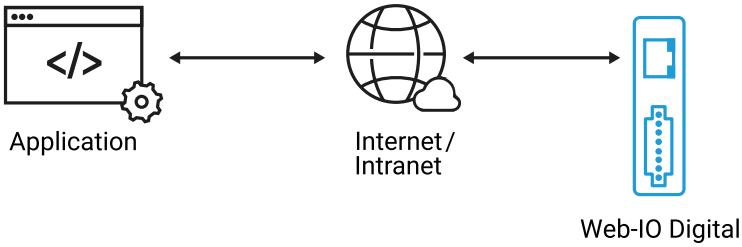
Tutorial al Web-IO digital:

Dirigir y controlar Web-IO digital con Visual C++

Resumen de productos
Sinopsis de aplicaciones

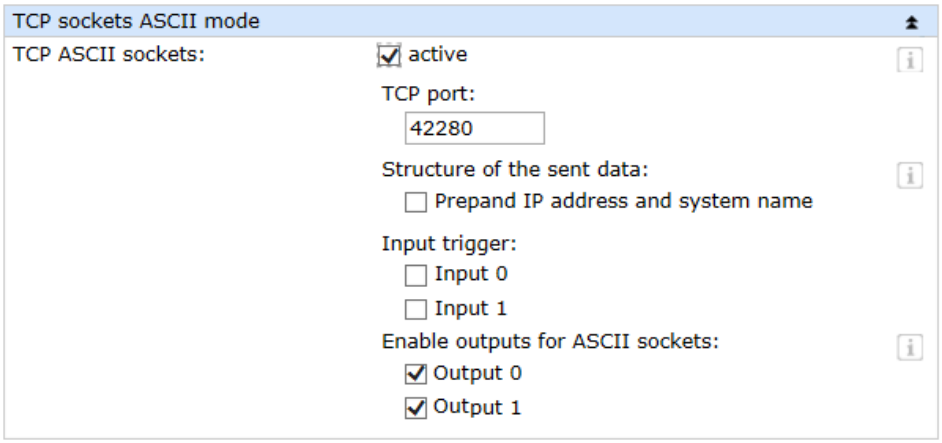
Para la creación de aplicaciones Windows Visual C++ es una de las plataformas de desarrollo más utilizadas.



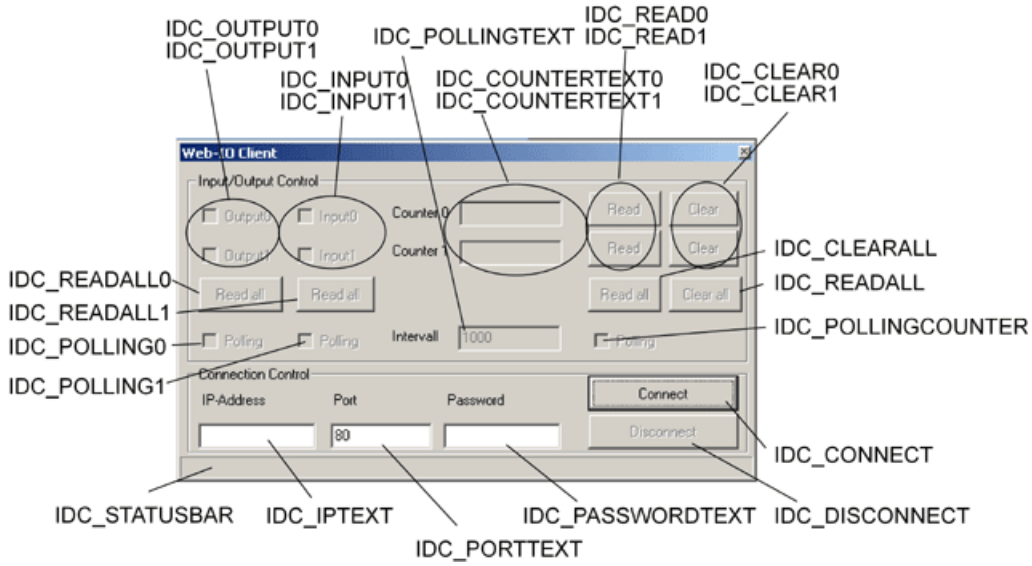
Con el ejemplo siguiente de programa C++ puede representar su Web-IO Digital con sus Inputs y Outputs en una aplicación Windows. Además de ello puede conectar los Outputs del Web-IO.

