

TYX CORPORATION

Reference	TYX_0051_21
Revision	1.0
Document	MultipleMeasurements.doc
Date	September 19, 2008

How to perform multiple measurements using one ATLAS statement

1	ATLAS multiple measured characteristic for the IEEE 716-1989 subset	3
1.1	The BNF statements	3
1.2	ATLAS syntax diagrams	3
1.3	An ATLAS FETCH example	3
2	Creating device datatbase file - .ddb file	4
2.1	Static description Example	4
3	Creating WCEM code	5
4	A complete PAWS Project sample	10
4.1	The 716.89/PAWS Atlas code:	10
4.2	The Device Database (*.ddb) file:	11
4.3	The WCEM code (*.cpp) code:	12
4.4	The log file:	13

This document will help to generate a PAWS project that will enable the user to perform multiple measurements from one ATLAS statement.

Introduction:

- **First the document will address the issue of creating the ATLAS part in order to utilize the multiple measurements within one ATLAS statements. Then the details on how to create the device database file (.ddb) we will be covered. The focus will be placed on the allocation problems, which may result from using the multiple INTO fields in the ATLAS sensor statements and the inability to allocate them properly to the static description in the device database. At the end, the WCEM code will be presented to demonstrate the details of how to return the multiple measurements from the driver back to ATLAS.**

1 ATLAS multiple measured characteristic for the IEEE 716-1989 subset.

Below are included the BNF's and the ATLAS syntax diagrams presenting the usage of the **FETCH** in the context of multiple measured characteristic.

1.1 The BNF statements.

The processing of multiple measured characteristics in the ATLAS 1989 compilers is as follows:

```
<FetchStatement> :: FETCH <fd> (<PrimaryMChar> INTO <Var1> { <fd>
<AuxiliaryMChar> INTO <VarN>}...)<fd> <Noun> <fd>
<StatementCharacteristics>... <sd>
```

The first Measured Characteristic is considered the '**primary**' Measured Characteristic.

The optional following Measured Characteristic(s) are considered '**auxiliary**' Measured Characteristics.

