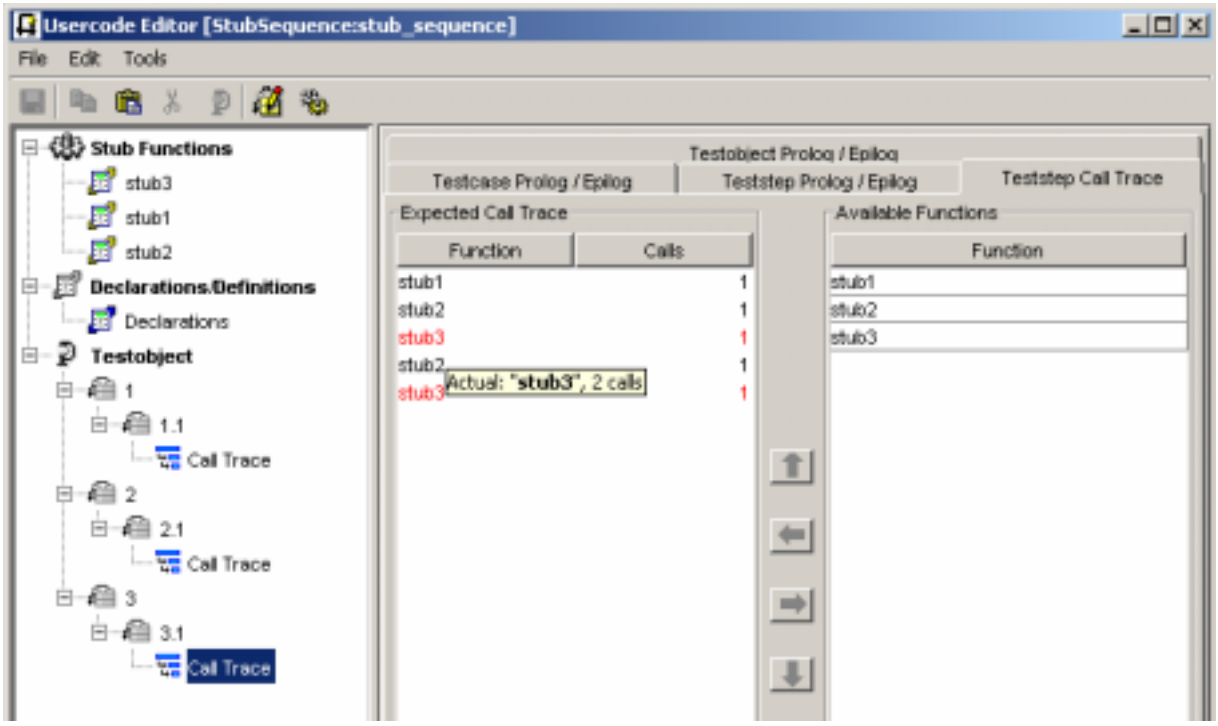
Tessy V2.3 Features

Checking The Order Of Stub Function Calls

Tessy can now check the order of stub function calls. This is useful, if a test object calls stub functions, and the order of the calls (and their number) shall be checked for each test case.



The test object calls three stub functions, but in the third test case, the number of calls to stub3 was obviously not as expected

Stubs For Functions In The Same Module

During unit testing, functions called by the test object shall be replaced by stub functions. In case the called function is implemented in the same C source module as the test object, normally the linker cannot be prevented from linking the called function in the same (unchanged) C source module. However, Tessy V2.3 allows now the use of stub functions in this situation.

Evaluating Variables Not In The Interface

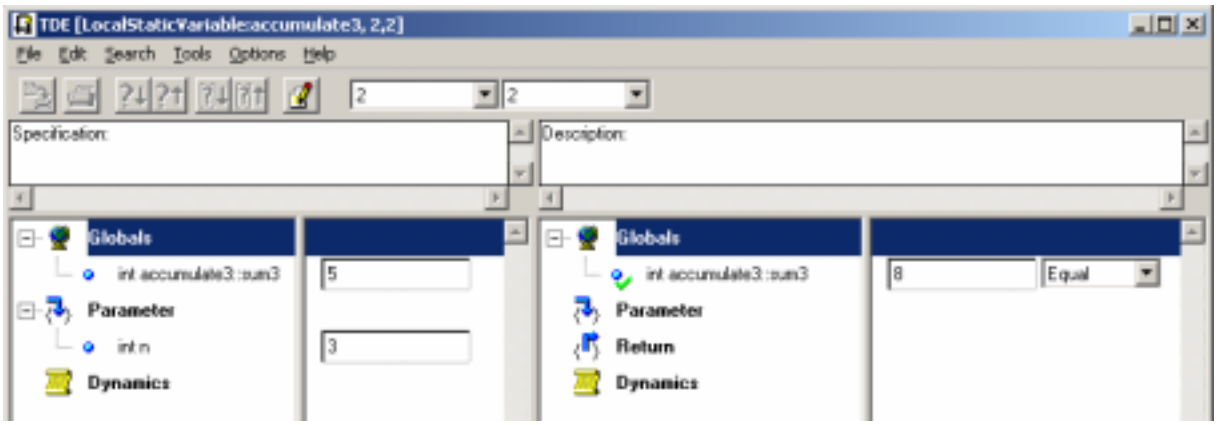
Tessy V2.3 introduces the possibility to include in the test results the evaluation of variables, that are not used in the test object and therefore do not show up in the interface of the test object. The evaluation can take place in the user code of stub functions or in the test step epilog of a test case. This allows e.g. to check for unwanted side effects of the test case or for the result of actions initiated by the test, e.g. in the Special Functions Register of a microcontroller.

