM.

Files relating to HLLAPI

The following files are delivered with Glink for Windows to provide HLLAPI support.

File name	Description
HLLAPI32.DLL	The Dynamic Link Library that precompiled HLLAPI applications will use at run time. It must reside somewhere in the PATH but will most usually be found in the \WINDOWS or \WINDOWS\SYSTEM directory.
HLLAPI32.LIB HLL32VC.LIB	The library file defining all HLLAPI functions which is needed for linking HLLAPI applications that have been written in C/C++. The file is only required for programming in C directly to the HLLAPI specifications. The file will normally reside in your C library directory. The HLLAPI32.LIB is for use with Borland BC++, the HLL32VC.LIB is for use with the Microsoft VC++.
GLHLLAPI.INI	The configuration file which will be read by the HLLAPI when it is started. See the section below describing the contents of this file.
WHLLAPI.H	The header file that must be used for compiling HLLAPI applications that are written in C/C++. The file is only needed by developers coding in C directly to the HLLAPI specifications.

File name	Description
HLLAPI.BAS	This is the definition file to be included into Visual Basic projects. The file is only needed by developers coding in VB directly to the HLLAPI specifications. This file contains a few utility functions to simplify programming.
WHLLAPI.PAS	Provides the same capabilities for those programmers using Borland Pascal 7.0, Turbo Pascal for Windows or Delphi. Just compile the Pascal source with the compiler you are using..
HLLDB32.DLL	is a utility that can help you debug your HLLAPI applications. See 'Debugging HLLAPI applications'.

HLLAPI installation

Which files you will need to install will depend whether you are going to be programming your own applications directly to the HLLAPI specifications or simply with pre-packaged HLLAPI products.

We will here assume that you will only be using Glink with pre-packaged HLLAPI products.

1. Copy HLLAPI32.DLL and GLHLLAPI.INI to your Windows directory.

Modify the \WINDOWS\GLHLLAPI.INI file according to the instructions provided in the file itself. The general parameters there should be straightforward enough. Each HLLAPI short name will correspond with a Glink configuration file that's set up with the appropriate setup for the host in question.

The GLHLLAPI.INI configuration file

GLHLLAPI.INI is used (in the same way as WIN.INI) by the Glink HLLAPI32.DLL to determine runtime parameters and to obtain the exact configuration to be used by Glink for each 'Short Name' used by the HLLAPI application. Each host parameter is specified in short name host-specific sections. The file must reside in your \WINDOWS directory. An example of a GLHLLAPI.INI file follows here:

```
[Configuration]
GlinkDir=c:\glwin
GlinkParameters=/R72

[Hllapi32]
DebugFile='HLLAPI32.DB'
Debugging=OFF
DebugAppend=OFF
HllapiDll=hllapi32.dll
PureDDE=FALSE
DisableHide=FALSE

[Default]
;[shortname]
;Name=glinktitle
;Config=filename
;ConnectCommand=command
;ConnectScript=name
;DisconnectCommand=command
;DisconnectScript=name
;Transfer=transfer_type

[A]
Name=ibmhost.ip.adr
Config=ib1.glinkconfig

[A]
Name=ibmhost.ip2.adr
Config=ib2.glinkconfig;GCOS7 Microfit/Ftran

;[transfername]
;e.g. [MYFTRAN]
;Dialog=Trnline "FTRAN MICROSYS;"
;Server=Emulate "^[401C^[6C"
;SendText=Trnline "FTRAN MICROSYS;";Putfile FTRAN "%p;%h"
;SendBinary=Trnline "FTRAN MICROSYS;";Putfile FTRAN "%p;%h"
;ReceiveText=Trnline "FTRAN MICROSYS;";Getfile FTRAN "%p;%h"
;ReceiveBinary=Trnline "FTRAN MICROSYS;";Getfile FTRAN "%p;%h"
```

The [Configuration] section

This specifies general information needed to start up the initial copy of Glink for Windows.