1.2 ATLAS syntax diagrams

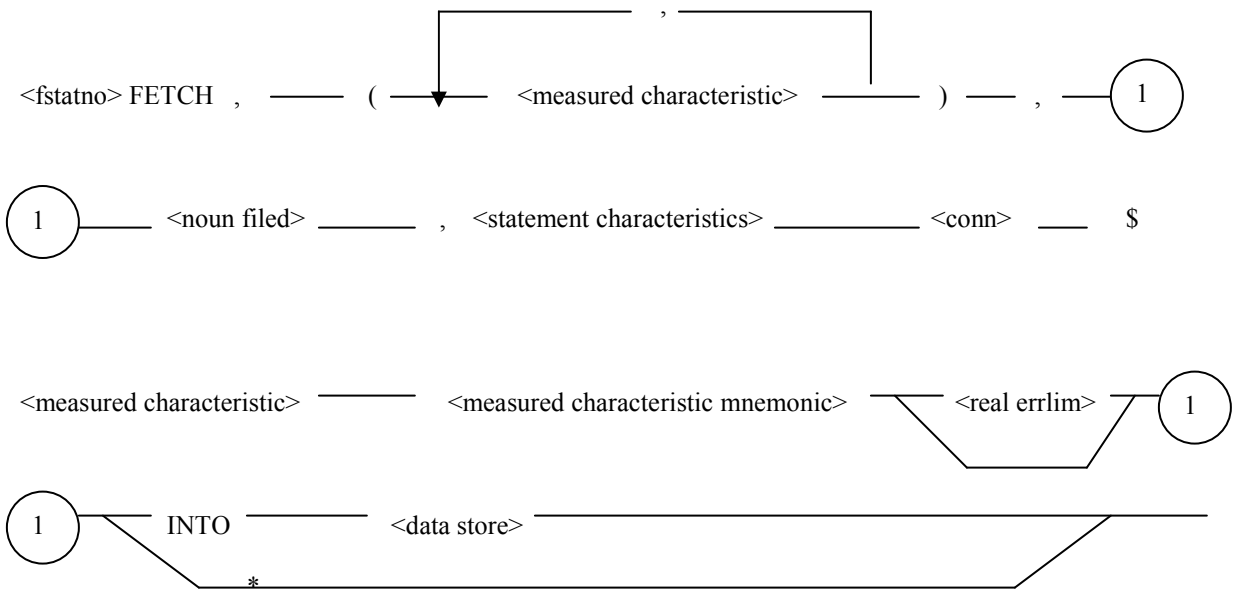


Figure 1.

* Only legal when the measured values will be stored in a RSP <array range> list.

1.3 An ATLAS FETCH example

In this example, the '**primary**' Measured Characteristic is the **frequency** and the **auxiliary Measured Characteristic** is the **voltage**. Each one of those characteristics is stored into separate variables:

```
FETCH, (FREQ INTO 'A', VOLTAGE INTO 'B'), AC SIGNAL,
      VOLTAGE RANGE 0 V TO 200 V,
      FREQ RANGE 10 HZ TO 500 HZ,
      MAX-TIME 1.0 SEC,
      CNX HI J2-1 LO J2-2
```

2 Creating device datatbase file - .ddb file.

In the Device Database the '**primary**' Measured Characteristic is modeled in the conventional manner as follows:

```
sensor (<PrimarMChar>) <Noun>;
```

The '**auxiliary**' Measured Characteristics are identified as a modifier associated with the <PrimaryMChar> sensor by enclosing the modifier in parenthesis, typically with a range specification, as follows:

```
(<AuxiliaryMChar>) range ...
```

2.1 Static description Example

```
begin FNC = 1;

  control
  {
    freq range 10 hz to 3e3 hz;
  }
  capability
  {
    (voltage) range 0 v to 300 v; * Auxiliary Measured Characteristic
  }

begin;
  sensor (freq) ac signal; * Primary Measured Characteristic
end;

end; *End FNC1
```

The example above can be used as a static description, which would allocate with the ATLAS statement presented in paragraph 1.3.

- Here the '**primary**' Measured Characteristic is the **frequency** and has a standard syntax.

```
sensor (freq) ac signal; * Primary Measured Characteristic
```

- The '**auxiliary**' Measured Characteristic, which is the **voltage**, is modeled as a modifier associated with the <PrimaryMChar> sensor by enclosing the modifier in parenthesis with the range:

```
{
  (voltage) range 0 v to 300 v; * Auxiliary Measured Characteristic
}
```

If both Measured Characteristics are modeled as in the above example, then the WCEM Wizard will be able to generate the code to return two readings:

1. MChar: "freq"
2. MChar: "voltage"

3 Creating WCEM code

The easiest way to deal with multiple measured characteristics from the WCEM code perspective is to use the automatic value retrieval using the WCEM Wizard.

Once the device database (.ddb) code is in place (the .ddb has to be built) and the WCEM module is added to the PAWS project, the WCEM Wizard can be used.

At this point, by navigating to the WCEM Wizard Device Database Map, the device tree (in this case the DMM) can be expanded as presented below:

