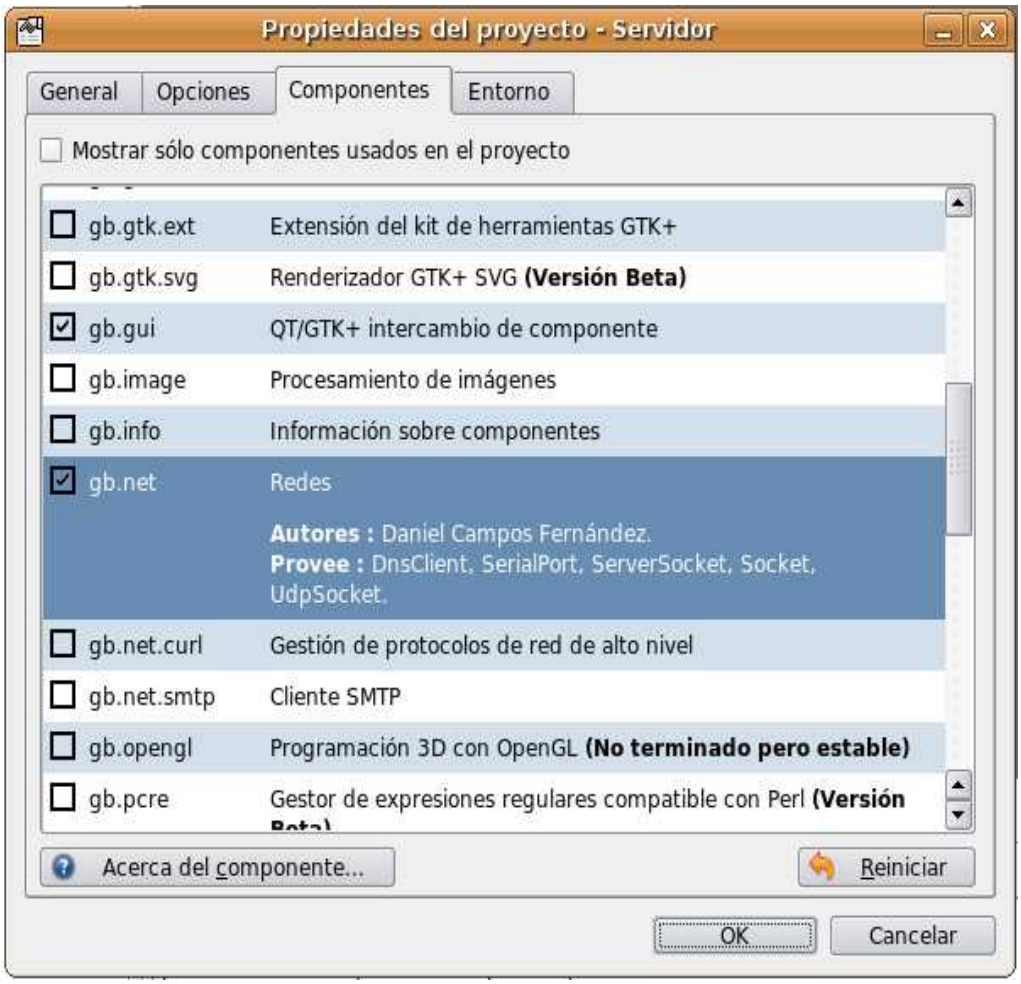
Al igual que un usuario se comunica con el programa por medio del teclado, dos programas se pueden comunicar entre sí por medio **Sockets**. Se podría decir que los sockets son "canales" por los cuales dos programas, posiblemente situados en computadoras distintas, pueden intercambiarse cualquier flujo de datos, generalmente de manera fiable y ordenada.

Los sockets permiten implementar una arquitectura cliente/servidor. En esta arquitectura la comunicación siempre es iniciada por uno de los programas que se denomina programa cliente. El segundo programa se encuentra a la escucha de peticiones entrantes con el fin de servir algún servicio, por este motivo se denomina programa servidor.

