AxoNet Software GmbH

Introduction

Using the AxClient components for asanetwork

Contents

Overview

This introduction shows the basic use of the AxClient components for asanetwork. Using AxClient you can easily implement asanetwork clients for e.g. test and measurement equipment. It is assumed that you have a basic knowledge of asanetwork, the asanetwork SDK and the AxAwn COM layer.

The AxClient controls are ActiveX controls. Therefore you can make use of them in nearly all modern development environments like Microsoft Visual Basic or Visual C++, Borland Delphi and C++ Builder or even with Office applications like Excel or Access.

Installation

Please close any IDE before you start the installation. Then start the installation program DAxClientX.X.X.exe (demo version) or AxClientX.X.X.exe (full version). X.X.X is the current version number, e.g. 1.2.2.

After this step is done you need to add a reference or import the controls. This depends on your environment. In Visual Basic select Project/Components… Here you should select “AxoNet AxClientMgmt” and “AxoNet AxClientDisplay”. If you use Borland Delphi or C++Builder you need to import the AxtiveX controls. Select Components/import ActiveX. Create a new unit called AxClient.dpk and import “AxoNet AxClientMgmt” and “AxoNet AxClientDisplay”.

Before you start

First you need to define the required services for asanetwork.

You always have a customer order service based on your license name (DID). In this demo we use the name TEST_. For your application please replace this TEST_ DId with the one you'll find in your license agreement.

Then you have one or more test and measurement services as defines in the "Design guide" of asanetwork. The AxClient controls support test and measurement services with XML data (these start with AWNTX) and manufacturer specific services (these start with your DID).

For our demo we assume that we have these services in our equipment:

- TEST_00000 customer order services
- AWNTXBR000 Brake test (only the general service)
- AWNTXEM000 Emission test, general service, sub divided into
  - AWNTXEM010 Emission test, gas engine, no catalyst
  - AWNTXEM020 Emission test, gas engine, open loop catalyst
  - AWNTXEM030 Emission test, gas engine, closed loop catalyst
  - AWNTXEM050 Emission test, diesel engine

Then you have to define the service location (DLOC) and the specific properties of each service. This data is mapped into an INI file structure like this:

[AWN]	
DLoc=	The dloc of your application goes here
Orders=	order service, e.g. TEST_00000
Results=	all general result services separated by ; e.g. AWNTXBR00000;AWNTXEM00000
{Debug= optional}	0 = off, 1 = enabled, displays the debug window from the AxAwn COM library
[RESULTSERVICE]	Name of the general result services, repeated as necessary, e.g. AWNTXEM00000
SubService=	service names of all subservices separated by ; e.g. AWNTXEM010;AWNTXEM020
DiQual=	input quality of service, if missing defaults to 1
DoQual=	ouput quality of services, if missing defaults to 1
DPrio=	priority of service, if missing defaults to 9
[SUBSERVICE]	Name of sub service, repeated as necessary
DiQual=	Same as above
DoQual=	
DPrio=	

A working example looks like this awn.ini (from our demo project)

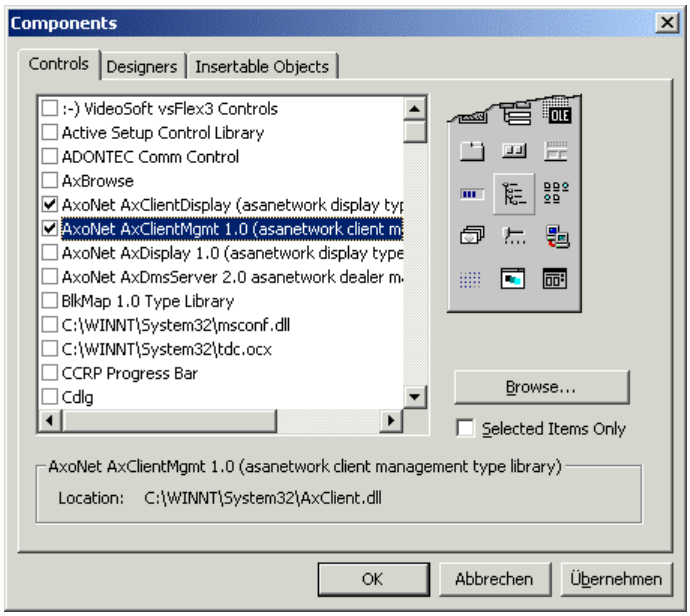
```
[AWN]
DLoc=AwnTest
Orders=TEST_00000
Results=AWNTXBR000;AWNTXEM000

[TEST_00000]
Info=customer order services for AwnTest

[AWNTXBR000]
Info=Brake test

[AWNTXEM000]
SubService=AWNTXEM010;AWNTXEM020;AWNTXEM030;AWNTXEM050
Info=Emission test, general service

[AWNTXEM010]
Info=Emission test, gas engine, no catalyst

[AWNTXEM020]
Info=Emission test, gas engine, open loop catalyst

[AWNTXEM030]
Info=Emission test, gas engine, closed loop catalyst

[AWNTXEM050]
Info=Emission test, diesel engine
```


Building a customer order client

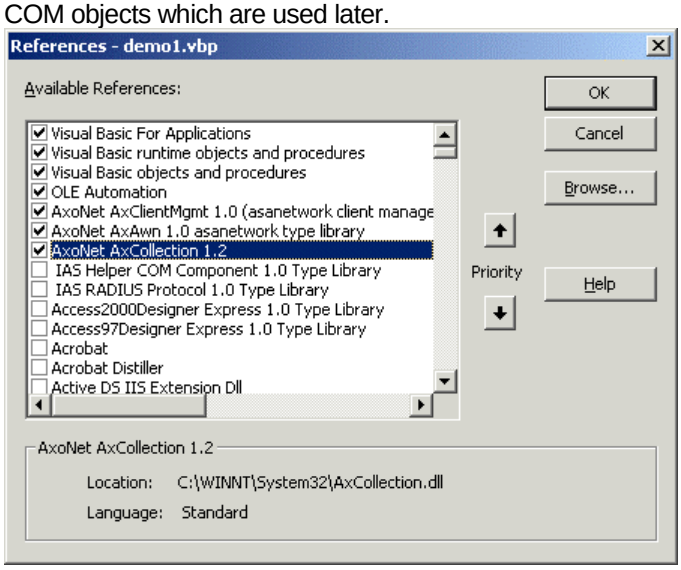
Step 1, building the first demo application

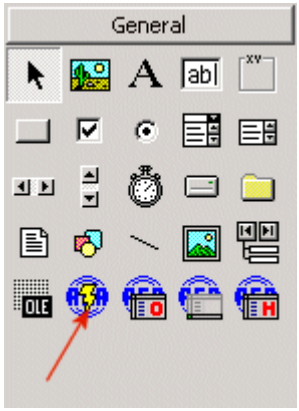
In this introduction we'll use Microsoft Visual Basic 6.0 but you can use other tools in a similar way. Start a new project, save the first form as OrderForm.frm and the project as demo1.prj.

Now open project/components and select “AxoNet AxClientMgmt” and “AxoNet AxClientDisplay”:

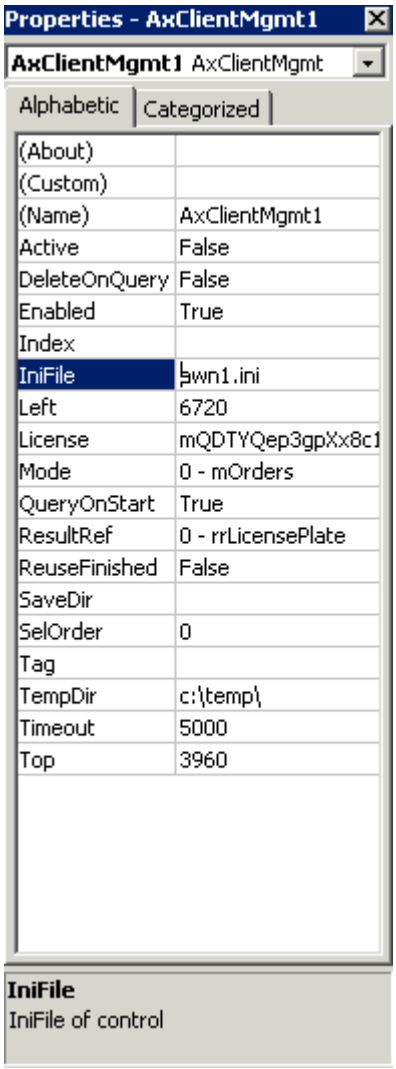


Then open project/references and select “AxoNet AxAwn” and “AxoNet AxCollection”. These libraries contain COM objects which are used later.





Then add an **AxClientMgmt** component from the palette and drop it onto the form. Place it in the lower right corner:

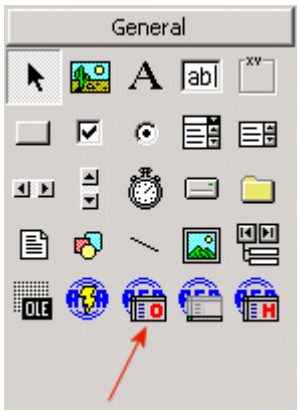


Look at the property window. We'll add some data. First put in the name of our previously created INI file. Here we use awn.ini. If you do not specify any path (like awn.ini) the ini file is search in the windows directory! So we use .\awn1.ini to specify the local directory.

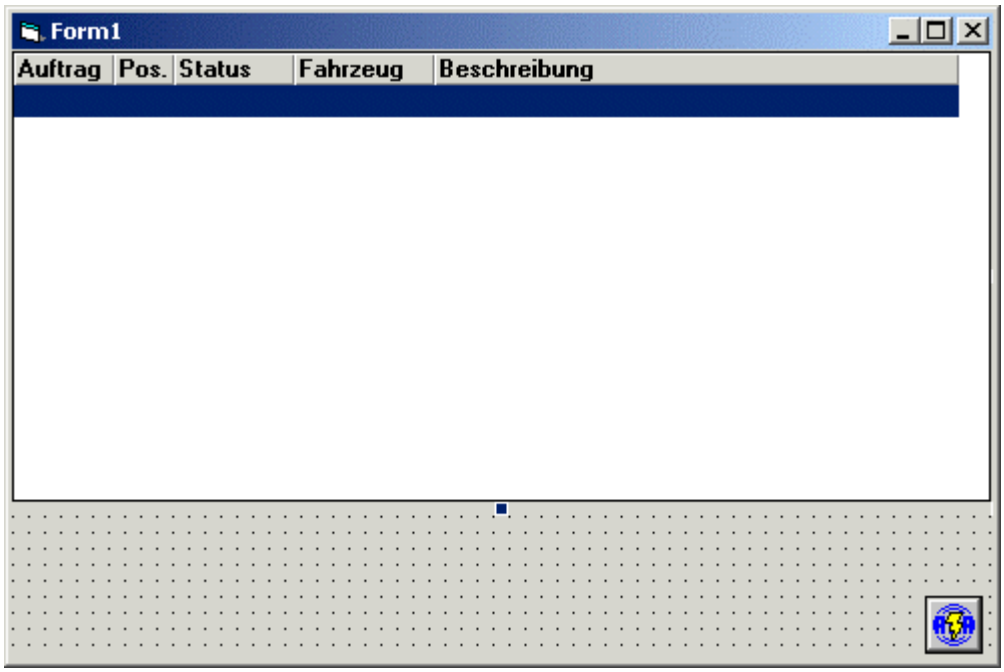
Then don't forget the license for your service name. Here we use the demo license for the TEST_ service.

The Mode property defaults to mOrders because we are building an order client.