GlinkDir=directoryname

This specifies the directory on which the `GL.EXE` program resides. It is not needed if this directory is included in your PATH.

GlinkParameters=parameters

This specifies any command line options you wish to be used when Glink for Windows is started.

The [Hllapi32] section

This specify debugging information needed for the HLLAPI DLL.

DebugFile=filename

This specifies the name of the file to which debugging output will be sent if debugging is enabled (see the next option). If debugging is enabled without a `DebugFile` directive then output will be sent to `HLLAPI.DB` in the current directory.

Debugging=ON

When turned ON, this enables debug tracing of HLLAPI functions. For this to happen the `HLLAPI32.DLL` you would otherwise have used must reside in your Windows directory as `GLHLL32.DLL`, and the `HLLDB32.DLL` supplied in this release copied to `HLLAPI32.DLL` so as to intercept the HLLAPI calls. Output will be sent to the file specified with the `DebugFile` option.

DebugAppend=ON

When turned ON, this tells the debug routines to append output to the end of the existing debug file rather than overwrite it.

HllapiDll=GLHLL.DLL

This specifies the name of the 'real' HLLAPI DLL to load. In most cases, you will have renamed it to GLHLL32.DLL and the HLLAPI DLL will be the debugging DLL.

PureDDE=TRUE

When set to TRUE, this tells the HLLAPI modules to use only DDE when communicating with Glink. Normally the most common functions will communicate directly with Glink using Windows messages but in some circumstances (for example when using Network DDE) it may be desirable to force the use of DDE for all operations.

DisableHide=TRUE

When set to TRUE, this tells the HLLAPI functions never to hide the Glink window. This can be useful when setting up and testing a HLLAPI application so as to be able to visually follow the execution of the host dialog.

The [shortname] sections

The HLLAPI 'shortname' is a letter from A-Z. The first site must be ***[A]***, the second ***[B]*** and so on.

Inside the site-specific sections (and the `[Default]` section, which supplies values that will apply for all connections) the following directives may be used:

Directive	Description
`Name`	This is the text found in the Glink title bar (minus the "GLINK – " part) when the config file is loaded. This allows HLLAPI to connect to active Glink windows. E.g. If the Glink title was "GLINK – myibm.host.com", then this directive would be Name=myibm.host.com
`Config`	The configuration file to use for this connection name. Normally not required because it has already been specified in the [Connections] section, but provided for those hosts that do not appear there.
`ConnectCommand`	The script command to use to connect to the host. Appropriate commands are `NETCONNECT` (default), `CONNECT`, `DIAL`, and `MANDIAL`.
`ConnectScript`	The script file to run when the initial connection to the host is made. If specified then it will override any `ConnectCommand` directive that might have been specified.
`Disconnect-Command`	Specifies a script command that will be executed at disconnect time.
`Disconnect-Script`	Specifies a script to run at disconnect time. If present this directive overrides any `DisconnectCommand` directive that might have been specified.

Directive	Description
`Transfer`	Specifies the file transfer type for this connection. Valid values for this are: GKRMGCOS8 Gkrm FTRAN8 GCOS8 Ftran FTRAN GCOS7 Ftran KERMIT GCOS6 Kermit GKERM GCOS6 G&R Gkerm INDMVS IBM MVS/TSO IND$FILE INDCICS IBM CICS/VS IND$FILE INDVM IBM VM/CMS IND$FILE transfername GLHLLAPI.INI-defined transfer type; see the [transfername] section below for details.

The [transfertype] sections

A separate [transfername] section may be created for specific file transfer needs with the host. Normally these sections are not required because the predefined GKRM, FTRAN8, FTRAN, KERMIT, GKERM, INDMVS, INDCICS and INDVM are preconfigured in the HLLAPI DLL.

HLLAPI will first look in this file for the file transfer script commands before using the ones from the HLLAPI DLL.

The special characters %p and %h will be replaced by the PC filename and the Host filename respectively as supplied by the HLLAPI application.