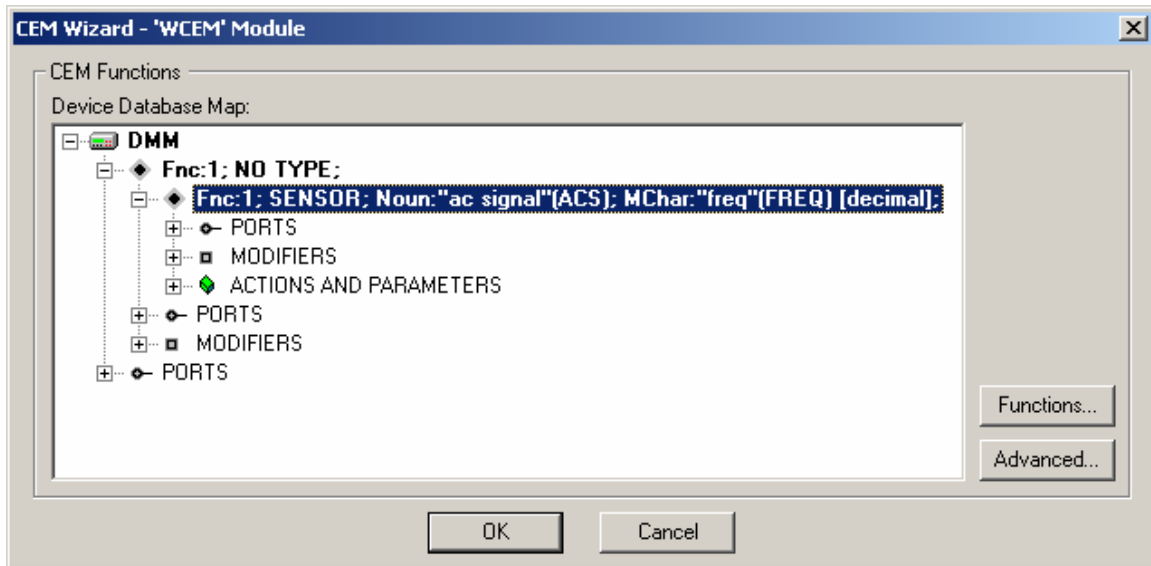


Figure 2.

As previously mentioned, the Fnc: 1 has the ability to return two measured characteristics: the frequency and the voltage.

The next step would be to enable the modifiers, which values need to be passed as the arguments to the **Interface function**.

Prior to doing this, at a minimum the **Fetch Interface function** has to be added to the WCEM code.

By right-clicking on the DMM entry in the **WCEM Wizard Device Database Map**, the following menu should be visible:

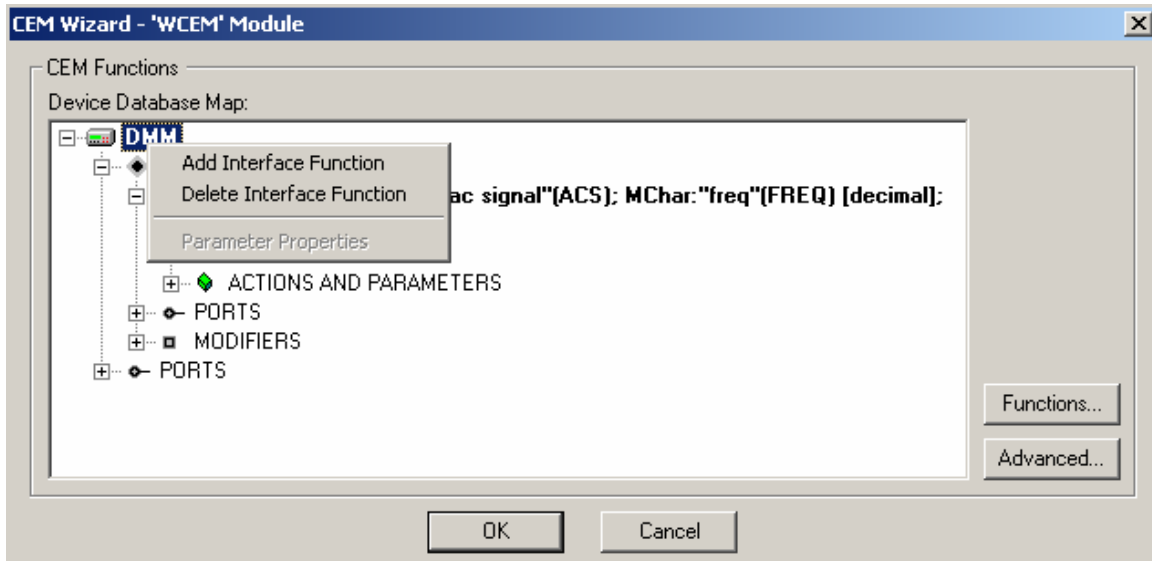


Figure 3.