Observing static variables local to a function

When implementing state machines, often static variables local to a function are used to maintain the internal state between subsequent calls to the C function representing the state machine. However, such a variable is per definition not visible outside the function, and therefore, up to now, the value of that variable (i.e. the state of the machine) could not be observed during the test. (Global state variables could be observed since ever.) With Tessy V2.3, this restriction is circumvented, and static variables local to a function can serve as test input and/ or output, i.e. they can be set prior to test execution and checked for the correct result after test execution.

```
void accumulate3(int n)
{
      static int sum3;
      sum3 += n;
}
```

In the application, the static variable sum3 is not visible outside the function accumulate()



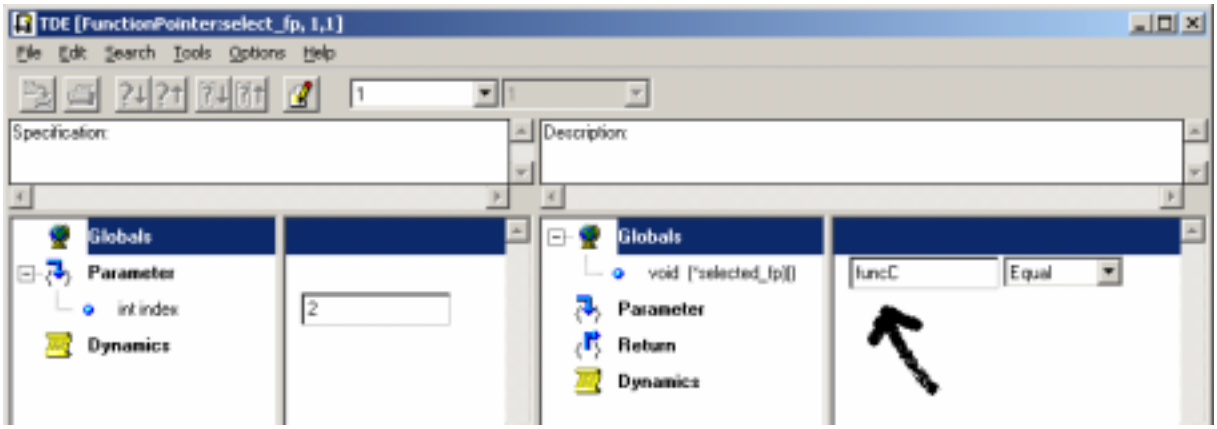
The static variable sum3 can be set prior to the test and checked afterwards

Checking Function Pointers

If the result of your function is a pointer to a function, the numeric result (i.e. the address of the function) may differ depending on the linking. Tessy is able to use the name of the function as expected result and will check automatically, if the resulting address corresponds to the correct function name or not.

```
void funcA (void) {}
void funcB (void) {}
void funcC (void) {}

void (*selected_fp)();

void (*fp[3])(void) = {funcA, funcB, funcC};

void select_fp (int index)
{
      switch (index)
      {
      case  0: selected_fp = fp[0];break;
      case  1: selected_fp = fp[1];break;
      default: selected_fp = fp[2];break;
      };

      return;
}
```