Un socket queda definido por una dirección IP, un protocolo y un número de puerto. Estos parámetros configuran las condiciones necesarias para que el programa servidor y el programa cliente puedan leer y escribir información coordinadamente.

Para trabajar con sockets en Gambas es necesario crear una referencia al componente de redes gb.net. Esta referencia la creamos desde Proyecto ---> Propiedades . Allí seleccionamos la solapa Componentes y en el listado tildamos gb.net, tal cual se observa a continuación.



Como resultado obtenemos una nueva categoría en la caja de herramientas, Network:

Para enviar un mensaje de una computadora a otra debemos desarrollar dos programas diferentes. Uno denominado servidor y otro denominado cliente. El servidor siempre estará esperando recibir una solicitud de conexión de otro programa, el cliente.

Aplicaciones Cliente/Servidor

Las aplicaciones que trabajan en una red (ya sea en una LAN o en Internet) se basan en la arquitectura cliente/servidor. Esta arquitectura consiste en una aplicación principal, que ofrece un servicio (servidor) y se mantiene a la espera de que una aplicación cliente se conecte solicitando una determinada información. En este tipo de arquitectura es común tener un único programa servidor y cientos de aplicaciones clientes que realizan peticiones para establecer una conexión.

Servidor: Es toda aplicación que se mantiene a la espera de que un cliente solicite información, la cual será entregada si el servidor así lo desea. Se dice que este ofrece o sirve un servicio.

Cliente: Es toda aplicación que se conecta a un servidor para solicitarle alguna información.

Un claro ejemplo de una aplicación cliente/servidor es el CiberControl, que a través de un programa servidor permite controlar una gran cantidad de programas clientes.

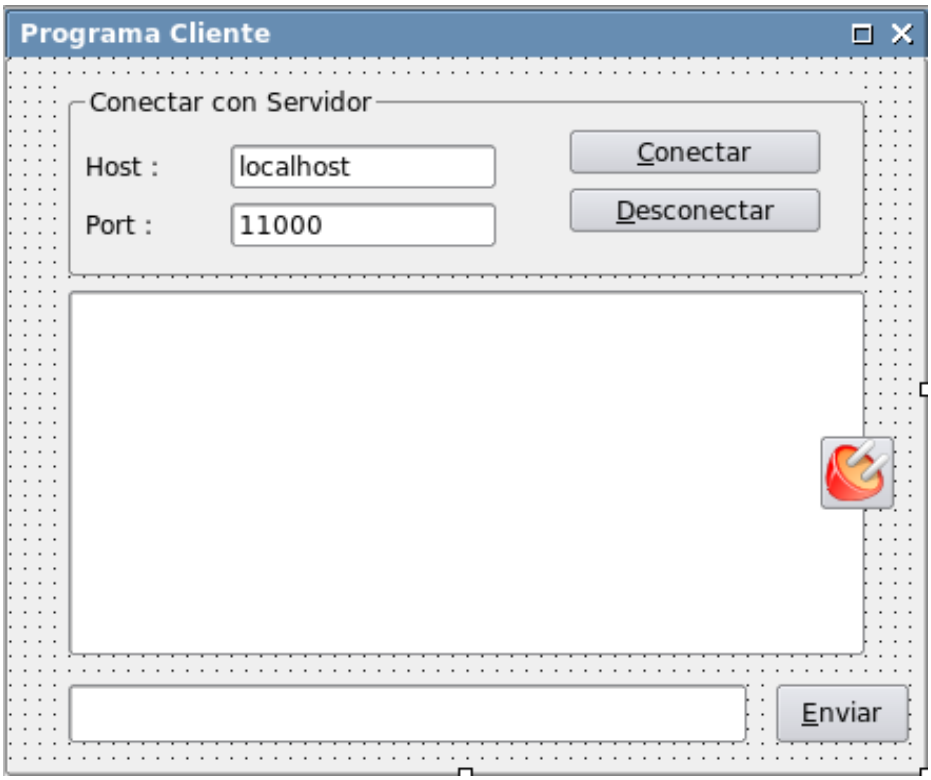
Controles ServerSocket y Socket 

Estos controles permiten realizar conexiones cliente/servidor a través del protocolo TCP.