Then selecting the **Add Interface Function** should lead to another dialog:

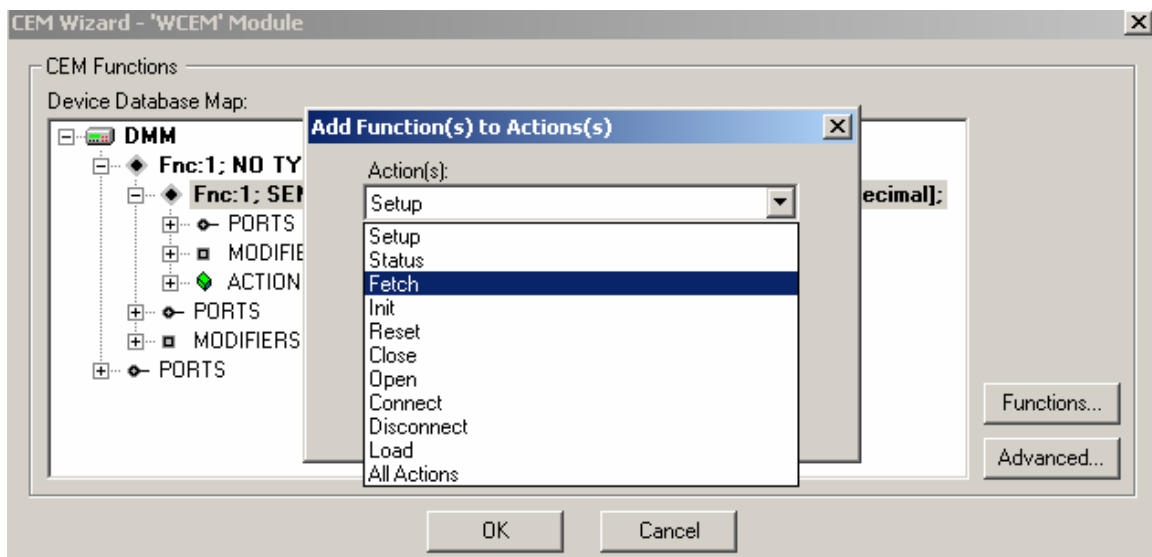
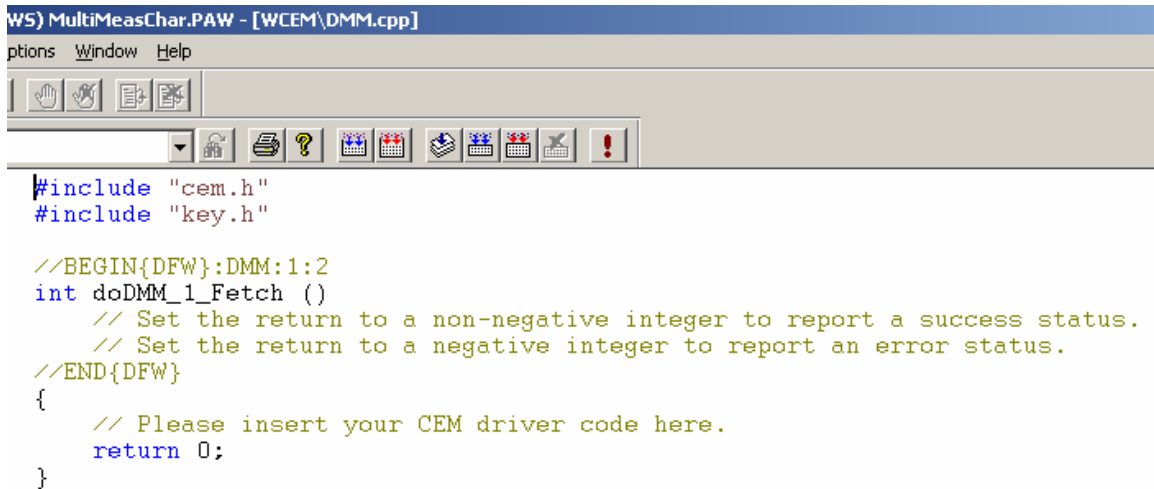


Figure 4.