Preparativos

- Conectar la tensión para Web-IO y cablear las IO
- Conectar el Web-IO a la red
- Asignar las direcciones IP
- En Web-IO, dentro de la sección *Vías de comunicación >> Zócalo-API* activar los *Zócalos TCP-ASCII* y habilitar las *salidas a conmutar*



Recopilación de los diferentes elementos de manejo y objetos de visualización en formulario Visual C++



Definición en el archivo Header de MySocket y CWeb_IO_ClientDlg

A fin de salvar las entradas en lso campos IDC_IPTEXT, IDC_PORTTEXT, IDC_PASSWORDTEXT e IDC_POLLINGTEXT en variables de miembros privadas de la clase CWeb_IO_ClientDlg, se definen algunas variables en el archivo Header.

Para no tener que crear una nueva variable para cada noticia enviada y recibida, se crean también para ello variables privadas en el Header. La entrada en la barra de estado también recibe aquí una variable.

```
private:
CString m_ip;
CString m_message1;
CString m_message2;
CString m_rcv;
CString m_password;
int m_port;
int m_range;
public:
CStatusBarCtrl*
m_statusBar;
```

También se declaran previamente en el Header algunos métodos necesarios.

```
public:
void changeAccess(int i);
void TryConnect();
void OnConnect();
void OnDisconnect();
void checkpass();
void OnReceive();
void OnSend();
void OnClose();
void OnTimer(UINT nIDEvent);
void readAndClearCounter(CString data)
```

Para inicializar ahora también una conexión, necesitamos todavía una clase, que se ocupa de enviar y recibir datos. Para ello se creó un archivo llamado MySocket, que es sucesor de la clase CAsyncSocket. Así se pasan todos los métodos públicos de esta clase a nuestra clase y se declaran de nuevo en el Header. Para poder acceder más tarde a los métodos de la clase CWeb_IO_ClientDlg, se declara un objeto en el Header.

```
public:
CWeb_IO_ClientDlg*
m_client;

public:
virtual void OnReceive(int nErrorCode);
virtual void OnSend(int nErrorCode);
virtual void OnClose(int nErrorCode);
```

Definición en MySocket.cpp

Puesto que la clase MySocket ha heredado de la clase CAsyncSocket, se superponen algunos de los métodos necesarios para una conexión en curso. De este modo la aplicación puede enviar y recibir más tarde perfectamente tanto datos asincrónicamente como también reaccionar a la terminación de la conexión por parte del Web-IO.

```
void MySocket::OnReceive(int nErrorCode)
{
  m_client->OnReceive();
  CAsyncSocket::OnReceive(nErrorCode);
}

void MySocket::OnSend(int nErrorCode)
{
  m_client->OnSend();
  CAsyncSocket::OnSend(nErrorCode);
}

void MySocket::OnClose(int nErrorCode)
{
  m_client->OnClose();
  CAsyncSocket::OnClose(nErrorCode);
}

void MySocket::OnConnect(int nErrorCode);
{
  if(nErrorCode == 0)
  {
    m_client->OnConnect();
    CAsyncSocket::OnConnect(nErrorCode);
  }
  else
  {
    m_client->OnDisconnect();
    m_client->m_statusBar->SetText("Error while connecting!", 0, 0);
  }
}
```

Arranque de programa

Instalar los elementos de manejo
El grupo con los elementos de manejo para el Web-IO se bloquea primero para el manejo. Tan pronto como se establezca una conexión, se da línea libre a todos los elementos, que poseen una versión conveniente.

A fin de enviar mensajes a una barra de estado, se genera una barra tal en este método. Además se entrega la clase del MySocket de modo que ambos están enlazados entre si.