The parameters in each section are:

Directive	Description
`Dialog`	Specifies the script commands to start the Glink File Transfer menu.

Directive	Description
Server	Specifies the script commands to start Glink in Server mode
SendText	Specifies the script commands to execute to initiate a text file transfer to the host.
SendBinary	Specifies the script commands to execute to initiate a binary file transfer to the host.
ReceiveText	Specifies the script commands to execute to initiate a text file transfer from the host.
ReceiveBinary	Specifies the script commands to execute to initiate a binary file transfer from the host.

Below is an example of a [transfername] section. It is in fact the same as the predefined FTRAN transfer type and therefore not really needed. However, to use it you could specify Transfer=Microfit in the [shortname] section for the relevant connection.

```
;GCOS7 Microfit/Ftran
[Microfit]
Dialog=Trnline "FTRAN MICROSYS;"
Server=Emulate "^[401C^[6C"
SendText=Trnline "FTRAN MICROSYS;";Putfile FTRAN "%p;%h"
SendBinary=Trnline "FTRAN MICROSYS;";Putfile FTRAN "%p;%h"
ReceiveText=Trnline "FTRAN MICROSYS;";Getfile FTRAN "%p;%h"
ReceiveBinary=Trnline "FTRAN MICROSYS;";Getfile FTRAN "%p;%h"
```

Debugging HLLAPI applications

The HLLDB32.DLL file can help in debugging HLLAPI applications. To use this utility, rename the HLLAPI32.DLL file in your Windows directory to GLHLL32.DLL. Copy the HLLDB32.DLL file to HLLAPI32.DLL in the Windows directory. This DLL will intercept calls to the HLLAPI and, if debugging is enabled with 'Debugging=On' in the [Hllapi32] section of the GLHLLAPI.INI file, will produce a trace of all HLLAPI calls and events on the file specified with 'DebugFile=filename' in the same section.

UVTI reference

This section does *not* attempt to describe the actual UVTI interface; for that information refer to the official UVTI documentation supplied by Bull and the on-line help file supplied with Glink.

Files relating to UVTI

The following files are delivered with Glink for Windows to provide UVTI support.

File name	Description
`UVTI32.DLL`	The Dynamic Link Library that precompiled UVTI applications will use at run time. Affinity Visual will also use this file. It must reside somewhere in the DOS PATH but will most usually be found in the `\WINDOWS` or `\WINDOWS\SYSTEM` directory.
`UVTI32.LIB`, `UVTI32VC.LIB`	The library file defining all UVTI functions which is needed for linking UVTI applications that have been written in C. The file is in practice identical to the file of the same name supplied with Affinity UVTI and indeed may be used interchangeably with that file. The file is only required for programming in C directly to the UVTI specifications. Applications using Affinity Visual do not require this file. The file will normally reside in your C library directory. The UVTI32.LIB is for use with Borland BC++, the UVTI32VC.LIB is for use with the Microsoft VC++.

File name	Description
GLUVTI32.LIB	An alternate library file that may be used if you are developing C applications for Glink/UVTI that must be able to coexist with Affinity/UVTI applications. To do this you will need to rename the UVTI32.DLL supplied with Glink to GLUVTI32.DLL (leaving the Affinity UVTI32.DLL) in place) and link your applications with GLUVTI32.LIB instead of UVTI32.LIB. The file will normally reside in your C library directory.
GLUVTI.INI	The configuration file which will be read by the UVTI when it is started. See the section below describing the contents of this file.
UVTI.H	The header file that must be used for compiling UVTI applications that are written in C. The file is in practice identical to the file of the same name provided with the Affinity UVTI, and may be used interchangeably with that file. The file is only needed by developers coding in C directly to the UVTI specifications. Applications using Affinity Visual do not require this file.
UVTI.HLP	This is the on-line UVTI help file. This file is the property of, and copyrighted by Groupe Bull.
UVTILIB.PAS	Provides the same capabilities for those programmers using Borland Pascal 7.0, Turbo Pascal for Windows or Delphi. Just compile the Pascal source with the compiler you are using. If you need to use Affinity emulations in coexistence with Pascal UVTI applications then rename the UVTI32.DLL supplied with Glink to GLUVTI32.DLL in the same way as for C applications, and replace the appropriate name in the UvtiModuleName constant.
UVTIDB32.DLL	is a utility that can help you debug your UVTI applications. See 'Debugging UVTI applications'.