El ServerSocket se comporta como un servidor se socket, es decir, se encuentra a la escucha de peticiones de clientes remotos o locales (dos aplicaciones en una misma computadora también se pueden comunicar) a través de un determinado puerto. Los puertos se enumeran del 1 al 65535. Los puertos del 1 al 1024 se consideran reservados por otros servicios y no es recomendable utilizarlos.

El control socket es el que estará en la aplicación cliente y es quien solicitará la conexión al servidor ubicándolo por su número de IP y contactándolo por el puerto en donde este “escuchando”.

Programa Cliente:



Al iniciar la aplicación cliente los botones Desconectar y Enviar deberán estar deshabilitados, ya que no existe conexión con servidor alguno.

```
PUBLIC SUB Form_Open()
  Button2.Enabled = FALSE
  Button3.Enabled = FALSE
END
```

La primera acción a realizar, y fundamental para toda aplicación de este tipo, es crear la conexión al servidor, ya que solo se puede transmitir información si la conexión cliente/servidor se encuentra activa.

Propiedades necesarias del control Socket:

- Host: Asignamos la dirección a la que deseamos conectar (puede ser el nombre de la PC o su número de IP). Este valor será ingresado en TextBox1.

- Port: Asignamos el puerto al que deseamos conectar con el Servidor. Es el número de puerto en que está "escuchando" el programa servidor. Este valor será ingresado en TextBox2.

Métodos necesarios del control Socket:

- Connect: Conecta al Servidor. Será codificado en el botón Conectar (Button1 en este caso).

A continuación la programación del botón Conectar, que debe ejecutar, como se ha dicho, el método Connect():

```
PUBLIC SUB Button1_Click()
  Socket1.Host = TextBox1.Text
  Socket1.Port = Val(TextBox2.Text)
  Socket1.Connect()
  Button1.Enabled = FALSE
  Button2.Enabled = TRUE
END
```

En las primeras dos líneas asignamos los datos de conexión al servidor remoto, como son la IP o DNS (Host) y número de puerto (Port). Luego llamamos al método Connect para realizar la conexión y cambiamos el estado de los botones.

Si la conexión se realiza con éxito el control Socket dispara el evento Ready(), aquí es donde podemos realizar acciones inmediatas en el momento preciso en que se logra establecer la conexión con el servidor. En este caso vamos a informar en el TextArea1 el éxito de la conexión y habilitamos el botón Enviar y Desconectar.

```
PUBLIC SUB Socket1_Ready()
  TextArea1.text = TextArea1.Text  & "Conexión exitosa con Servidor : " & Socket1.LocalHost & ":" &
Socket1.LocalPort & gb.Newline
  Button1.Enabled = FALSE
  Button2.Enabled = TRUE
  Button3.Enabled = TRUE
END
```

También hay que tener presente que en cualquier momento el servidor nos puede cerrar la conexión, o bien cerrarse por algún error. Para ello contamos con el evento Closed(), que es disparado por el control Socket al perder la conexión con el servidor y será informado de la siguiente manera:

```
PUBLIC SUB Socket1_Closed()
  TextArea1.text = TextArea1.Text  & "Conexión cerrada por el Servidor." & gb.NewLine
  Button1.Enabled = TRUE
  Button2.Enabled = FALSE
  Button3.Enabled = FALSE
END
```

En cambio, si queremos cerrar nosotros mismos la conexión con el servidor basta con cerrar el control Socket cuando se presione el botón Desconectar (Button2).

```
PUBLIC SUB Button2_Click()
  CLOSE Socket1
  TextArea1.Text = TextArea1.Text & "Conexión cerrada por el Cliente" & gb.newline
  Button1.Enabled = TRUE
  Button2.Enabled = FALSE
  Button3.Enabled = FALSE
END
```

Solo resta programar nuestro Cliente para enviar y recibir datos. Cuando se reciben datos el control Socket dispara el evento Read(), desde allí podemos leer los datos de la misma manera que se lo hace con un archivo:

```
PUBLIC SUB Socket1_Read()
  DIM Recibido AS String
  IF Socket1.Status = Net.Connected THEN
      READ #Socket1, Recibido, Lof(Socket1)
      TextArea1.Text = TextArea1.Text & "Servidor ---> " & Recibido & gb.NewLine
  END IF
END
```