A fin de enlazar variables o métodos con los componentes de la aplicación, se puede aprovechar la ayuda del asistente de clases de Visual C++, o a partir del Visual Studio 2003 la correspondiente funcionalidad en las características de clase.

```
BOOL CWeb_IO_ClientDlg::OnInitDialog()
{
  CDialog::OnInitDialog();
  // weitere Implementierungen...

  m_statusBar = new CStatusBarCtrl;
  m_statusBar->Create(WS_CHILD|WS_VISIBLE, CRect(0, 0, 0, 0), this, 0);
  clientSocket.m_client = this;
  changeAccess(FALSE);
  SendMessage(DM_SETDEFID,
  IDC_CONNECT);

  return TRUE;
}

void CWeb_IO_ClientDlg::changeAccess(bool b)
{
  if(b == TRUE)
  {
    GetDlgItem(IDC_CONNECT)->EnableWindow(FALSE);
    GetDlgItem(IDC_DISCONNECT)->EnableWindow(TRUE);
    GetDlgItem(IDC_OUTPUT0)->EnableWindow(TRUE);
    GetDlgItem(IDC_OUTPUT1)->EnableWindow(TRUE);
    GetDlgItem(IDC_READALL0)->EnableWindow(TRUE);
    GetDlgItem(IDC_READALL1)->EnableWindow(TRUE);
    GetDlgItem(IDC_POLLING0)->EnableWindow(TRUE);
    GetDlgItem(IDC_POLLING1)->EnableWindow(TRUE);
    GetDlgItem(IDC_POLLINGTEXT)->EnableWindow(TRUE);
    GetDlgItem(IDC_POLLINGCOUNTER)->EnableWindow(TRUE);
    GetDlgItem(IDC_READ0)->EnableWindow(TRUE);
    GetDlgItem(IDC_READ1)->EnableWindow(TRUE);
    GetDlgItem(IDC_CLEAR0)->EnableWindow(TRUE);
    GetDlgItem(IDC_CLEAR1)->EnableWindow(TRUE);
    GetDlgItem(IDC_READALL)->EnableWindow(TRUE);
    GetDlgItem(IDC_CLEARALL)->EnableWindow(TRUE);
  }
  else
  {
    GetDlgItem(IDC_CONNECT)->EnableWindow(TRUE);
    GetDlgItem(IDC_DISCONNECT)->EnableWindow(FALSE);
    GetDlgItem(IDC_OUTPUT0)->EnableWindow(FALSE);
    GetDlgItem(IDC_OUTPUT1)->EnableWindow(FALSE);
    GetDlgItem(IDC_INPUT0)->EnableWindow(FALSE);
    GetDlgItem(IDC_INPUT1)->EnableWindow(FALSE);
    GetDlgItem(IDC_READALL0)->EnableWindow(FALSE);
    GetDlgItem(IDC_READALL1)->EnableWindow(FALSE);
    GetDlgItem(IDC_POLLING0)->EnableWindow(FALSE);
    GetDlgItem(IDC_POLLING1)->EnableWindow(FALSE);
    GetDlgItem(IDC_POLLINGTEXT)->EnableWindow(FALSE);
    GetDlgItem(IDC_POLLINGCOUNTER)->EnableWindow(FALSE);
    GetDlgItem(IDC_COUNTERTEXT0)->EnableWindow(FALSE);
    GetDlgItem(IDC_COUNTERTEXT1)->EnableWindow(FALSE);
    GetDlgItem(IDC_READ0)->EnableWindow(FALSE);
    GetDlgItem(IDC_READ1)->EnableWindow(FALSE);
    GetDlgItem(IDC_CLEAR0)->EnableWindow(FALSE);
    GetDlgItem(IDC_CLEAR1)->EnableWindow(FALSE);
    GetDlgItem(IDC_READALL)->EnableWindow(FALSE);
    GetDlgItem(IDC_CLEARALL)->EnableWindow(FALSE);
  }
}
```