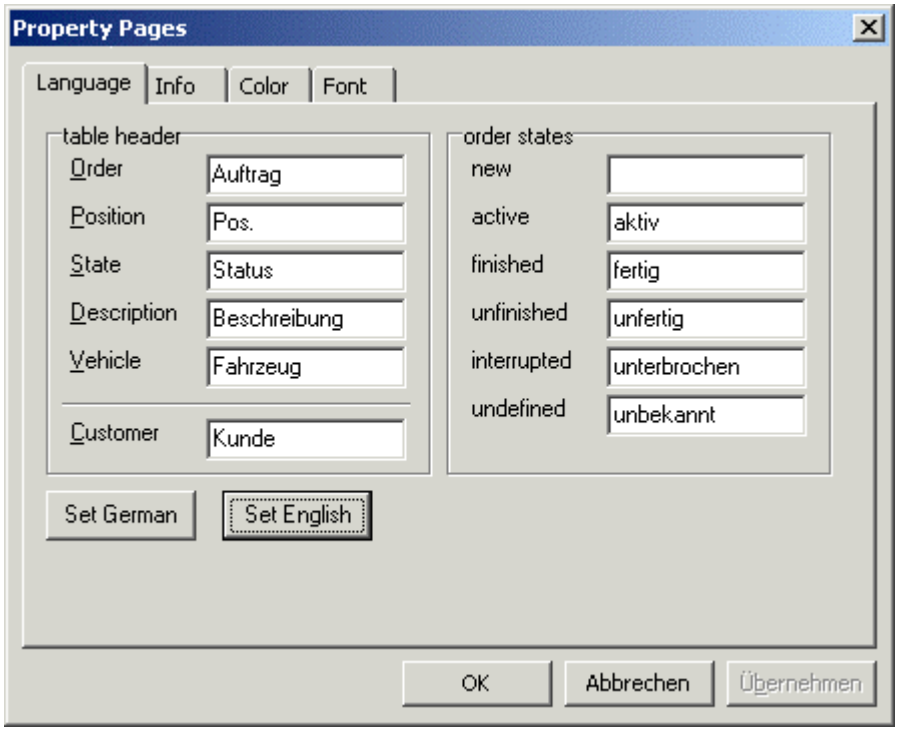
You may want to change the TempDir setting to a local sub directory of your project.

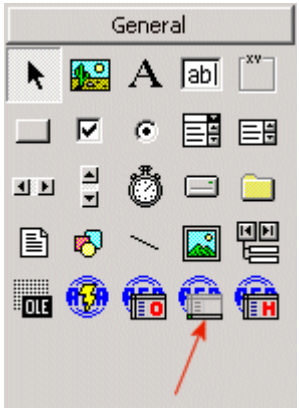


Now let's add the GUI part for an asanetwork client. Select the AxClientOrders control from the palette and place it on the form .This control is used to display orders. The default language for the table headers is German. But we'll change this in a second.



Go to the property window to set up the language strings for the table header. A quicker way is to right click on the control and select properties. In this property pages you can simply click on Set English to get English strings. Then click apply. On the color and font page you can change the colors and fonts used inside of the table.

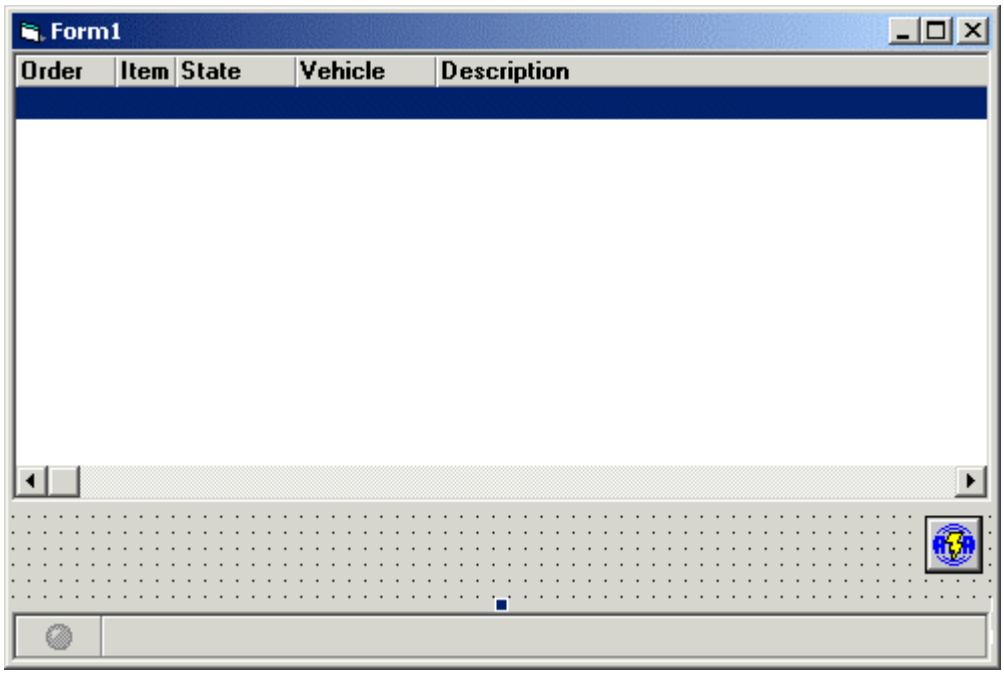




Next place an AxClientStatus control on the bottom of the form (see below). This control displays a LED and a status line. The led is grey if no connection to network manager has been made. If a connection is established, the led will change to green. If the connection is lost, the led goes to red. The status line is used to show some more information about the selected order.

Both controls have an AxBorderStyle property. Here we use the afbSunken style. You may want to play with the other values as well.

Your form should now look like this:



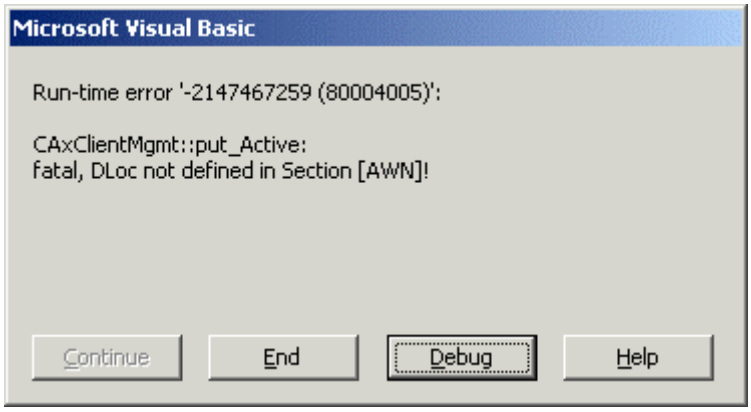
Before we start our application for the first time, we need to create two event handlers for the form itself. In the Load event we link our controls together and enable the AxOrderMgmt control, in the Unload event we disable our control and save the column widths back into the ini file:

```
' link the controls together and
' activate the AxOrderMgmt control
Private Sub Form_Load()
    AxClientOrders1.StatusBar = AxClientStatus1
    AxClientMgmt1.OrderDisplay = AxClientOrders1

    AxClientMgmt1.Active = True
End Sub

' disable control
Private Sub Form_Unload(Cancel As Integer)
    AxClientMgmt1.SaveColWidths
    AxClientMgmt1.Active = False
End Sub
```

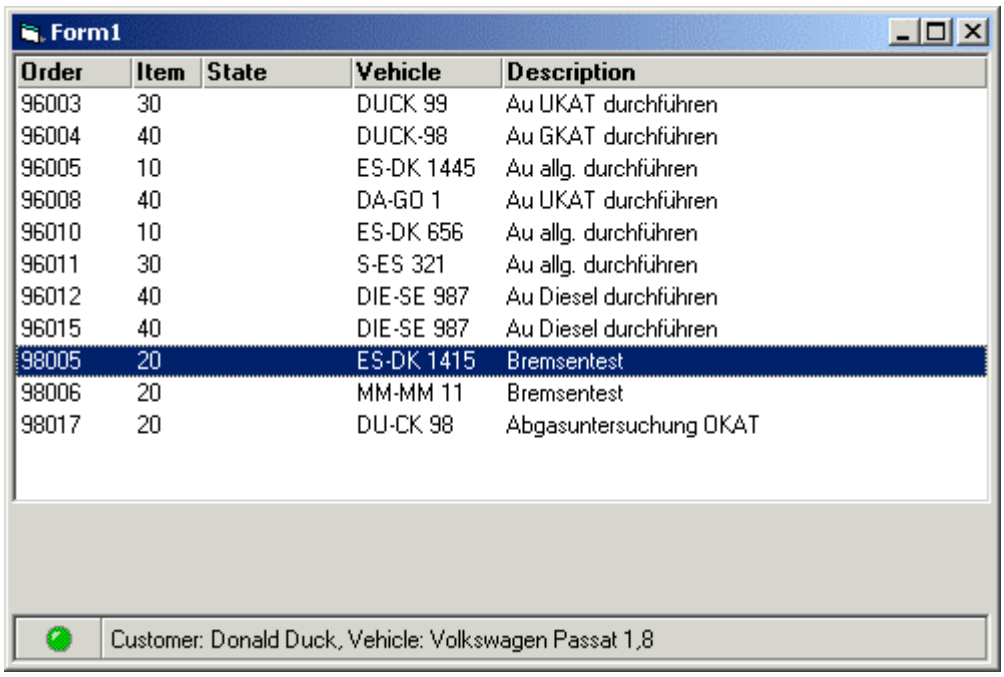

Run the application.

If you get an error message like the one on the right side the control probably didn't found the ini file.

This is most often a result of the wrong working directory. Either close the project, change to your demo1 project directory and double click on the demo1.vbp file – or – use an absolute path for the ini file.



Now you should see a dialog similar to the one below. If you don't see any orders please ensure that the network manager is running and that you have orders available. To create orders you can simply start the order generator from the asanetwork SDK.



Examine some of the features. You can sort each column with a click on the column header. You can change the widths of the columns. If you click on an order the status line displays additional information.

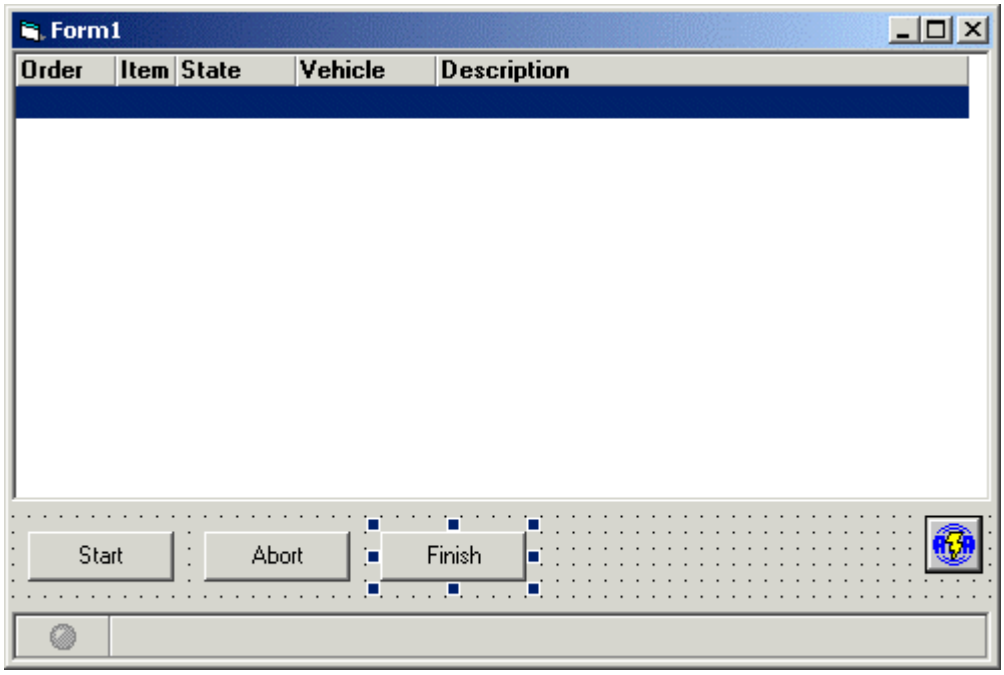
What is displayed?

- The first column contains the order number.
- The second column is the position (item) number of this order.
- The third column displays the order state together with a bitmap (see Design guide for a description of these symbols).
- The forth column displays the license plate of the vehicle.
- The fifth and last column display a verbal description of this order position (item).