Para enviar datos al servidor lo hacemos basándonos en la escritura de archivos, en este caso escribiendo en el Socket como si se tratase de un archivo. Esto se ejecutará cada vez que se pulse el botón Enviar (Button3).

```
PUBLIC SUB Button3_Click()
  IF Socket1.Status = Net.Connected THEN
    WRITE #Socket1, TextArea2.Text, Len(TextArea2.Text)
    TextArea1.Text = TextArea1.text & "Cliente ---> " & TextArea2.Text & gb.newline
    TextArea2.Text = ""
  END IF
END
```

Por ultimo, podemos utilizar el evento Error para detectar cualquier problema de conexión. En caso que ocurra uno, procedemos a cerrar la conexión e informar al usuario sobre dicho error:

```
PUBLIC SUB Socket1_Error()
  SELECT CASE Socket1.Status
    CASE Net.CannotCreateSocket
      Message.Error("El sistema no permite crear un socket")
    CASE Net.HostNotFound
       Message.Error("Host no encontrado")
    CASE Net.ConnectionRefused
       Message.Error("No es posible conectar. La solicitud fue rechazada")
    CASE Net.CannotRead
       Message.Error("Error leyendo datos")
    CASE Net.CannotWrite
       Message.Error("Error escribiendo datos")
  END SELECT
  Button1.Enabled = TRUE
  Button2.Enabled = FALSE
  Button3.Enabled = FALSE
END
```

Antes de dar por finalizada la aplicación, es recomendable cerrar el socket si es que esta activo al momento de cerrar el formulario.

```
PUBLIC SUB Form_Close()
  IF Socket1.Status = Net.Active THEN
     CLOSE Socket1
  END IF
END
```

De esta manera el programa Cliente está terminado, para probarlo es necesario crear el programa Servidor. Pero mientras tanto es posible probarlo con un servidor web, por ejemplo google. Para ello iniciamos el programa y conectamos con www.google.com.ar (DNS), el puerto designado para transferencia de páginas web es el 80. De esta manera podemos verificar si nuestra aplicación es capaz de conectarse o no con una aplicación remota.

Programa Servidor:

El programa servidor es una aplicación totalmente independiente de la aplicación cliente, por lo cual habrá que abrir un nuevo proyecto en Gambas y diseñar el formulario propuesto.

Al iniciar el servidor los botones Desconectar y Enviar deberán estar deshabilitados, ya que no se encuentra activo con conexiones aceptadas.

```
PUBLIC SUB Form_Open()
  Button2.Enabled = FALSE
  Button3.Enabled = FALSE
END
```

El siguiente paso será habilitar el control ServerSocket para que pueda atender conexiones entrantes, es decir, dejarlo "a la escucha". Para esto necesitamos un botón "Escuchar" y un número de puerto local (a elección, siempre y cuando no sea un puerto reservado) por donde deseamos recibir conexiones entrantes.

Propiedades necesarias:

-Type: Especificamos el protocolo de comunicación, en nuestro caso vía TCP.

-Port: Asignamos el puerto local por donde deseamos recibir conexiones.

Métodos necesarios:

- Listen: Escucha peticiones entrantes.

- Close: Cierra el ServerSocket, desatendiendo de esta manera las solicitudes de los clientes.

Eventos involucrados:

- ConnectionRequest(): Ocurre cuando un Cliente solicita una conexión al Servidor.

- Socket_Closed(): Ocurre cuando el Cliente cierra la conexión, no es disparado por el control ServerSocket.

- Error(): Ocurre en caso de errores.