From the **Add Function(s) to Action(s)** dialog, the **Fetch** function has to be selected. This step will generate the **DMM.cpp** file in the WCEM module.

The file should have the framework for the **Fetch** function as displayed on the following image:



```

W5) MultiMeasChar.PAW - [WCEM\DMM.cpp]
ptions Window Help

#include "cem.h"
#include "key.h"

//BEGIN{DFW}:DMM:1:2
int doDMM_1_Fetch ()
    // Set the return to a non-negative integer to report a success status.
    // Set the return to a negative integer to report an error status.
//END{DFW}
{
    // Please insert your CEM driver code here.
    return 0;
}

```

Figure 5.

In the real project there should be additional Interface functions added to the WCEM code.

For the need of this document there will be only two functions, **Fetch** and **Setup** (adding Setup function can be done in the same way as the Fetch was added).

Once the framework is implemented, the measured values to be retrieved automatically need to be selected.

This has to be done again from the WCEM Wizard level.

By expanding the **Actions and Parameters** entry from the WCEM Wizard, the following list of modifiers should be visible:

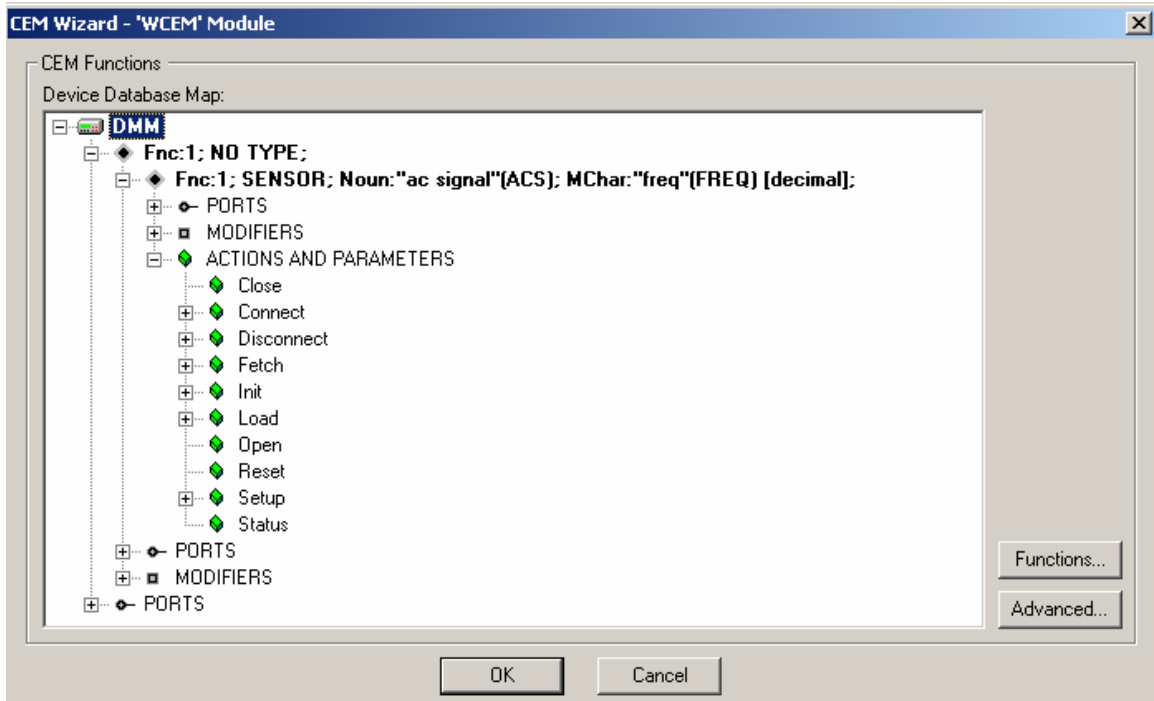


Figure 6.