The bottom line shows a green led – we are currently connected to the network manager. The remaining space is used for a status line. Here the control displays information about the customer and vehicle for new orders.

Step 2, processing orders

In this step we'll add some buttons to our application to start, abort and finish orders.



Add 3 buttons to your form. Call them "start", "abort" and "finish".

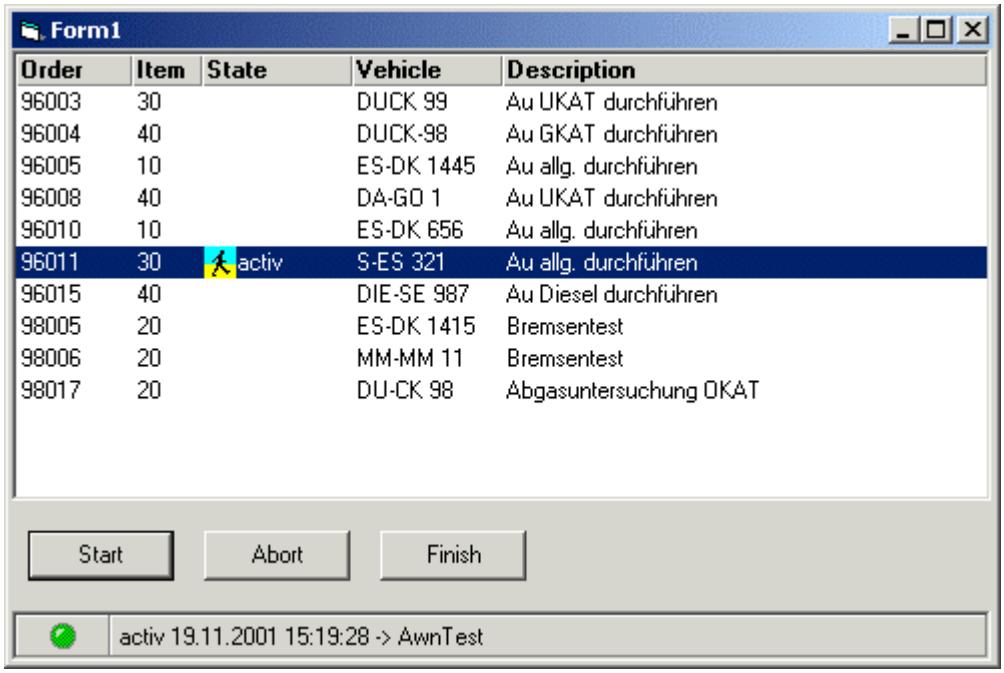
Double click on the buttons to set up the event handlers. Add the code shown below to each of the button's event handlers.

```
' save the selection
' start the selected order
Private Sub Start_Click()
    AxClientMgmt1.SelOrder = AxClientOrders1.ItemIndex
    AxClientMgmt1.StartOrder
End Sub

' order aborted by user
' set result flag to rAborted
Private Sub Abort_Click()
    AxClientMgmt1.AbortOrder rAborted, "aborted by operator", ""
End Sub

' order finished, transmit result data
' in our demo we use an existing file called gas.xml
' set result according to your testing, e.g.
' rOk, rWarning, rBad, rDanger, rTimeout, rOverflow
Private Sub Finish_Click()
    AxClientMgmt1.AbortOrder rOk, "tests passed", "gas.xml"
End Sub
```

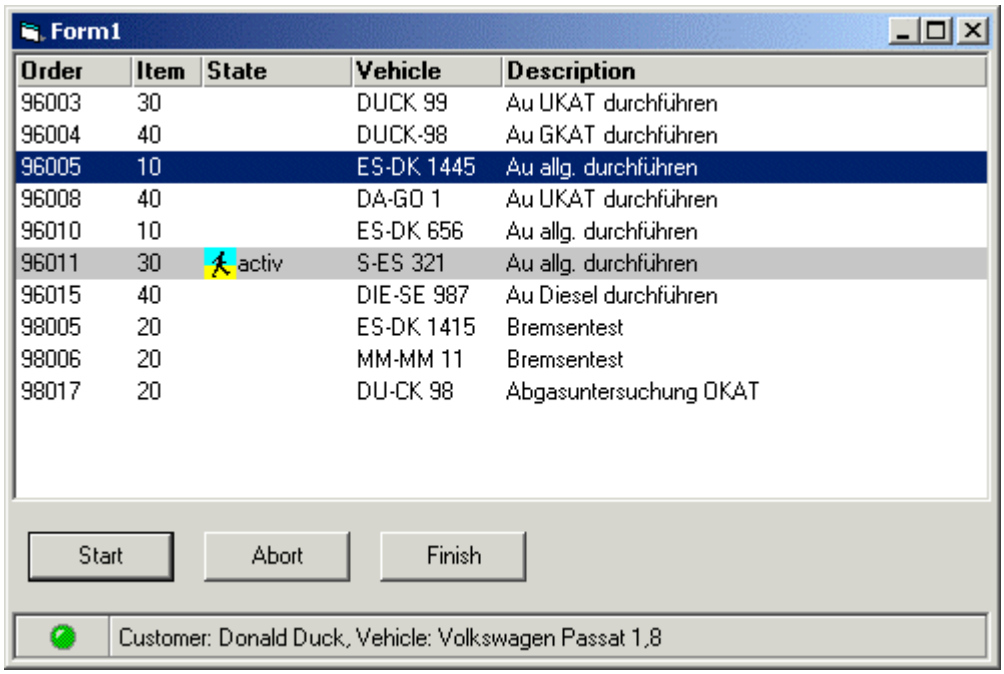
To start an order we use the StartOrder() method of the AxClientMgmt control.

Before we can successfully call any of these order related methods we need to define the order we want to process. This can be done either by assigning a valid value to the SelOrder property or by calling the ItemIndex property of the grid control. ItemIndex returns the index of the currently selected item in the grid control. Assign this index to SelOrder. If the selection is valid, SelOrder is greater than zero. If SelOrder is zero after the assignment, the order is either in the wrong state (i.e. not neutral) or the index is invalid. This selection is used for all further actions like AbortOrder, FinishOrder or InterruptOrder. A selected order is displayed with a gray background (default) in the list. As long as you have not started the order, SelOrder may be changed.

```
' save the selection
' start the selected order
Private Sub Start_Click()
    AxClientMgmt1.SelOrder = AxClientOrders1.ItemIndex
    AxClientMgmt1.StartOrder
End Sub
```

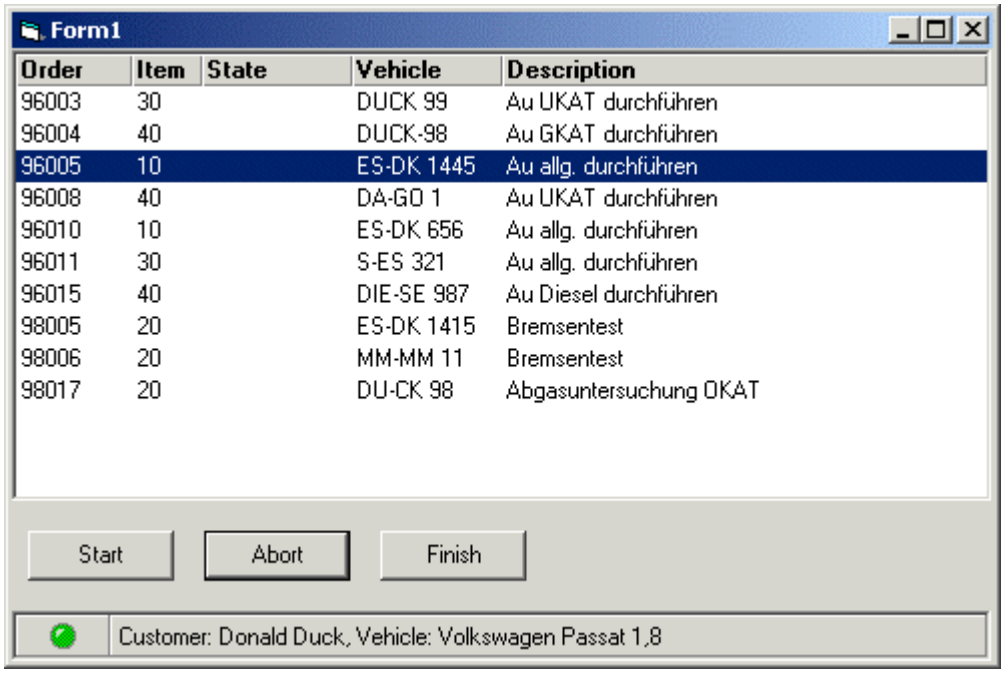


Select an entry in the list and click on the start button. If a customer order system is running (or the order generator) you'll see that the state changes to active.



If you select any other item inside the list, the currently assigned SelOrder is displayed with a grey background while the status line displays information about the new selected item.

Now click on Abort. The order immediately is set to the neutral state. You only need to call AbortOrder().



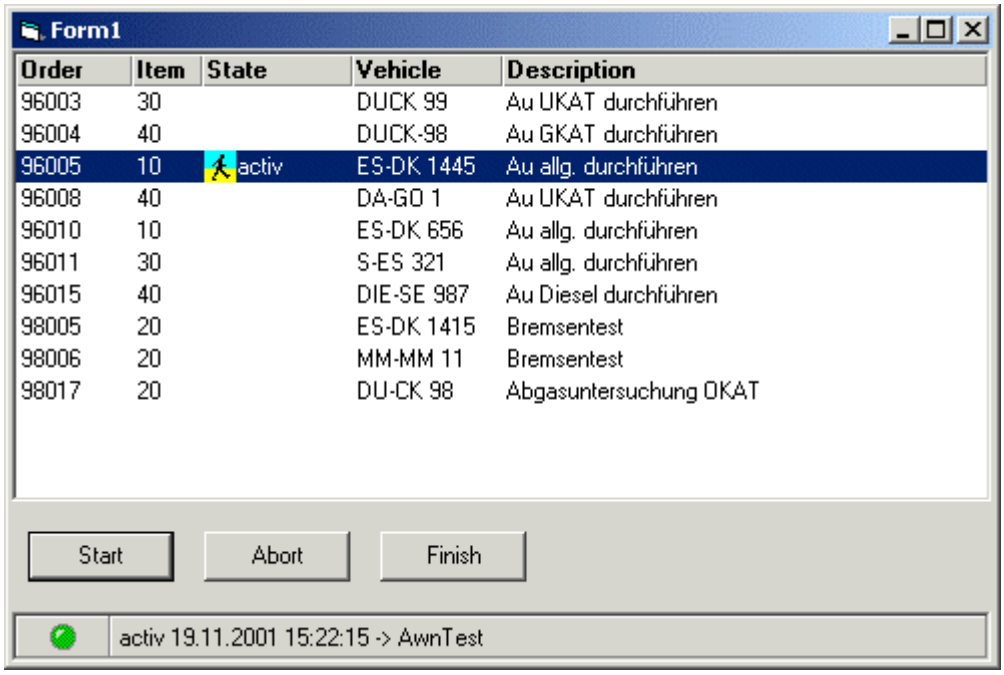

```
' order aborted by user
' set result flag to rAborted
Private Sub Abort_Click()
    AxClientMgmt1.AbortOrder rAborted, "aborted by operator", ""
End Sub
```

The first parameter is the result code. Here we use rAborted.