Iniciamos entonces la codificación con el botón escuchar, que debe llamar al método Listen.

```
PUBLIC SUB Button1_Click()

  ServerSocket1.Type = Net.Internet
  ServerSocket1.Port = Val(TextBox1.Text)
  ServerSocket1.Listen()
  IF ServerSocket1.Status = Net.Active THEN
     Button1.Enabled = FALSE
     Button2.Enabled = TRUE
     TextArea1.Text = "Servidor activo a la escucha de conexiones entrantes" & gb.NewLine
  END IF
END
```

Como se observa la primera acción es indicar el protocolo de conexión, Net.Internet hace referencia al protocolo TCP sobre el cuál se apoya internet. Luego se indica el puerto por donde entrarán las conexiones remotas y finalmente el método Listen lo hace operativo.

Hasta aquí el Socket sólo esta "escuchando" conexiones. Cuando un cliente intenta conectarse el ServerSocket dispara el evento Connection(), en donde podemos aceptar o rechazar la conexión como se lo hace a continuación.

```
PUBLIC SUB ServerSocket1_Connection(RemoteHostIP AS String)
  TextArea1.Text = TextArea1.Text & "Conexión solicitada por :" & RemoteHostIP & gb.NewLine
  SocketAceptado = ServerSocket1.Accept()
  TextArea1.Text = TextArea1.Text & "Conectado con " & SocketAceptado.RemoteHost & ":" &
SocketAceptado.RemotePort & " por puerto local " & SocketAceptado.LocalPort & gb.newline
  Button3.Enabled = TRUE
END
```

Una cosa resalta del código anterior, al aceptar la conexión se crea una instancia de un nuevo objeto que representa la conexión con el cliente. Este objeto debe existir a lo largo de nuestro proyecto, por lo cual es necesario definir esta variable como publica del formulario. Lo hacemos en el inicio del código:

```
PUBLIC SocketAceptado AS Object
```

A esta altura ya estamos en condiciones de aceptar conexiones de clientes. Cuando un cliente se intenta conectar por nuestro puerto, el ServerSocket lo detectará y generará el evento Connection(). En ese momento podemos identificar al cliente por su IP, que se encuentra en el parametro RemoteHostIP.

A continuación vamos a codificar el botón desconectar para anular la conexión con el programa cliente:

```
PUBLIC SUB Button2_Click()
  ServerSocket1.Close()
  TextArea1.Text = TextArea1.Text & "La conexión fue cerrada por el servidor." & gb.newline
  Button1.Enabled = TRUE
  Button2.Enabled = FALSE
  Button3.Enabled = FALSE
END
```

Los objetos Socket (ServerSocket y Socket) tienen un gestor de eventos, justamente llamado Socket, es por esto que, para detectar el cierre de la conexión por parte del cliente, lo hacemos desde el evento Socket_Closed():

```
PUBLIC SUB Socket_Closed()
    TextArea1.Text = TextArea1.Text & "El cliente cerró la conexión." & gb.NewLine
END
```

Solo resta programar nuestro servidor para enviar y recibir los datos que provienen del objeto SocketAceptado. Lo hacemos como si el objeto SocketAceptado fuese un archivo, a través del evento Read generado por el gestor de eventos Socket:

```
PUBLIC SUB Socket_Read()
    DIM Recibido AS String
    READ #SocketAceptado, Recibido, Lof(SocketAceptado)
    TextArea1.Text = TextArea1.Text & "Cliente ---> " & Recibido & gb.NewLine
END
```

Para enviar datos programamos el botón Enviar (Button3) también considerando el objeto SocketAceptado como si fuese un archivo.

```
PUBLIC SUB Button3_Click()
    WRITE #SocketAceptado, TextArea2.Text, Len(TextArea2.Text)
    TextArea1.Text = TextArea1.text & "Servidor ----> " & TextArea2.Text & gb.newline
    TextArea2.Text = ""
END
```

Por ultimo podemos utilizar el evento error para detectar cualquier problema de conexión. En caso que ocurra uno podemos cerrar la conexión e informar al usuario de dicho error:

```
PUBLIC SUB ServerSocket1_Error()
  Message.Error("Error en el Servidor de Socket")
  CLOSE ServerSocket1
END
```

De esta manera el programa servidor está terminado y listo para probarlo junto al programa cliente.

Pablo Mileti

pablomileti@gmail.com