El control de conexión

Introducir la conexión

Entrando la dirección IP del Web-IO en el campo de texto IDC_IPTEXT y en el puerto 80 en el campo de texto IDC_PORTTEXT se puede establecer una conexión activando el botón IDC_CONNECT. Si no se entra ninguna dirección IP o ningún puerto, se da un mensaje en la barra de estado.

```
void CWeb_IO_ClientDlg::OnClickConnect()
{
  GetDlgItemText(IDC_IPTEXT, m_ip);
  GetDlgItemText(IDC_PORTTEXT, m_message2);
  sscanf(m_message2, "%d", &m_port);

  if(m_ip == "")
  {
    m_statusBar->SetText("No IP entered!", 0, 0);
  }
  else if(m_port == 0)
  {
    m_statusBar->SetText("No port entered!", 0, 0);
  }
  else
  {
    TryConnect();
  }
}
```

La conexión

Para poder realizar una conexión TCP, utilizamos en nuestro ejemplo un objeto global de la clase MySocket, que pone a disposición métodos como Create(), Connect() y Close().

En el caso de que se hubiera entrado un puerto negativo o hubiera fallado la conexión, se enviará un mensaje respectivamente.

Puesto que la conexión no pudo establecerse, el zócalo se cierra de nuevo de modo que puede volverse a iniciar sin problema alguno una nueva conexión.

Si todo ha funcionado bien, se da línea libre a todos los componentes, que se pueden utilizar después de establecer la conexión, y se bloquea el botón IDC_CONNECT para evitar una nueva conexión existiendo ya otra.

```
void CWeb_IO_ClientDlg::TryConnect()
{
  if(m_port < 0)
  {
    m_statusBar->SetText("No
    negative port allowed!", 0, 0);
    return;
  }
  clientSocket.Create();
  clientSocket.Connect(m_ip, m_port);
  GetDlgItem(IDC_CONNECT)->EnableWindow(FALSE);
}
```

Se establece la conexión

Si se ha establecido la conexión, la aplicación puede comunicarse con el Web-IO, para la que se pueden utilizar los diferentes componentes.

```
void CWeb_IO_ClientDlg::OnConnect()
{
  m_statusBar->SetText("Connection made!", 0, 0);
  changeAccess(TRUE);
}
```

Separar la conexión

La conexión permanece tanto tiempo hasta que el usuario la finalice chasqueando el botón Disconnect o el Web-IO finalice la conexión. Si se finaliza la conexión, se envía un mensaje.

```
void CWeb_IO_ClientDlg::OnClickDisconnect()
{
  OnDisconnect();
}
```

Si se separa la conexión, tienen que transformarse todos los elementos a la posición de partida, pues entonces debe ser imposible poder chasquear el botón IDC_DISCONNECT.

```
void CWeb_IO_ClientDlg::OnDisconnect()
{
  clientSocket.Close();
  m_statusBar->SetText("Disconnected!", 0, 0);
  changeAccess(FALSE);
}
```

Ahora se ha finalizado la conexión y la aplicación ha retrocedido a su estado de partida.

Manejo y comunicación por parte del cliente

Tan pronto como se ha establecido la conexión con el Web-IO, el usuario puede enviar comandos al Web-IO manejando los correspondientes elementos de programa.

Poner Outputs
Los Outputs del Web-IO pueden conectarse con ayuda de las dos cajas de chequeo IDC_OUTPUT0 e IDC_OUTPUT1. La caja de chequeo nota si se va a chasquear y dispara entonces una acción. Para poder enviar una solicitud al Web-IO, se controla primero si ha sido introducida una contraseña en el campo de texto IDC_PASSWORDTEXT sin la cual el Web-IO no aceptaría ninguna pregunta. Si no se colocó ninguna contraseña, se envía la pregunta sin contraseña.

```
void CWeb_IO_ClientDlg::checkpass()
{
  GetDlgItemText(IDC_PASSWORDTEXT, m_password);
}
```

En el paso siguiente se controla si la caja de chequeo ya está asentada o no, asentándose entonces el Output o siendo repuesto de nuevo a cero.

```
void CWeb_IO_ClientDlg::OnOutput0()
{
  checkpass();
  m_message1 = "GET /outputaccess0?PW=" + password + "&State=ON&";
  m_message2 = "GET /outputaccess0?PW=" + password + "&State=OFF&";
  if(IsDlgButtonChecked(IDC_OUTPUT0))
    clientSocket.Send(m_message1, m_message1.GetLength());
  else
    clientSocket.Send(m_message2, m_message2.GetLength());
}

void CWeb_IO_ClientDlg::OnOutput1()
{
  checkpass();
  m_message1 = "GET /outputaccess1?PW=" + password + "&State=ON&";
  m_message2 = "GET /outputaccess1?PW=" + password + "&State=OFF&";
  if(IsDlgButtonChecked(IDC_OUTPUT1))
    clientSocket.Send(m_message1, m_message1.GetLength());
  else
    clientSocket.Send(m_message2, m_message2.GetLength());
}
```