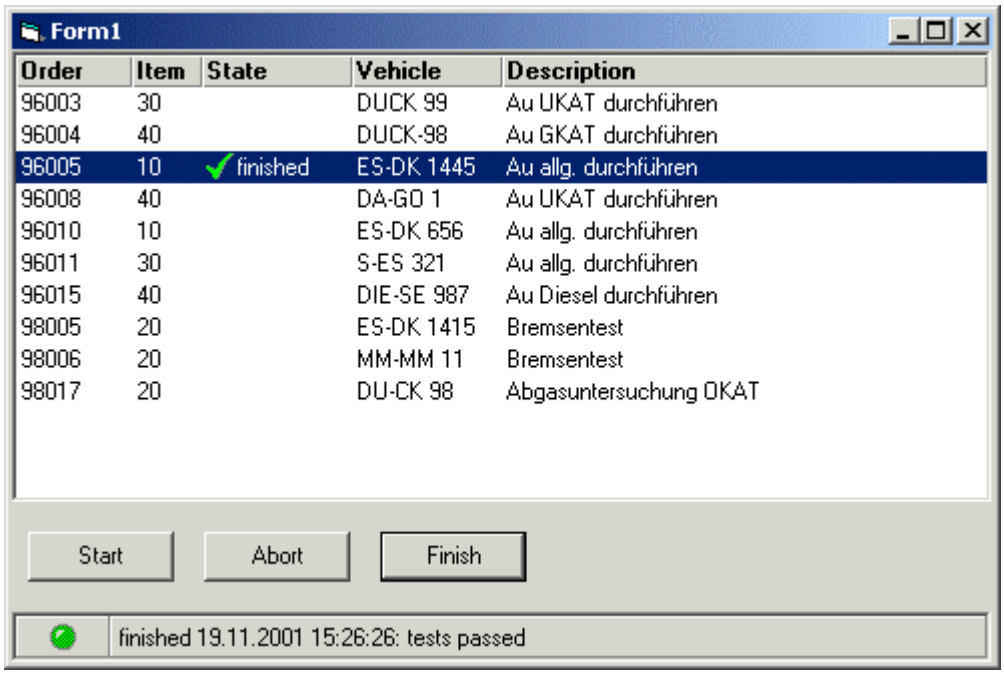
The second parameter is a brief description. Here we put in "aborted by operator".

The last parameter is the path name of a file containing result data. If the order was aborted after you already stored some results you can supply the file name here. In our demo we do not use this parameter.

Start the selected order again. The status line has changed and now displays the start time of processing and the device (DLoc).



Now we're going to finish this order. Click the finish button. The order is immediately set to the finished state. You only need to call FinishOrder().




```
' order finished, transmit result data
' in our demo we use an existing file called gas.xml
' set result according to your testing, e.g.
' rOk, rWarning, rBad, rDanger, rTimeout, rOverflow
Private Sub Finish_Click()
    AxClientMgmt1.FinishOrder rOk, "tests passed", "gas.xml"
End Sub
```

The first parameter is the result code. In our demo we put in rOk. Therefore we get a green check mark.

The second parameter is a brief description. Here we put in "tests passed".

The last parameter is the path name of a file containing result data. In our demo we use a static file with some results from a gas analyser.

Once again the status line changed and now displays the end time of processing and a verbal description of the result code (not the brief result).

Step 3, more properties for orders

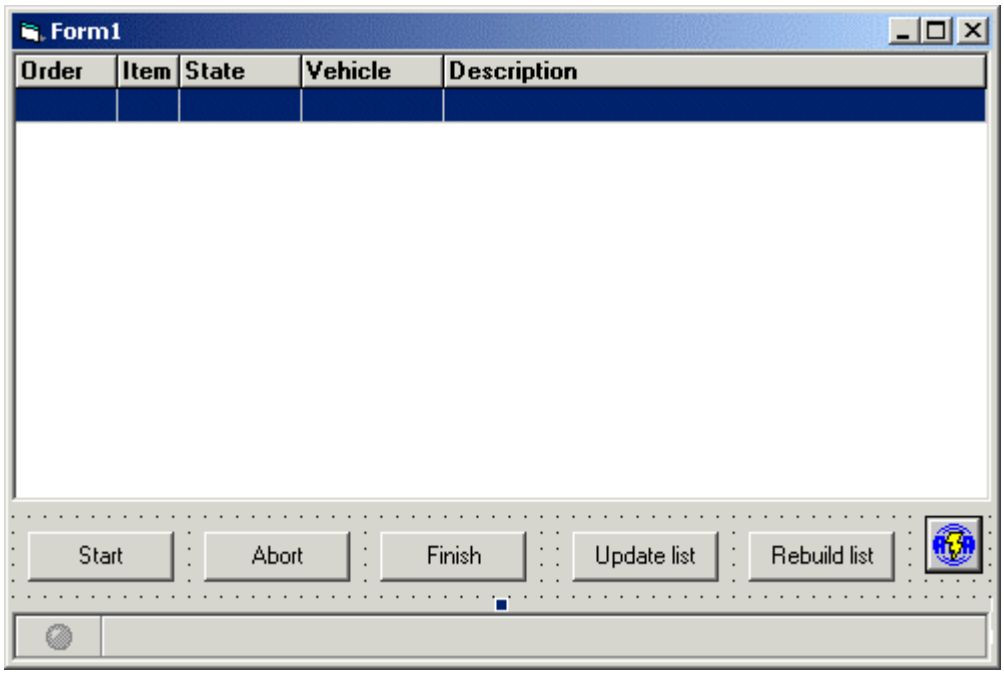
In this step we'll show some more properties of the control: The QueryOnStart and DeleteOnQuery property.

Let's start with QueryOnStart. If QueryOnStart is true (default) the control automatically sends a query for new and active orders to the asanetwork. In this way you always make sure, that orders are displayed. As long as you don't provide any way to update the list manually, you should not change this property.

DeleteOnQuery (which defaults to false) controls what happens with any existing list. If false the list is not changed before a query is executed. If true, the list is cleared (deleted) and rebuild from the query.

What's the difference? If the list is not changed, finished orders remain in the list. If you do any query, only new and active orders are updated. Finished orders are only removed from the list if the dealer management system deletes the whole order. If you clear the list finished orders are removed and the list is rebuild with neutral and active orders only.

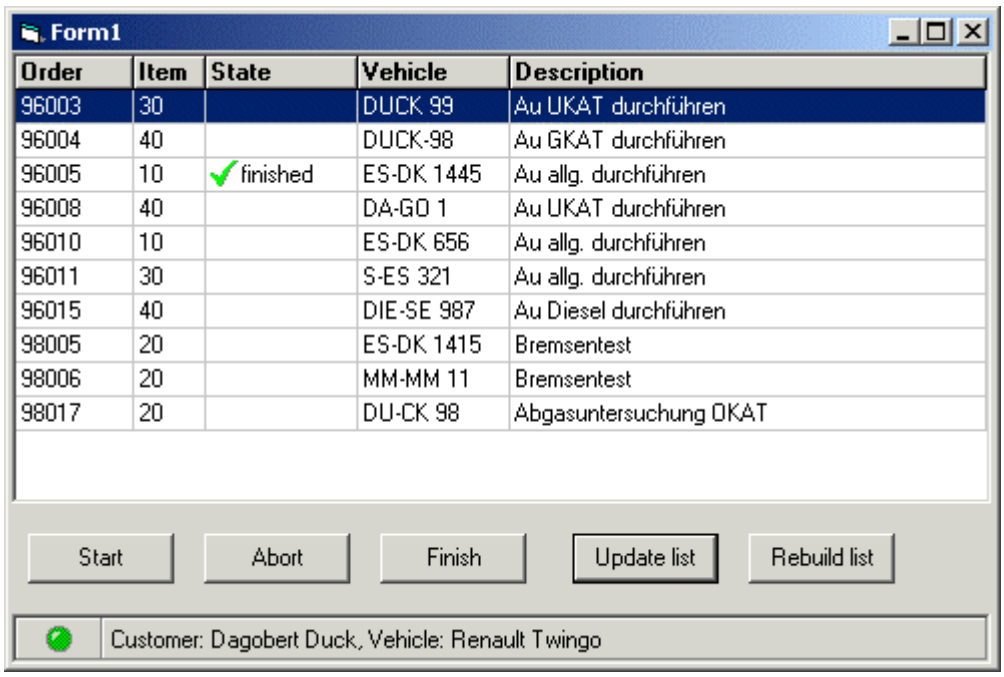
The best approach is to provide both ways to the user. Add 2 buttons called "Update list" and "Rebuild list". Set both QueryOnStart and DeleteOnQuery to false.



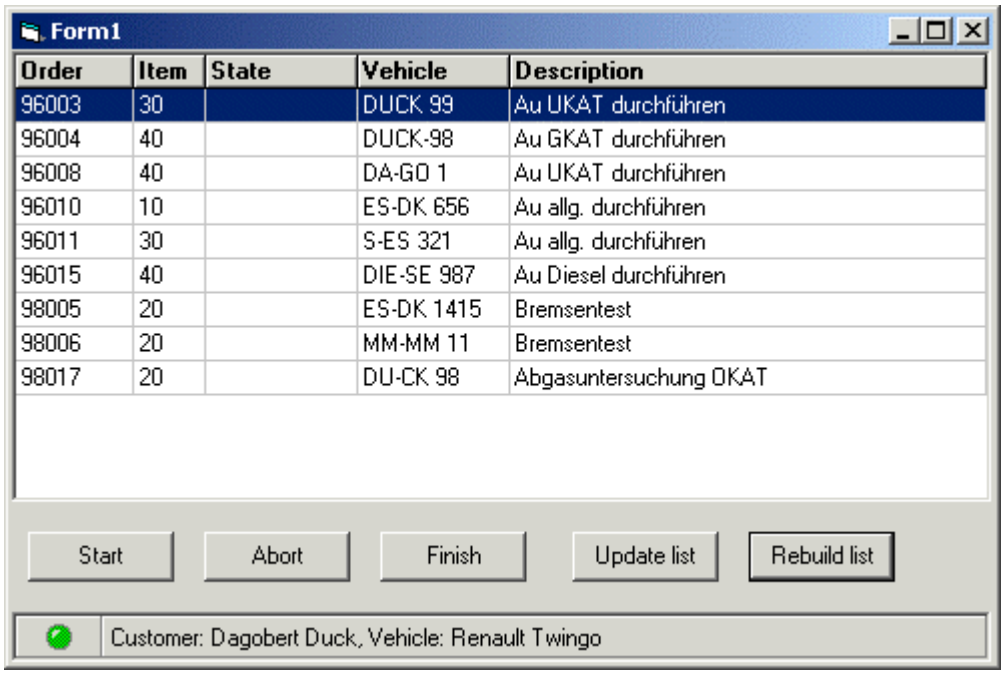
Add the code for the event handlers as shown below:

```
' clear and rebuild the list
Private Sub Rebuild_Click()
    On Error Resume Next
    AxClientMgmt1.ClearList
    AxClientMgmt1.QueryOrder "*", AxAwnPosQueryAll
End Sub

' update list entries
Private Sub Update_Click()
    AxClientMgmt1.QueryOrder "*", AxAwnPosQueryAll
End Sub
```

Note the error handler for the ClearList method. ClearList may fail if you have active orders in the list.



Click on update list, the finished order is still displayed.



Click on rebuild list and the finished order is removed.

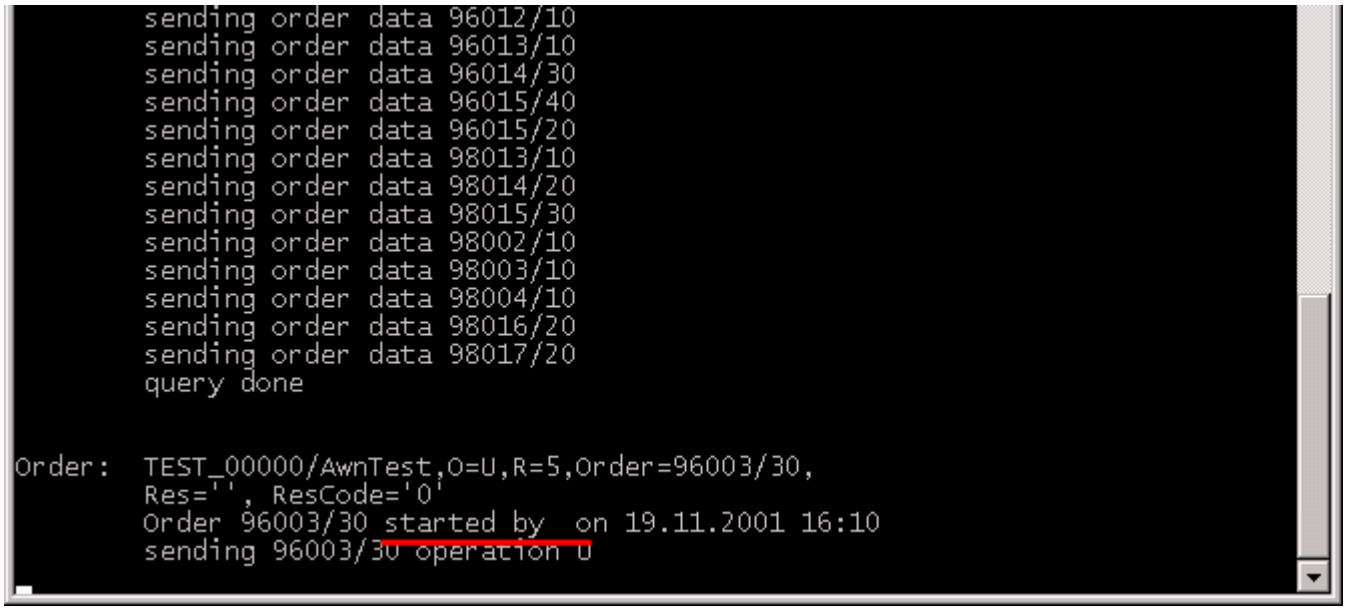
If you look at your working directory you may notice a file called state.dat. This file is used to persistently save the order list if your application is closed. If your restart your application this file is used to fill the order list.