UVTI installation

Which files you will need to install will depend on whether you are going to use both Glink and the Affinity emulator on your system, and also whether you are going to be programming your own applications directly to the UVTI specifications (as opposed to using pre-packaged products such as Affinity Visual).

We will here assume that you will only be using Glink; details of how to install a dual-function setup are supplied both in the online help and in the descriptions of the files themselves supplied above.

1. Copy `UVTI32.DLL` and `GLUVTI.INI` to your Windows directory.

Modify the `\WINDOWS\GLUVTI.INI` file according to the instructions provided in the file itself. The general parameters there should be straightforward enough, but you will also need to provide a list of `[Connections]` that can be addressed from UVTI applications. Each connection name will correspond with a Glink configuration file that's set up with the appropriate setup for the host in question.

The GLUVTI.INI configuration file

`GLUVTI.INI` is used by the Glink `UVTI32.DLL` to determine runtime parameters and to obtain the exact configuration to be used by Glink for each 'Connection Name' used by the UVTI application. Each host is listed with its configuration file in the `[Connections]` section and any additional parameters that may be required are then specified in separate host-specific sections. The file must reside in your `\WINDOWS` directory. An example of a `GLUVTI.INI` file follows here:

```
[Configuration]
GlinkDir=c:\glwin
GlinkParameters=/R72
DebugFile='GLUVTI.DB'
Debugging=OFF
DebugAppend=OFF
DebugSysTime=OFF
PureDDE=FALSE
DisableHide=FALSE
```

```
[Connections]
;sitename=config
LanHost=lan.glinkconfig
LanHostS=lan.glinkconfig
Modem=asy.glinkconfig
ModemDial=asy.glinkconfig
Dps6DNTD=dnt.glinkconfig

[Default]
;[sitename]
;Config=filename
;ConnectCommand=command
;ConnectScript=name
;DisconnectCommand=command
;DisconnectScript=name
;Transfer=transfer_type
;LineStatus=CHECK
;PF_pres=presentation_type
;LK_comm=comms_type
;WaitETX=false
;ReturnETX=false
;RawData=false
;RecWait=0
;GlinkVer=540

[LanHost]
ConnectCommand=NETCONNECT 192.150.211.2
DisconnectCommand=NETDISCONNECT

[LanHostS]
ConnectCommand=NETCONNECT 192.150.211.2
DisconnectCommand=NETDISCONNECT

[ModemDial]
ConnectCommand=NETCONNECT;MANDIAL 22416588
LineStatus=NOCHECK

[DPS6DNTD]
ConnectScript=LOGIN.SCR MyDPS6
Transfer=GKERM
LK_comm=LK_DNTD

;[transfername]
;e.g. [MYFTRAN]
;Dialog=Trnline "FTRAN MICROSYS;"
;Server=Emulate "^[401C^[6C"
;SendText=Trnline "FTRAN MICROSYS;";Putfile FTRAN "%p;%h"
;SendBinary=Trnline "FTRAN MICROSYS;";Putfile FTRAN "%p;%h"
;ReceiveText=Trnline "FTRAN MICROSYS;";Getfile FTRAN "%p;%h"
;ReceiveBinary=Trnline "FTRAN MICROSYS;";Getfile FTRAN "%p;%h"
```

The [Configuration] section

This specifies general information needed to start up the initial copy of Glink for Windows.

GlinkDir=directoryname

This specifies the directory on which the `GL.EXE` program resides. It is not needed if this directory is included in your PATH.