Now the **Fetch** function has to be expanded:

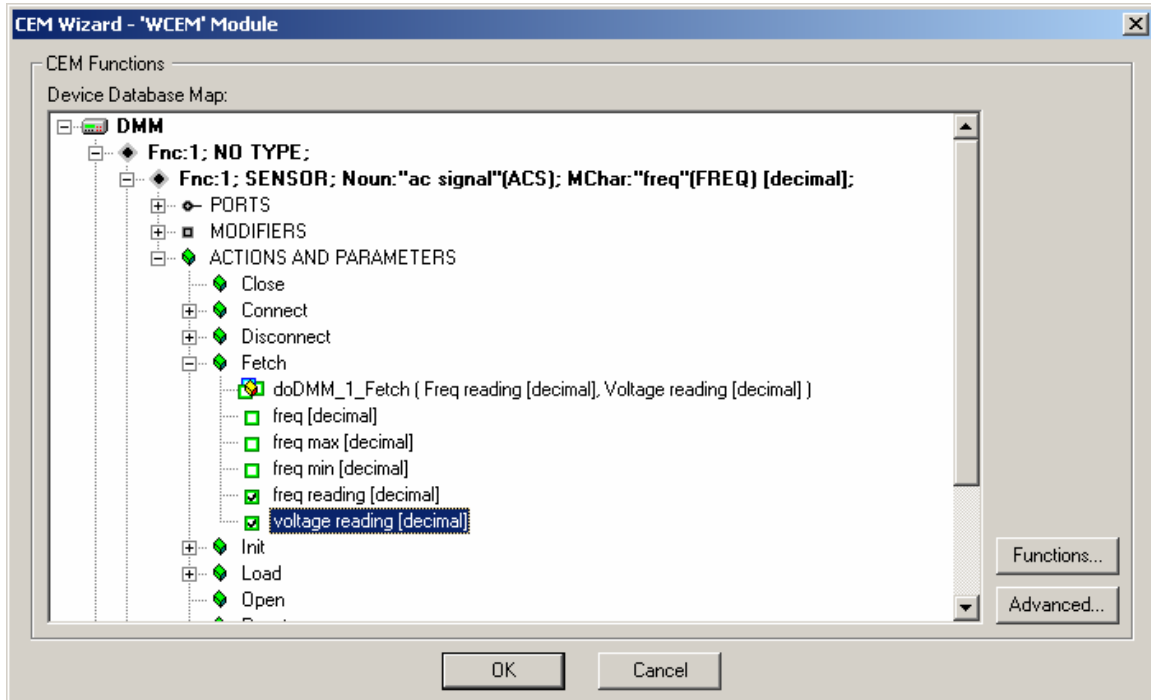


Figure 7.

For the Frequency, the **freq reading [decimal]** (primary measured characteristic) entry has to be selected, and for the voltage, the **voltage reading [decimal]** (auxiliary measured characteristic) needs to be chosen as shown on the figure above:

Those selections should result in generating some additional code inside **DMM.cpp** file as presented below.

```
#include "cem.h"
#include "key.h"

//BEGIN{DFW}:DMM:1:2
int doDMM_1_Fetch (
    // freq reading [decimal]
    double* pFREQ,
    // voltage reading [decimal]
    double* pVOLT)
    // Use this function argument to pass back one or more elements to ATLAS.
    // Set the return to the number of elements passed back in the function argument.
    // Set the return to a negative integer to report an error status.
//END{DFW}
{
    // Please insert your CEM driver code here.
    return 0;
}
```

Figure 8.

Now the **doDMM_1_Fetch** function is taking two arguments, which are pointers **pFREQ** and **pVOLT**.