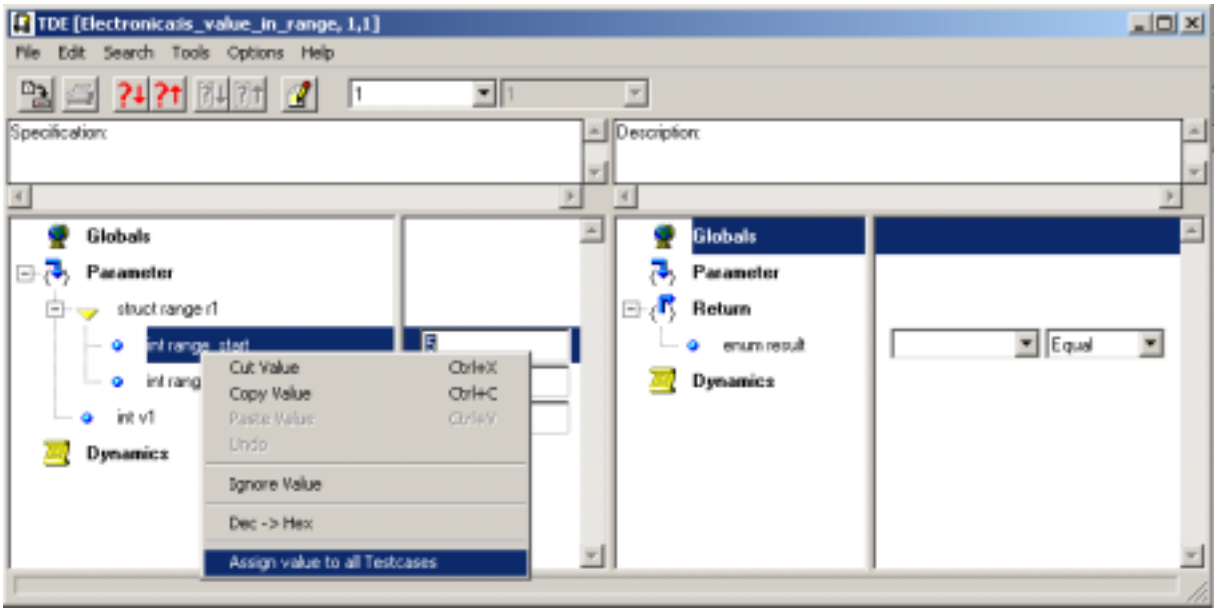
After the test, the pointer selected_fp points to one of the three functions funcA, funcB, funcC



Tessy can check if the correct function name is selected

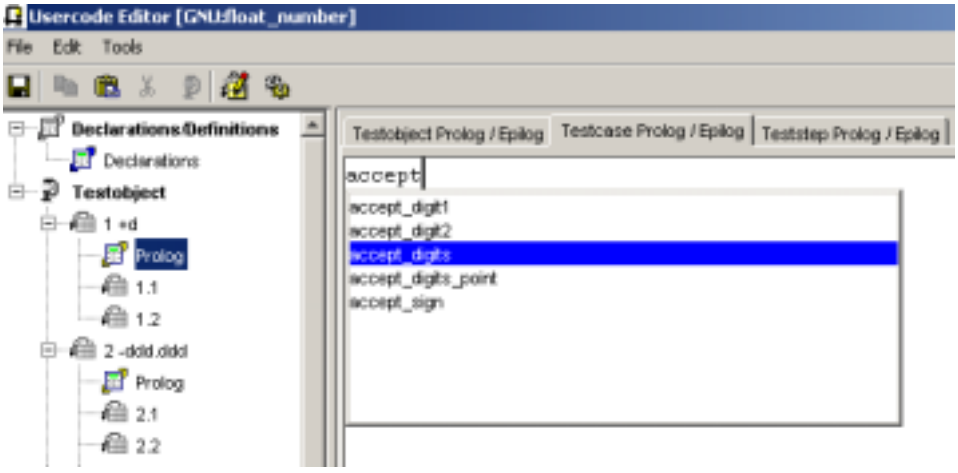
Assigning A Value To A Variable In All Test Cases

The Test Data Editor (TDE) allows now very conveniently to set a variable to a certain value for all existing test cases at the same time.



The context menu allows the setting of a variable for all test cases

Active Usercode



Auto completion in the usercode

When pressing Ctrl+Space in the Usercode Editor, an auto completion menu pops up, which allows convenient usage of variables from the interface of the test object.

Furthermore, if the name of the variable in the interface should change (because the source code of the test object was changed), the name of the variable in the usercode is automatically adjusted by Tessy.

Additional Microcontroller Architectures and Compilers Supported

Tessy V2.3 supports additionally to Tessy V2.2.9 the following microcontroller architectures	
Manufacturer	**Architecture**
Atmel	AVR
Infineon	XC166
Mitsubishi	M32C
Freescale (Motorola)	HCS12, HCS08
Texas Instruments	TMS320, TMS470

New compilers supported are GNU for ARM, Hightec for TriCore, and Texas Instruments (CodeComposerStudio) for TMSxxx.

Tessy supports debuggers from various manufacturers.