GlinkParameters=parameters

This specifies any command line options you wish to be used when Glink for Windows is started.

DebugFile=filename

This specifies the name of the file to which debugging output will be sent if debugging is enabled (see the next option). If debugging is enabled without a `DebugFile` directive then output will be sent to `GLUVTI.DB` in the current directory.

Debugging=ON

When turned ON, this enables debug tracing of UVTI functions. For this to happen the `UVTI32.DLL` you would otherwise have used must reside in your Windows directory as `GLUVTI32.DLL`, and the `UVTIDB32.DLL` supplied in this release copied to `UVTI32.DLL` so as to intercept the UVTI calls. Output will be sent to the file specified with the `DebugFile` option.

DebugAppend=ON

When turned ON, this tells the debug routines to append output to the end of the existing debug file rather than overwrite it.

DebugSysTime=OFF

When turned ON, this tells the debug routines to log times in milliseconds since Windows was started rather than the default, which is to log times in milliseconds from the first call made to UVTI.

PureDDE=TRUE

When set to TRUE, this tells the UVTI modules to use only DDE when communicating with Glink. Normally the most common functions will communicate directly with Glink using Windows messages but in some circumstances (for example when using Network DDE) it may be desirable to force the use of DDE for all operations.

DisableHide=TRUE

When set to TRUE, this tells the UVTI functions never to hide the Glink window. This can be useful when setting up and testing a UVTI application so as to be able to visually follow the execution of the host dialog.

The [Connections] section

This section provides a list of all the names to which UVTI applications will be able to connect. One directive is needed for each name, specifying the name of the configuration file associated with the name. Each name in this section will additionally have a section using the site name as its heading further on in the file. Such additional sections may also be specified for site names *not* listed in the `[Connections]` section. You will be able to connect to such names, but they will not be listed in the dialog box presented to the user when the GetConnName() routine is called by the UVTI application.

Inside the site-specific sections (and the `[Default]` section, which supplies values that will apply for all connections) the following directives may be used:

Directive	Description
`Config`	The configuration file to use for this connection name. Normally not required because it has already been specified in the [Connections] section, but provided for those hosts that do not appear there.
`ConnectCommand`	The script command to use to connect to the host. Appropriate commands are `NETCONNECT` (default), `CONNECT`, `DIAL`, and `MANDIAL`.

Directive	Description
`ConnectScript`	The script file to run when the initial connection to the host is made. If specified then it will override any `ConnectCommand` directive that might have been specified.
`Disconnect-Command`	Specifies a script command that will be executed at disconnect time.
`Disconnect-Script`	Specifies a script to run at disconnect time. If present this directive overrides any `DisconnectCommand` directive that might have been specified.
`LineStatus`	Specifies whether the real line status should be returned to the UVTI application (specified as CHECK) or only a good status (specified as NOCHECK). NOCHECK may be needed on async interfaces because of bugs in the Windows communication drivers that report this status incorrectly. The default is CHECK.

`Transfer`	File transfer type. Valid values here are:
	`GKRM (default) (GCOS8 GKRM)`
	`FTRAN8 (GCOS8 FTRAN)`
	`FTRAN (GCOS7 FTRAN)`
	`KERMIT (GCOS6 Kermit)`
	`GKERM (GCOS6 G&R GKERM)`
	`INDMVS (IBM MVS/TSO IND$FILE)`
	`INDCICS (IBM CICS/VS IND$FILE)`
	`INDVM (IBM VM/CMS IND$FILE)`
	`transfername` GLUVTI.INI-defined transfer type; see the [transfername] section below for details.

`PF_pres`	The presentation type to report to any UVTI application needing the information (this is because the emulations supported by Glink don't match the available values in UVTI.H). Possible values are PF_NULL, PF_HDS7 (the default), PF_P200, PF_VT320, PF_7102, and PF_VDX.
`LK_comm`	In the same way as PF_pres, this specifies the comms type to report to the UVTI application. Possible values are LK_VIP, LK_TGX25, LK_TM, LK_SDLC, LK_SNAX25, LK_DNTD, LK_ASYNC (the default) and LK_PAD.