The simple code, which will pass the **frequency** and **voltage** fixed values measurements back to ATLAS, is presented below:

```
#include "cem.h"
#include "key.h"

//BEGIN{DFW}:DMM:1:2
int doDMM_1_Fetch (
    // freq reading [decimal]
    double* pFREQ,
    // voltage reading [decimal]
    double* pVOLT)
    // Use this function argument to pass back one or more elements to ATLAS.
    // Set the return to the number of elements passed back in the function argument.
    // Set the return to a negative integer to report an error status.
//END{DFW}
{
    // Please insert your CEM driver code here.Display("CEM: Entering doDMM2_2_Fetch\n");
    int nRet = 0;
    if (pFREQ != NULL)
    {
        Display("CEM: FREQ measurement\n");
        *pFREQ = 1.123;
        nRet++;
    }
    if (pVOLT != NULL)
    {
        Display("CEM: VOLT measurement\n");
        *pVOLT = 2.468;
        nRet++;
    }
    return nRet;
}
```

Figure 9.

As seen in the above figure the measurement values are simulated.

The ATLAS variables should return should return **1.123 Hz** for frequency and **2.468 V** for voltage.

4 A complete PAWS Project sample

4.1 The 716.89/PAWS Atlas code:

```

001000 BEGIN, ATLAS PROGRAM 'SAMPLE'                                $
  10 DECLARE, VARIABLE, 'A', 'B' IS DECIMAL                        $
E010000 SETUP, (FREQ, VOLTAGE), AC SIGNAL,
  VOLTAGE RANGE 0 V TO 200 V,
  FREQ RANGE 10 HZ TO 500 HZ,
  CNX HI J2-1 LO J2-2                                            $
  20 CONNECT, (FREQ, VOLTAGE), AC SIGNAL,
  VOLTAGE RANGE 0 V TO 200 V,
  FREQ RANGE 10 HZ TO 500 HZ,
  CNX HI J2-1 LO J2-2                                            $
  30 INITIATE, (FREQ, VOLTAGE), AC SIGNAL,
  VOLTAGE RANGE 0 V TO 200 V,
  FREQ RANGE 10 HZ TO 500 HZ,
  MAX-TIME 1.0 SEC,
  CNX HI J2-1 LO J2-2                                            $
C   FETCH is using two measurement characteristics.                $
  40 FETCH, (FREQ INTO 'A', VOLTAGE INTO 'B'), AC SIGNAL,
  VOLTAGE RANGE 0 V TO 200 V,
  FREQ RANGE 10 HZ TO 500 HZ,
  MAX-TIME 1.0 SEC,
  CNX HI J2-1 LO J2-2                                            $
  OUTPUT, C'ATLAS: FREQ = ', 'A', C' VOLT = ', 'B'                $
  70 DISCONNECT, (FREQ, VOLTAGE), AC SIGNAL,
  VOLTAGE RANGE 0 V TO 200 V,
  FREQ RANGE 10 HZ TO 500 HZ,
  CNX HI J2-1 LO J2-2                                            $
  80 RESET, (FREQ, VOLTAGE), AC SIGNAL,
  VOLTAGE RANGE 0 V TO 200 V,
  FREQ RANGE 10 HZ TO 500 HZ,
  CNX HI J2-1 LO J2-2                                            $
040000 TERMINATE, ATLAS PROGRAM 'SAMPLE'                          $

```

Figure 10.

4.2 The Device Database (*.ddb) file:

```
begin DEV DMM2 using DMM2;
  cnx hi DMM2-HI, lo DMM2-LO;

  begin FNC = 1;

  control
  {
    freq range 10 hz to 3e3 hz;
  }
  capability
  {
    (voltage) range 0 v to 300 v; * Auxiliary Measured Characteristic
  }

  begin;
    sensor (freq) ac signal; * Primary Measured Characteristic
  end;

  end; *End FNC1

end; *End begin DEV
```

Figure 11.