Solicitar estado de Output/Input
Ahora se puede controlar también el estado de los Outputs e Inputs para actualizar respectivamente las cajas de control en la aplicación. El botón IDC_READALL0 lee el estado de los Outputs y el botón IDC_READALL1 lee el estado de los Inputs. La orden para preguntar el estado es algo diferente a la de asentar. O sea que el estado de los Outputs se puede solicitar con "GET /output?PW=&".

```
void CWeb_IO_ClientDlg::OnClickReadall0()
{
  checkpass();
  m_message1 = "GET /output?PW=" + password + "&";
  clientSocket.Send(m_message1, m_message1.GetLength());
}

void CWeb_IO_ClientDlg::OnClickReadall1()
{
  checkpass();
  m_message1 = "GET /input?PW=" + password + "&";
  clientSocket.Send(m_message1,
  m_message1.GetLength());
}
```

Solicitar Contadores

Puesto que los sucesos entrantes de Input sólo se anotan en el Web-IO mismo y así el contador interior cuenta hacia arriba, también se debería poder preguntar a éste. El método siguiente envía una solicitud a un contador determinado y exige una respuesta con el estado actual del contador. De la respuesta del Web-IO se lee el número del contador respectivo y de su estado de cuenta y se visualiza en la aplicación.

```
void CWeb_IO_ClientDlg::OnClickRead0()
{
  checkpass();
  m_message1 = "GET /counter0?PW=" + password + "&";
  clientSocket.Send(m_message1, m_message1.GetLength());
}

void CWeb_IO_ClientDlg::OnClickRead1()
{
  checkpass();
  m_message1 = "GET /counter1?PW=" + password + "&";
  clientSocket.Send(m_message1, m_message1.GetLength());
}
```

Naturalmente también se pueden solicitar todos los estados de contador con un solo comando.

```
void CWeb_IO_ClientDlg::OnClickReadall()
{
  checkpass();
  m_message1 = "GET /counter?PW=" + password + "&";
  clientSocket.Send(m_message1, m_message1.GetLength());
}
```

Reposición a cero de contadores

Puesto que se puede leer el estado de cuenta, debe ser también posible poner a 0 el contador. Para ello se envía una noticia al contador respectivo, que repone a cero éste.

```
void CWeb_IO_ClientDlg::OnClickClear0()
{
  checkpass();
  m_message1 = "GET /counterclear0?PW=" + password + "&";
  clientSocket.Send(m_message1, m_message1.GetLength());
}

void CWeb_IO_ClientDlg::OnClickClear1()
{
  checkpass();
  m_message1 = "GET /counterclear1?PW=" + password + "&";
  clientSocket.Send(m_message1, m_message1.GetLength());
}
```

Naturalmente también se pueden reponer a cero todos los contadores con un sólo comando.

```
void CWeb_IO_ClientDlg::OnClickClearall()
{
  checkpass();
  m_message1 = "GET /counterclear?PW=" + password + "&";
  clientSocket.Send(m_message1, m_message1.GetLength());
}
```

Puesto que todos los estados de contador se pueden leer o reponer a cero con un sólo comando, tiene que implementarse todavía un método, que procese el string de respuesta del Web-IO y que asigne su estado específico a cada contador en la aplicación.

```
void CWeb_IO_ClientDlg::readAndClearCounter(CString data)
{
  int j = 0;
  CString counter[12];

  for(int i = 0; i < data.GetLength(); i++)
  {
    if(data[i] == ';')
      j++;
    else
      counter[j]+= data[i];
  }

  SetDlgItemText(IDC_COUNTERTEXT0, counter[0]);
  SetDlgItemText(IDC_COUNTERTEXT1, counter[1]);
}
```