Directive	Description
`WaitETX`	Specifies that the UVTI interface defer delivery until receipt of ETX rather than ETB (default is FALSE). Valid only for synchronous types of interface (LK_VIP, LK_TGX25, etc).
`ReturnETX`	Specifies that received ETX characters should be delivered to the application (the default FALSE will strip these). Valid only for synchronous types of interface (LK_VIP, LK_TGX25, etc).
`RawData`	Specifies that all received characters be delivered to the application. Valid only for synchronous types of interface (LK_VIP, LK_TGX25, etc).
`RecWait`	Specifies the time in tenths of a second that UVTI will wait for more incoming data before returning to the application. This parameter is used for UVTI applications which expect to receive complete blocks of data, but don't allow slow lines (dial-up, Ggate over PPP, etc) enough time for the complete block to be received. The default value is zero except if a UVTIReceive timeout of -1 is used on asynchronous types of interface and Glink is in a TEXT, FORM or TX-RET mode, in which case a wait of 5 tenths of a second is forced. For example, if RecWait=10 is used, all UVTIReceives will wait 1 second for more incoming data before returning.
`GlinkVer`	This allows backward compatibility between versions. The version number 'nnn' must be the 3 digits of a Glink version, e.g. 520, 521, 530, 540. This option should only be needed for some old UVTI applications.

The [transfertype] sections

A separate [transfername] section may be created for specific file transfer needs with the host. Normally these sections are not required because the predefined GKRM, FTRAN8, FTRAN, KERMIT, GKERM, INDMVS, INDCICS and INDVM are preconfigured in the UVTI DLL.

UVTI will first look in this file for the file transfer script commands before using the ones from the UVTI DLL.

The special characters %p and %h will be replaced by the PC filename and the Host filename respectively as supplied by the UVTI application.

The parameters in each section are:

Directive	Description
`Dialog`	Specifies the script commands to start the Glink File Transfer menu.
`Server`	Specifies the script commands to start Glink in Server mode
`SendText`	Specifies the script commands to execute to initiate a text file transfer to the host.
`SendBinary`	Specifies the script commands to execute to initiate a binary file transfer to the host.
`ReceiveText`	Specifies the script commands to execute to initiate a text file transfer from the host.
`ReceiveBinary`	Specifies the script commands to execute to initiate a binary file transfer from the host.

Below is an example of a [transfername] section. It is in fact the same as the predefined FTRAN transfer type and therefore not really needed. However, to use it you could specify Transfer=Microfit in the [shortname] section for the relevant connection.

```
;GCOS7 Microfit/Ftran
[Microfit]
Dialog=Trnline "FTRAN MICROSYS;"
Server=Emulate "^[401C^[6C"
SendText=Trnline "FTRAN MICROSYS;";Putfile FTRAN "%p;%h"
SendBinary=Trnline "FTRAN MICROSYS;";Putfile FTRAN "%p;%h"
ReceiveText=Trnline "FTRAN MICROSYS;";Getfile FTRAN "%p;%h"
ReceiveBinary=Trnline "FTRAN MICROSYS;";Getfile FTRAN "%p;%h"
```

Debugging UVTI applications

The `UVTIDB32.DLL` file can help in debugging UVTI applications. To use this utility, rename the `UVTI32.DLL` file in your Windows directory to `GLUVTI32.DLL`. Copy the `UVTIDB32.DLL` file to `UVTI32.DLL` in the Windows directory. This DLL will intercept calls to the UVTI and, if debugging is enabled with `'Debugging=On'` in the `[Configuration]` section of the `GLUVTI.INI` file, will produce a trace of all UVTI calls and events on the file specified with `'DebugFile=filename'` in the same section.