4.3 The WCEM code (*.cpp) code:

```
#include "cem.h"
#include "key.h"

//BEGIN{DFW}:DMM2:1:2
int doDMM2_1_Fetch (
    // freq reading [decimal]
    double* pFREQ,
    // voltage reading [decimal]
    double* pVOLT)
    // Use this function argument to pass back one or more elements to ATLAS.
    // Set the return to the number of elements passed back in the function argument.
    // Set the return to a negative integer to report an error status.
//END{DFW}
{
    // Please insert your CEM driver code here.
    Display("CEM: Entering doDMM2_2_Fetch\n");
    int nRet = 0;
    if (pFREQ != NULL)
    {
        Display("CEM: FREQ measurement\n");
        *pFREQ = 1.123;
        nRet++;
    }
    if (pVOLT != NULL)
    {
        Display("CEM: VOLT measurement\n");
        *pVOLT = 2.468;
        nRet++;
    }
    return nRet;
}
```

Figure 12.

4.4 The log file:

Running the TPS above will yield to the following log file:

```

INF: Binary files "atlas", version 20080325<LF>Date stamp: Fri Sep 19 16:46:59 2008
INF: Built In LEX Information
INF: CEM
"\\Ntserver\Office\Prob_Rep\2008\pr08034\Docs\TYX_Bartek15Aug08\PawsProject\WCEM.DLL',
enhanced error reporting
BUS: OPEN gpib0 -- Ok
INF: CEM Module User / Kernel Model 2 Version 20060925 (3.9.33)
BUS: OPEN Channel -- Ok
STM: 16:49:13 010000 6151 SET (FREQ) LACS
BUS: WRITE TO = 10 DMM "FNC ACS FREQ :CH1 SRN VOLT !R:1 +.000000000000000E+00\
      SRX VOLT !R:1 +.200000000000000E+03 SRN FREQ !R:1 +.100000000000000E+02\
      SRX FREQ !R:1 +.500000000000000E+03<0x0D><0x0A>"
BUS: WRITE TO = 10 DMM "STA<0x0D><0x0A>"
BUS: READ TO = 10 DMM " <0x0D><0x0A>"
STM: 16:49:13 010020 6306 CON (FREQ) LACS
STM: 16:49:13 010030 6362 INX (FREQ) LACS
BUS: WRITE TO = 10 DMM " SET MAXT !R:1 +.100000000000000E+01<0x0D><0x0A>"
BUS: WRITE TO = 10 DMM "INX FREQ<0x0D><0x0A>"
BUS: READ TO = 10 DMM " <0x0D><0x0A>"
STM: 16:49:13 010040 6454 FTH (FREQ) LACS
BUS: WRITE TO = 10 DMM " SET MAXT !R:1 +.100000000000000E+01<0x0D><0x0A>"
DSP: CEM: FREQ measurement
BUS: WRITE TO = 10 DMM "FTH FREQ !R:1<0x0D><0x0A>"
BUS: READ TO = 10 DMM " +.1123E+1<0x0D><0x0A>"
DSP: CEM: VOLT measurement
BUS: WRITE TO = 10 DMM "FTH VOLT !R:1<0x0D><0x0A>"
BUS: READ TO = 10 DMM " +.2468E+1<0x0D><0x0A>"
STM: 16:49:13 010040 6575 OUT
OUT: ATLAS: FREQ = .112300000000000E+01 VOLT = .246800000000000E+01
STM: 16:49:13 010070 6731 DIS (FREQ) LACS
STM: 16:49:13 010080 7011 RST (FREQ) LACS
BUS: WRITE TO = 10 DMM "RST ACS FREQ :CH1<0x0D><0x0A>"
STM: 16:49:13 040000 7105 TRM
BUS: RESET gpib0 -- Ok
BUS: RESET Channel -- Ok
BUS: CLOSE gpib0 -- Ok
BUS: CLOSE Channel -- Ok

```

As we can see, 2 values have been returned to the ATLAS, one for the voltage value and one for the frequency value.