Recepción de datos del Web-IO

Evaluar e indicar los datos recibidos

- Todos los comandos y solicitudes al Web-IO se confirman con un String de respuesta. Aquí las respuestas presentan una estructura específica según el tipo.
- Se diferencia la respuesta entre un sólo Output o Input y de todos los Outputs e Inputs.
- Para los Outputs: output;<valor binario del estado de salida en formato hexadecimal>
- Para un Output especial: outputx;<ON u OFF>
- Para los Inputs: input;<valor binario del estado de salida en formato hexadecimal>
- Para un Input especial: inputx;<ON u OFF>
- Después hay también el string de respuesta para un contador que tiene el siguiente aspecto:
  counterx;<estado decimal de contador >
  o counter;<estado decimal de contador 0 >; <estado decimal de contador 1 >; ..... si todos los contadores se deben leer de una sola vez.
  Todos los Strings de respuesta se finalizan con un Byte 0.
- En nuestra aplicación se llama el método OnReceive() para recibir una noticia tal. En este método se procesa la respuesta del Web-IO.

```
void CWeb_IO_ClientDlg::OnReceive()
{
  clientSocket.Receive(m_rcv.GetBuffer(50),50);
  m_rcv.ReleaseBuffer();

  if(m_rcv.IsEmpty())
    OnDisconnect();
  else if(m_rcv.GetLength() > 1)
  {
    if(m_rcv[0] == 'o')
    {
      int i;
      sscanf(m_rcv.Right(m_rcv.GetLength() - 7), "%x", &i);
      if((i & 1) == 1)
        CheckDlgButton(IDC_OUTPUT0, BST_CHECKED);
      else
        CheckDlgButton(IDC_OUTPUT0, BST_UNCHECKED);
      if((i & 2) == 2)
        CheckDlgButton(IDC_OUTPUT1, BST_CHECKED);
      else
        CheckDlgButton(IDC_OUTPUT0, BST_UNCHECKED);
    }
    if(m_rcv[0] == 'i')
    {
      int i;
      sscanf(m_rcv.Right(m_rcv.GetLength() - 6), "%x", &i);
      if((i & 1) == 1)
        CheckDlgButton(IDC_INPUT0, BST_CHECKED);
      else
        CheckDlgButton(IDC_INPUT0, BST_UNCHECKED);
      if((i & 2) == 2)
        CheckDlgButton(IDC_INPUT1, BST_CHECKED);
      else
        CheckDlgButton(IDC_INPUT1, BST_UNCHECKED);
    }
    if(m_rcv[0] == 'c')
    {
      if(m_rcv[7] == '0')
        SetDlgItemText(IDC_COUNTERTEXT0, m_rcv.Right(m_rcv.GetLength() - 9));
      if(m_rcv[7] == '1')
        SetDlgItemText(IDC_COUNTERTEXT1, m_rcv.Right(m_rcv.GetLength() - 9));
      if(m_rcv[7] == ';')
        readAndClearCounter(m_rcv.Right(m_rcv.GetLength() - 8));
    }
  }
}
```

Polling

Solicitud cíclica de determinados valores
Ahora se desearía que el estado de un sólo componente se actualice por mí mismo y así la aplicación siempre presente el estado actual. Para ello se utiliza un temporizador en este programa, que envía solicitudes al Web-IO cíclicamente en un intervalo de tiempo fijado por el usuario.

Para ello se pude entrar en primer lugar un valor íntegro en el campo IDC_POLLINGTEXT, que define el tiempo en milisegundos para la solicitud cíclica. Si no se pone ningún valor, el tiempo para el intervalo está asentado de manera estándar a 1 segundo (1000 ms).