You may also have noticed that the list is displayed with grid lines. There's a property called ShowGrid, if it's true the lines are displayed.

Step 4, Accessing order data

This step shows how we can easily access and change the data before the order is sent to the net.

If you look very carefully at the start order command, you'll notice, that the name of the operator is missing. This is true, because this information has to be supplied by the application. Let's add this information now.

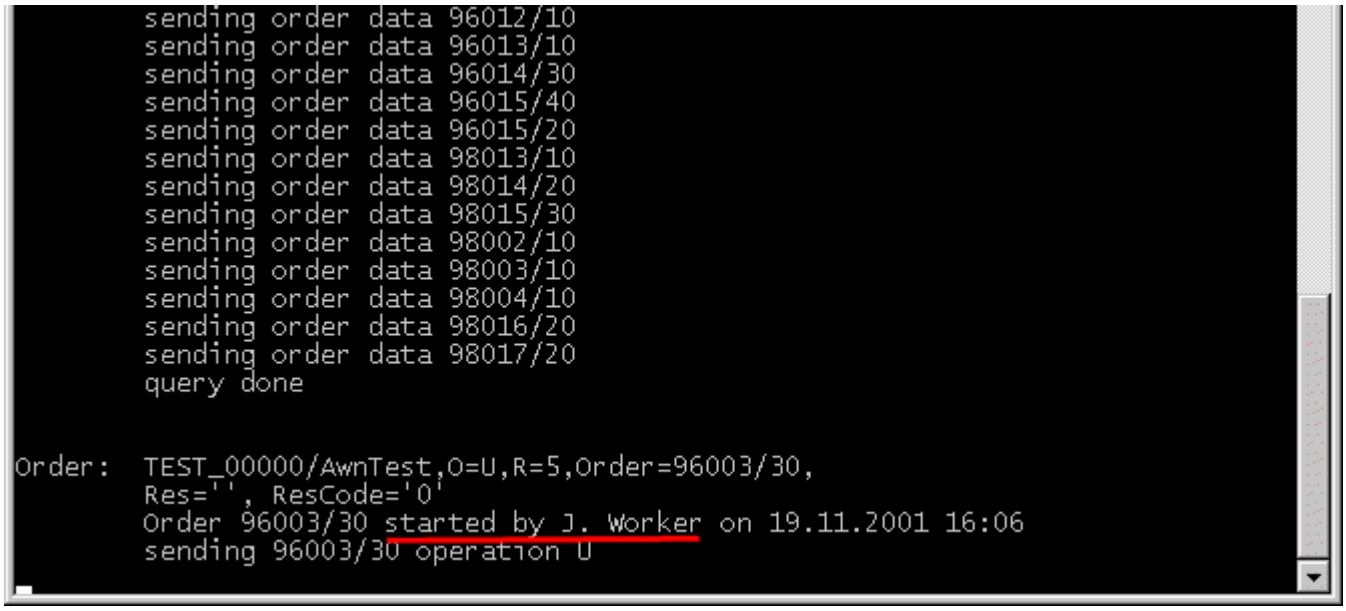


For every order action there's an OnBeforeXXX event. To add the operator name we use an OnBeforeStarted event. This event is created after you called the StartOrder() method but just before any data is transmitted. So it's the ideal place to do any data modification.

```
' modify order data
' transmitted with the StartOrder() method
Private Sub AxClientMgmt1_OnBeforeStarted()
    With AxClientMgmt1.CurrentOrder
        .OrderRealWorker = "J. Worker"
    End With
End Sub
```

In the OnBeforeXXX event handlers we have access to the selected send and order data via the properties CuurentSend and CurrentOrder. Note that CurrentSend and CurrentOrder are only valid if you have already called StartOrder!

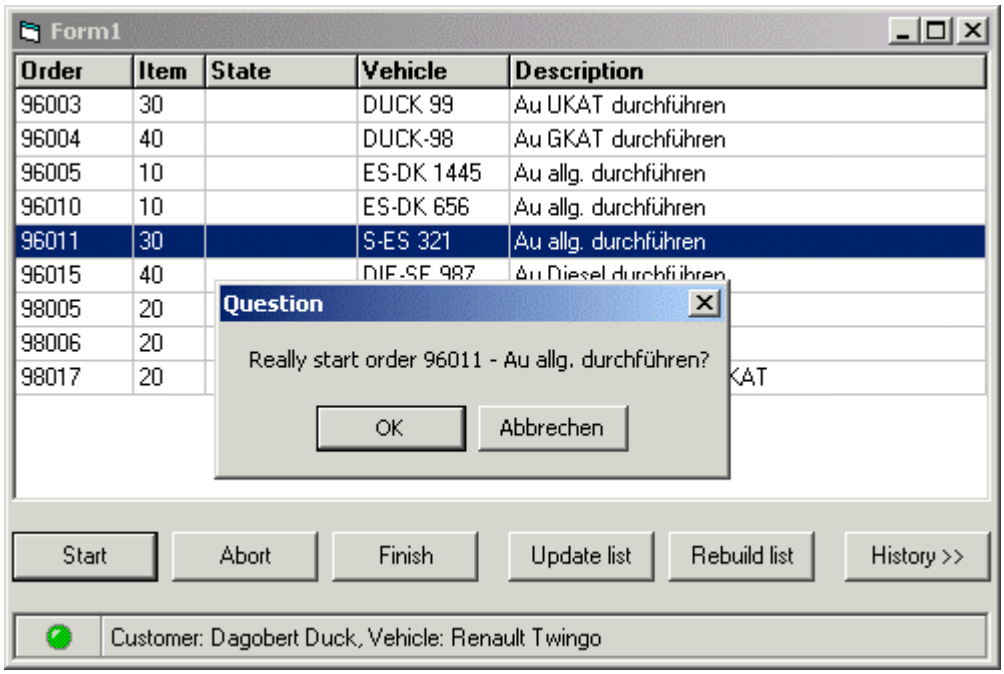
If you start the application again and start an order the order generator now contains the name of the operator (see below).



But what if we want to access order data before we called StartOrder? As described above we can't use CurrentOrder because it's undefined before a call to StartOrder was made. In the next example we want to display a dialog before the order is started. For the dialog we need some data (order number and verbal description).

To access the data we need to get the index from the list and directly access the data in the list. The list is an AxRefList object and accessible via the OrderList property.

```
' access order list and ask user then
' save the selection and
' start the selected order
Private Sub Start_Click()
    Dim index As Long
    Dim OrderObj As AxAwnOrder
    Dim SendObj As AxAwnSend
    Dim Msg As String

    index = AxClientOrders1.ItemIndex
    With AxClientMgmt1

        Set SendObj = .OrderList.Item(index)
        Set OrderObj = SendObj.Data

        Msg = "Really start order " & SendObj.Reference.Order
        Msg = Msg & " - " & OrderObj.OrderTxt & "?"

        ' now ask user
        If MsgBox(Msg, vbOKCancel, "Question") = vbOK Then
            .SelOrder = index
            .StartOrder
        End If
    End With
End Sub
```

You should never store the index in a variable for later processing. The list of orders always may grow or shrink. So it's very likely that your index is no longer the same object as before. Always use DispIndex or SearchOrder() to determine the index number. If you already have an selected order use the SelOrder property. SelOrder is automatically adopted if the list grows or shrinks.

Step 5, Crash handling

The control automatically keeps track of started orders in the state.dat file as described above. If any order is active, a dirty flag is written tio the ini file. If your application terminates unexpectedly, the previously started order is automatically aborted on restart if the dirty flag is present. If you close your application, the dirty flag is removed and the previously started order is automatically resumed on restart. To demonstrate this feature, start an order and then close the application. If you restart the program you'll notice the previous order is displayed again in the selected and active state.

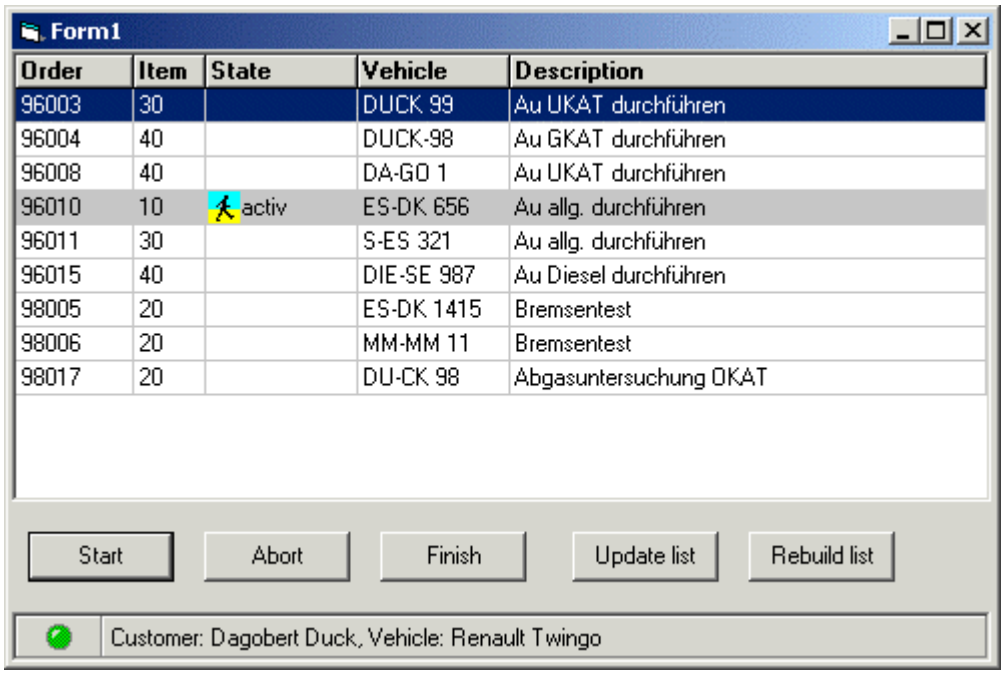
To give you more control over this process two events are available:

The OnCrash event only gives you the chance of showing information because generally it makes no sense to resume a crashed system.

The OnResume event allows you to either resume the order or abort the processing.

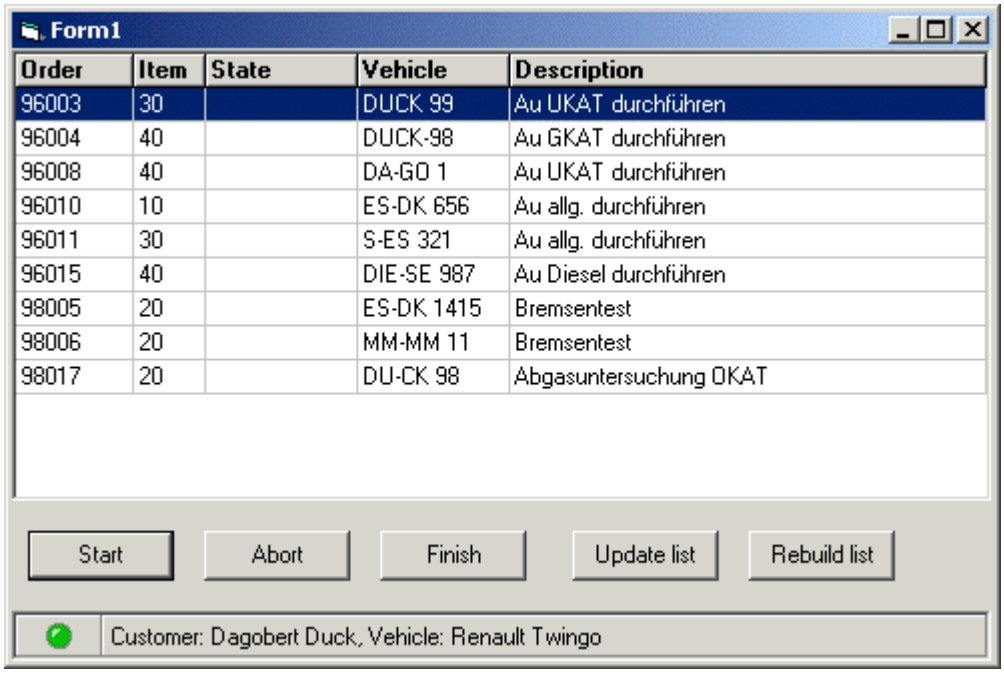
```
' crash handler
Private Sub AxClientMgmt1_OnCrash(ByVal Order As String, ByVal Pos As Long)
    MsgBox "Application crashed, order " & Order & " aborted!", vbOKOnly
End Sub

' resume or abort previously active order
Private Sub AxClientMgmt1_OnResume(ByVal Order As String, ByVal Pos As Long)
    If MsgBox("Resume order " & Order & "?", vbOKCancel) = vbOK Then
        ' again use order data for processing
    Else
        AxClientMgmt1.AbortOrder rAborted, "aborted by operator", ""
    End If
End Sub
```

If you click OK the previously selected order is again selected and active:



Now kill the application (e.g. use the task manager, go to the process tab, select demo1 and kill the process). Restart the application.

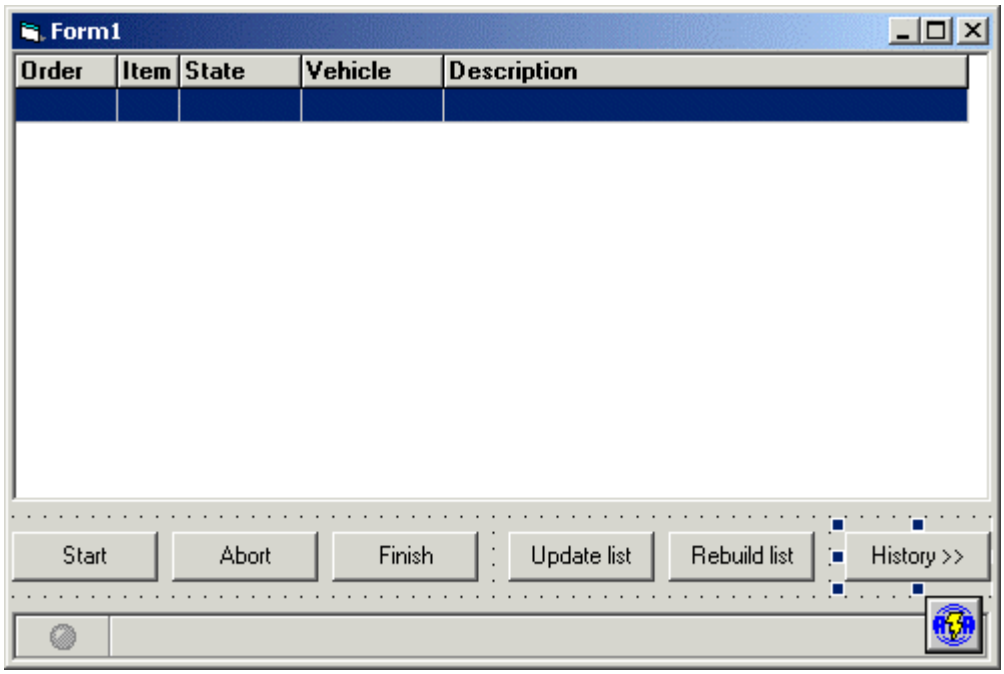
Now the order is reset:



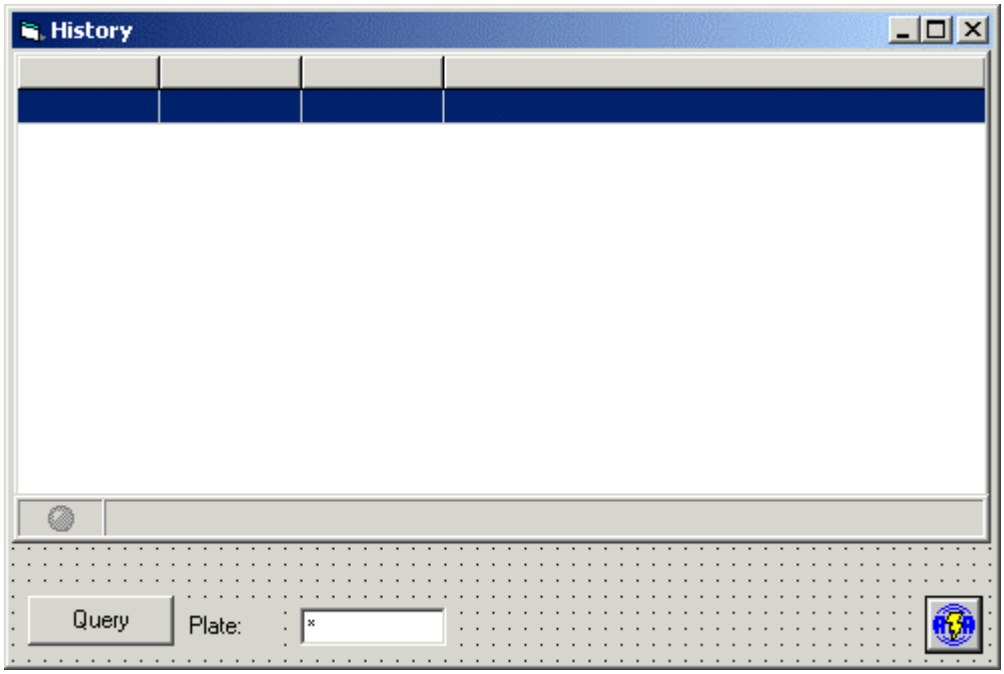
Building a history client

Step 6, the basic form

We'll now leave the order handling and start to build a history client. A history client is used to get previously saved results back from the asanetwork. You can then display these old results or use them for comparisons. To display the history form, let's add a new button on the order form called History. Later on we add the OnClick code for this button.



Create a new form and again add an AxClientMgmt control. In the same way as before add An AxClientHistory and AxClientStatus control. Here we use the afbRaised value for AxBorderStyle and place the status line below the list. Add a button called "Query" and a textbox called plate for the license plate of the vehicle. Save the form as HistoryForm.



Select the AxClientMgmt control and change the mode property to mHistory, QueryOnStart to false, DeleteOnQuery to true and add the Ini file and License as before.

Again add some code to the Load and Unload event of the form to enable or disable the control.

Finally add an event handler for the query button. In this event handler we call the QueryResult method of the control. The first parameter is the service used to send the query, if you enter an empty string, all services defined in the ini file are queried. The second parameter is the license plate we're looking for and the last parameter is the time limit in months. We use 0 = unlimited.

```
' enable control
Private Sub Form_Load()
    AxClientHistory1.StatusBar = AxClientStatus1
    AxClientMgmt1.ResultDisplay = AxClientHistory1

    AxClientMgmt1.Active = True
End Sub

' disable control
Private Sub Form_Unload(Cancel As Integer)
    AxClientMgmt1.Active = False
End Sub

' create a simple query for results
Private Sub Query_Click()
    AxClientMgmt1.QueryResult "", Plate.Text, 0
End Sub
```

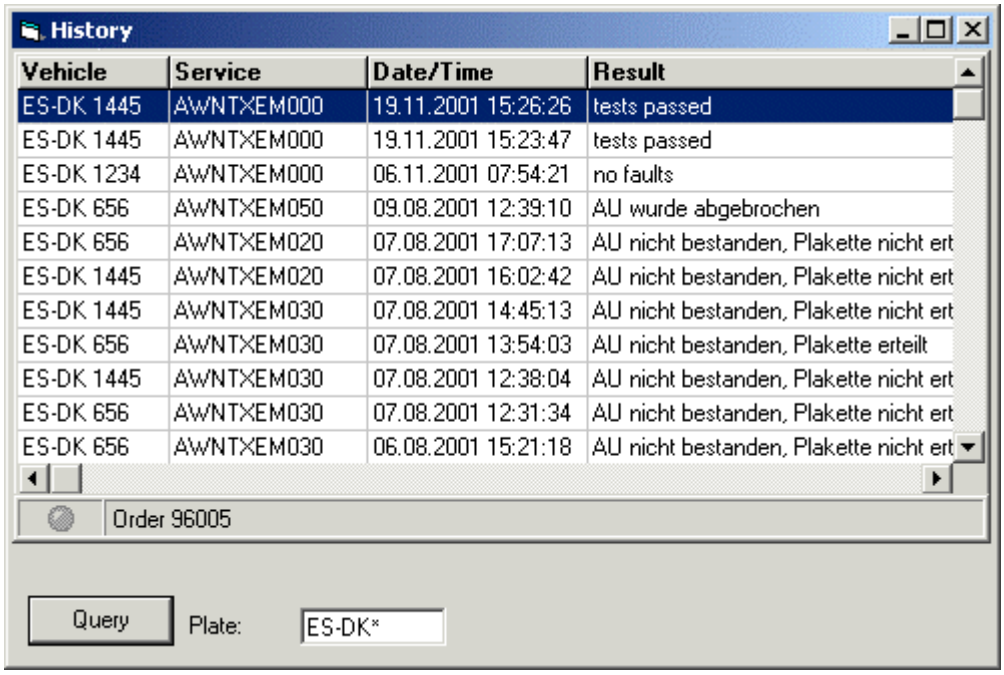
Back in the order form add this event handler for your "History" button:

```
' show history form
Private Sub History_Click()
    HistoryForm.Show
End Sub
```

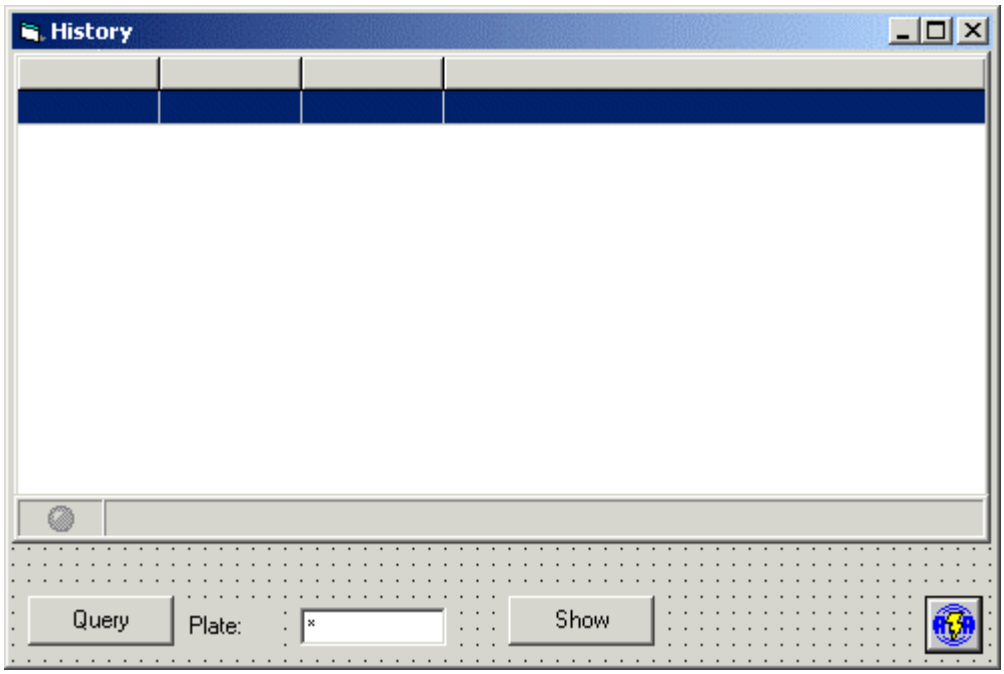
Start your application and enter a plate number or a piece of a plate number then click query. The control creates a query for every defined service and retrieves the results. Your display now looks like this:

- The first column shows the license plate
- The second column shows the service which created this result
- The third column shows date and time of creation
- The last column shows the brief result

All results have been retrieved and are stored as files on your hard disk (in the TempDir directory). You have direct access to the AxAwnSend objects and the files using the ResultList property.