Naturalmente también se captura en el caso de que el usuario realice una entrada absurda, como p. ej. un valor negativo de tiempo. Entonces aparece inmediatamente un mensaje y no se asume el valor.

```
void CWeb_IO_ClientDlg::OnChangePollingtext()
{
  GetDlgItemText(IDC_POLLINGTEXT, m_message2);
  int check;
  sscanf(m_message2, "%d", &check);
  if(check <= 0)
  {
    m_statusBar->SetText("No negative
    value or character allowed!", 0, 0);
    return;
  }
  m_range = check;
  m_statusBar->SetText("Range
  changed!", 0, 0);
}
```

Para realizar ahora también la solicitud cíclica de los estados del Web-IO, lo que se llama también Polling, existe la elección entre el Polling de los Outputs, de los Inputs o de los contadores.

Se inicializa un temporizador propio por variante de Polling. La solicitud del temporizador es entonces "SetTimer(Número del temporizador, Intervalo, CERO)".

Si se activa la caja de chequeo IDC_POLLING0 se aplica el Polling a los Outputs. Aquí se inicializa un temporizador, si se asienta. Si la caja de chequeo se repone de nuevo a cero, entonces el temporizador se destruye también de nuevo con el número correspondiente.

```
void CWeb_IO_ClientDlg::OnPolling0()
{
  if(IsDlgButtonChecked(IDC_POLLING0))
    SetTimer(1, m_range, NULL);
  else
    KillTimer(1);
}
```

Si se activa la caja de chequeo IDC_POLLING1 se aplica el Polling a los Inputs. Entonces también se inicializa un nuevo temporizador, que se puede destruir también entonces por este método.

```
void CWeb_IO_ClientDlg::OnPolling1()
{
  if(IsDlgButtonChecked(IDC_POLLING1))
    SetTimer(2, m_range, NULL);
  else
    KillTimer(2);
}
```

Si se deben consultar también los contadores con Polling, entonces se puede utilizar para ello la caja de chequeo IDC_POLLINGCOUNTER.

```
void CWeb_IO_ClientDlg::OnPollingcounter()
{
  if(IsDlgButtonChecked(IDC_POLLINGCOUNTER))
    SetTimer(3,m_range, NULL);
  else
    KillTimer(3);
}
```

Ahora se inicializaron tres temporizadores diferentes, que disparan una acción según intervalo en determinados periodos de tiempo, que no se captura sin embargo hasta ahora. Para capturar los eventos tiene que implementarse un método todavía.

En este método se captura el evento respectivo del temporizador, que se está anunciando, y se le asigna una acción determinada.

Recordemos: En nuestro caso se consultan cíclicamente los estados de los Outputs, Inputs y de los contadores según evento.

```
void CWeb_IO_ClientDlg::OnTimer(UINT nIDEvent)
{
  if(nIDEvent == 1) OnClickReadall0();
  if(nIDEvent == 2) OnClickReadall1();
  if(nIDEvent == 3) OnClickReadall();
  CDialog::OnTimer(nIDEvent);
}
```

Todo el programa se pone a disposición en esta página, conteniendo adicionalmente algunos archivos y métodos no mencionados aquí. La aplicación MFC en Visual C++ genera un Overhead, que forma el marco gráfico, pero no está en contacto directo con la funcionalidad. Los métodos y variables que le han sido presentados en esta página, se implementan en el marco generado por Visual C++ y trabajan como unidad con la superficie gráfica.

El programa ejemplo asiste todas las funciones corrientes del Web-IO en el modo String de comando, optimado para el Web-IO 2x Digital Input, 2x Digital Output PoE . Para los otros modelos Web-IO tienen que realizarse en caso necesario adaptaciones en el programa. Otros ejemplos de programa para la programación del zócalo los encontrarán en las páginas de herramientas al Web-IO. Una descripción detallada de la interfaz del zócalo de los modelos Web-IO digitales la encontrarán en el manual de referencia.

Descargar el programa ejemplo

Productos

Web-IO 4.0 Digital
2xIn, 2xOut

Si es necesario, alimentación también por PoE

Web-IO 4.0 Digital
12xIn, 12xOut

12 entradas,
12 salidas

Otros Web-IO

Todos los Web-IO Digital 24V de W&T

W&T
www.WuT.de

Le atendemos personalmente:

Wiesemann & Theis GmbH
Porschestr. 12
42279 Wuppertal
Tel: +49 202/2680-110 (lu-vi de 8-17 horas)
Fax: +49-202/2680-265
info@wut.de