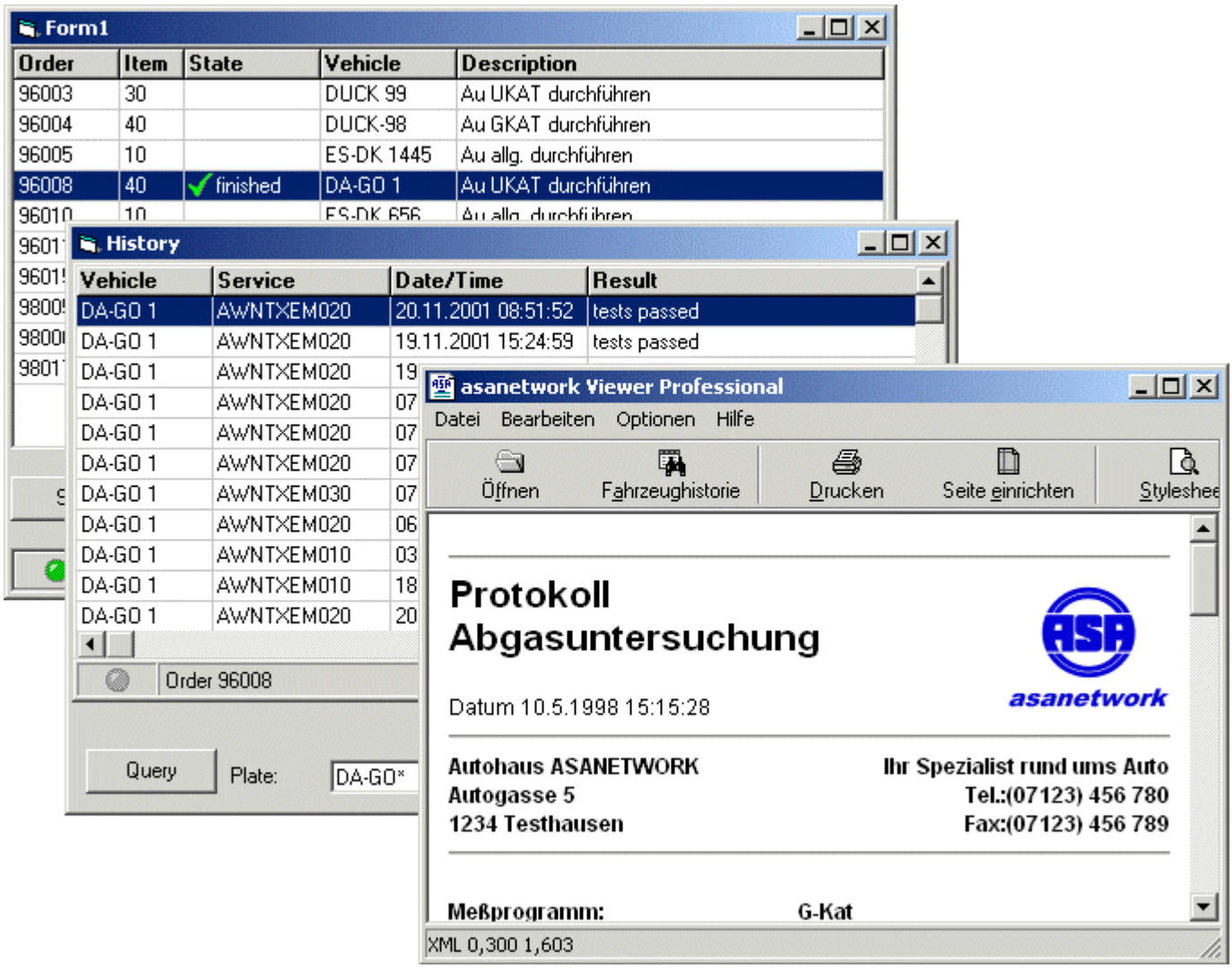
Now let's add a show button to display the selected result:



We use the ItemIndex property again to get the index of the focused item. Then we directly call the asanetwork viewer with the file name of the selected result:

```
' display selected result with the asanetwork viewer
Private Sub Show_Click()
    Dim index As Long
    Dim cmdLine As String

    ' get index of selected item
    index = AxClientHistory1.ItemIndex
    ' set up path to viewer, if you have the pro version change the exe name
    cmdLine = "c:\progra~1\axonet~1\awnview2\awnview2.exe "

    ' check if we have enough results
    If AxClientMgmt1.ResultList.Count >= index Then
        ' append file name to cmdline
        cmdLine = cmdLine & AxClientMgmt1.ResultList.FileName(index)

        ' start viewer
        Shell cmdLine, vbNormalFocus
    End If
End Sub
```

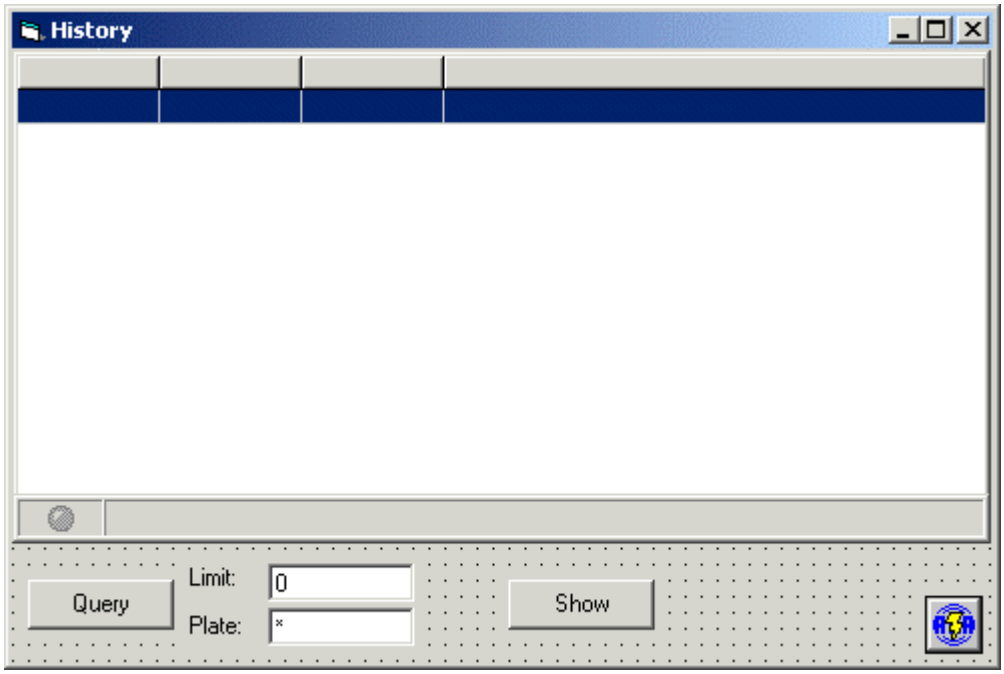

Start you application again, perform a query, select an item and click the "Show" button. The viewer starts up and displays the results:



Step 7, Adding a time limit

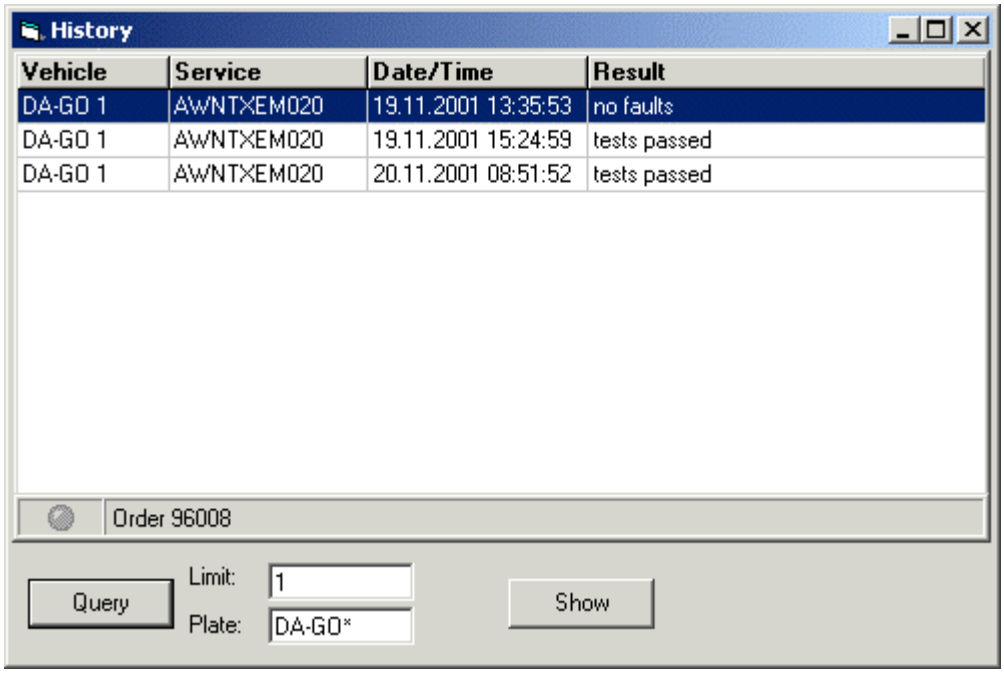
In this step we add the possibility to limit the query for a given period of time.

Add a textbox for the input of the time limit. The time limit works in units of months. If you enter e.g. 1, you'll only receive results which are up to 1 month old. if you enter 12 you'll receive all results of the last year.

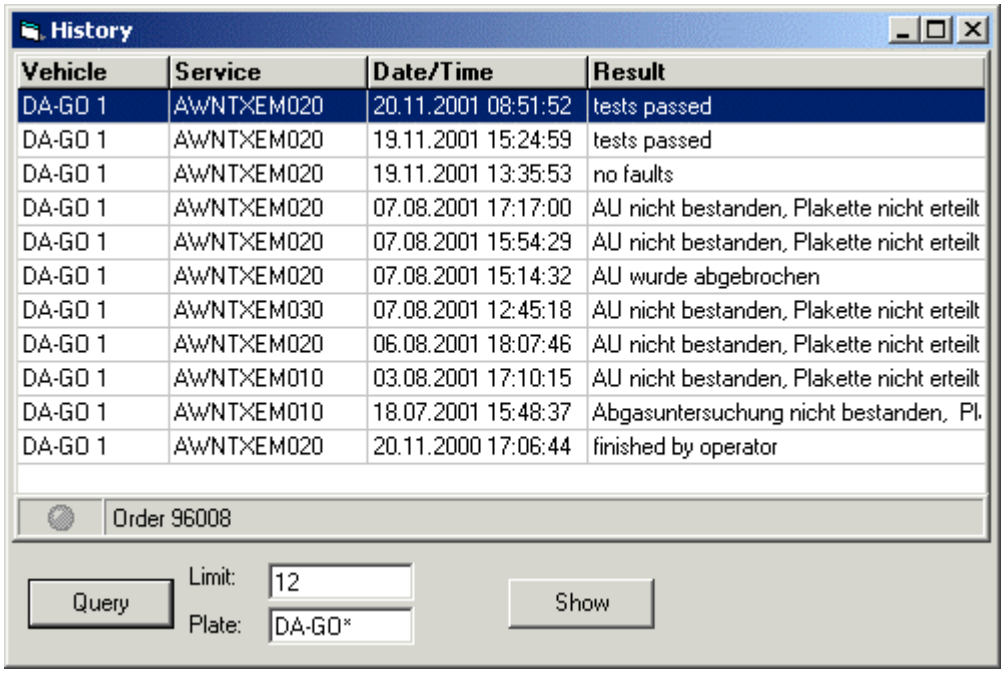



```
' create a time limited query for results
Private Sub Query_Click()
    AxClientMgmt1.QueryResult "", Plate.Text, Int(Limit.Text)
End Sub
```

First example, limit = 1 month (compare this with the next screen shot).



Second example, limit = 12 months.



Step 8, More properties

Again there's the DeleteOnQuery property. If false, you can accumulate different queries, if true the lists are deleted first. Try it.

Another property used with queries is the Timeout property. If you start a query the control waits for a response from the network manager. If there is no response in a given time (= Timeout value) the control aborts the query.

While the query is running every received item creates an OnQuery event. If the query is finished (either the last item was received or there was a timeout) an OnQueryEnd event is created.

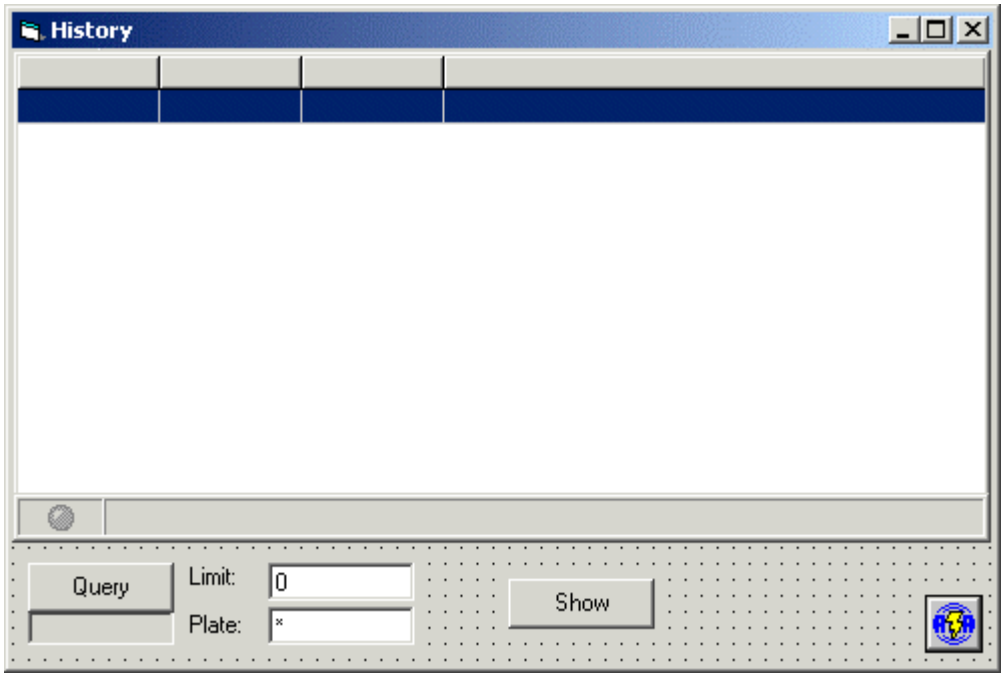
We'll use these events to create a blinking panel while a query is running together with a display of the number of received items.

Add a label under the query button. Set BorderStyle to "1 – Fixed Single".

On start of our query we display the word "running".

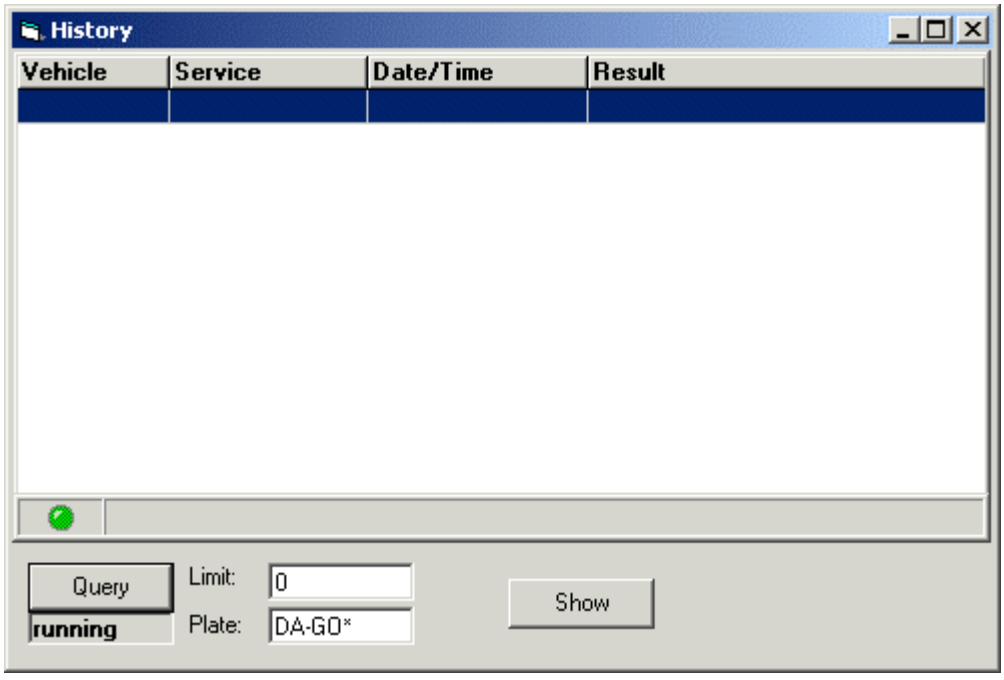
On every OnQuery event we'll toggle the background color of the label between blue and the standard color. The label caption displays the word running and the number of received items.

If we receive an OnQueryEnd event, we'll reset the background color and display the total number of received items.

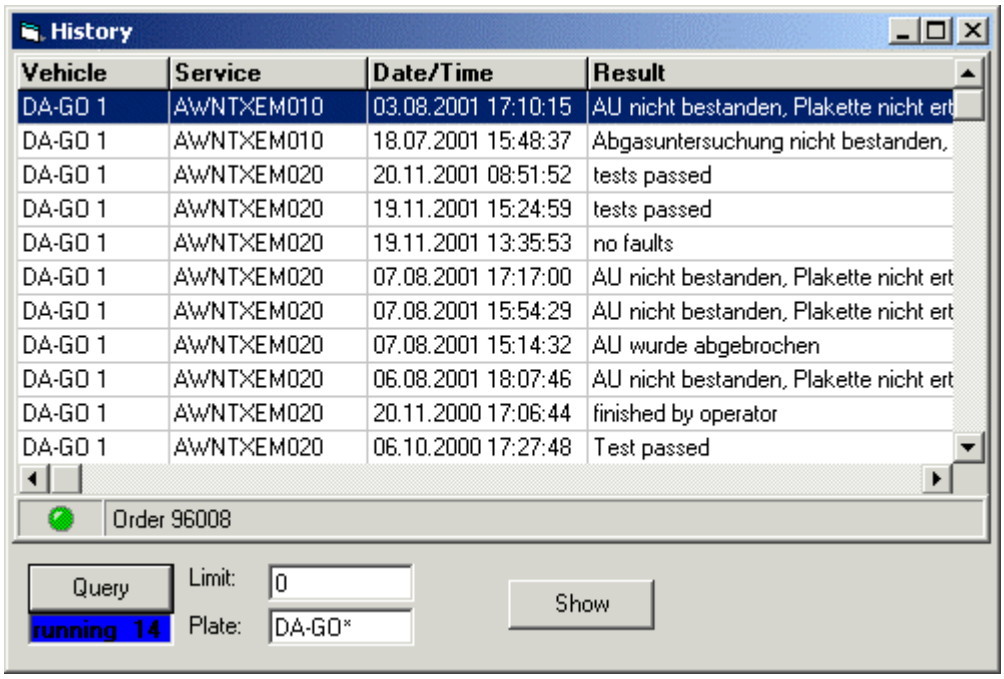



```
' create a time limited query for results
Private Sub Query_Click()
    Label3.Caption = "running"
    AxClientMgmt1.QueryResult "", Plate.Text, Int(Limit.Text)
End Sub

' show number of received items and toggle color
Private Sub AxClientMgmt1_OnQuery(ByVal cnt As Long)
    If Label3.BackColor = &H8000000F Then
        Label3.BackColor = &HFF0000
    Else
        Label3.BackColor = &H8000000F
    End If

    Label3.Caption = "running " & Str(cnt)
End Sub

' query finish, reset color and show results
Private Sub AxClientMgmt1_OnQueryEnd(ByVal cnt As Long)
    Label3.BackColor = &H8000000F
    Label3.Caption = "finished " & Str(cnt)
End Sub
```

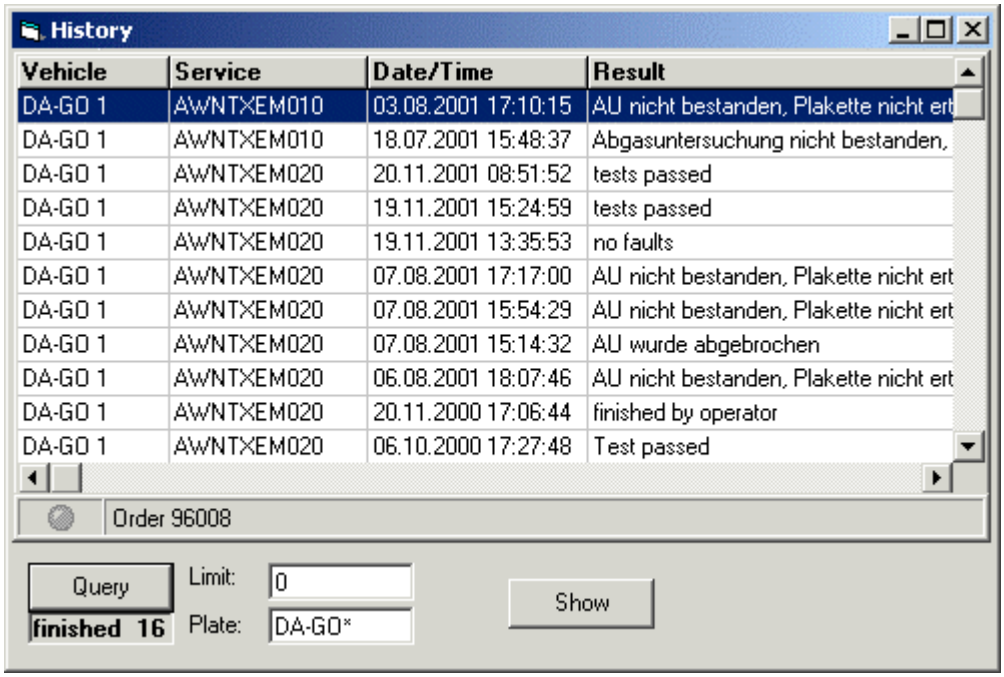
Query started, notice that the led is green.



14 items received:



Query finished with 16 items, notice that the led is now off.



Topics not covered

There are some additional properties and methods not mentioned in this introduction. Please have a look at the online help file for additional information:

Methods (items marked with * are for experienced users only)

StoreResult – save results without an associated order
SearchOrder – search the list for a specific order and return the index
* InsertSend – create a new position for a given order
HasVehicle – returns true if order has extended vehicle data available

Properties

ReuseFinished – if false (default) you can't start an already finished order again
ResultRef – determines the reference used for history queries: rrLicensePlate, rrVehicleIdentification or rrOrder
Vehicle – returns an extended vehicle data object

