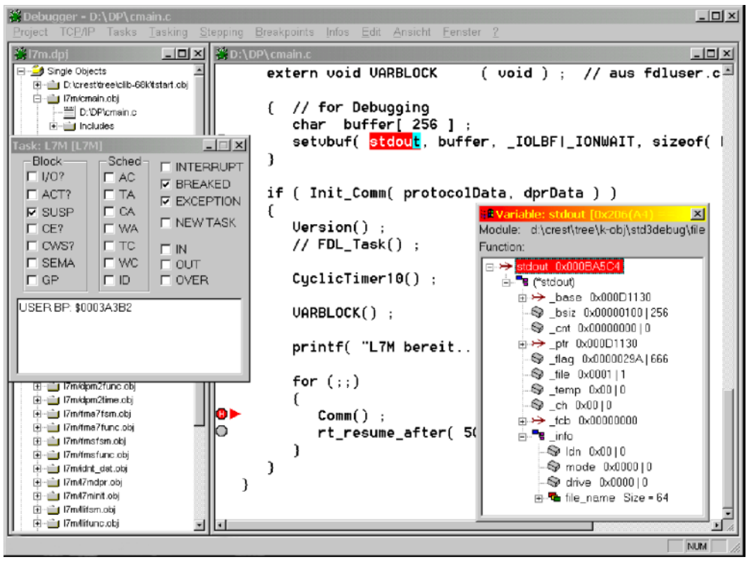
Ingenieurbüro für Echtzeitprogrammierung

RT-Debug

Source level debugging for ANSI-C and PEARL



Block	Sched	
☐ I/O?	☐ AC	☐ INTERRUPT
☐ ACT?	☐ TA	☑ BREAKED
☑ SUSP	☐ CA	☑ EXCEPTION
☐ CE?	☐ WA	☐ NEW TASK
☐ CWS?	☐ TC	☐ IN
☐ SEMA	☐ WC	☐ OUT
☐ GP	☐ ID	☐ OVER

USER BP: $0003A3B2

D:\DP\cmain.c

```
extern void VARBLOCK     ( void ) ;  // aus fdluser.c

{  // for Debugging
   char  buffer[ 256 ] ;
   setvbuf( stdout, buffer, _IOLBF|_IONWAIT, sizeof( 
}

if ( Init_Comm( protocolData, dprData ) )
{
   Version() ;
   // FDL_Task() ;

   CyclicTimer10() ;

   VARBLOCK() ;

   printf( "L7M bereit..

   for (;;)
   {
      Comm() ;
      rt_resume_after( 5(
   }
}
   }
```

Variable: stdout [0x206(A4)]
Module: d:\crest\tree\k-obj\std3debug\file
Function:

```
stdout  0x000BA5C4
  (*stdout)
    _base  0x000D1130
    _bsiz  0x00000100 | 256
    _cnt  0x00000000 | 0
    _ptr  0x000D1130
    _flag  0x0000029A | 666
    _file  0x0001 | 1
    _temp  0x00 | 0
    _ch  0x00 | 0
    _fcb  0x00000000
    _info
      ldn  0x00 | 0
      mode  0x0000 | 0
      drive  0x0000 | 0
      file_name  Size = 64
```

Remote Debugging

Remote debugging gives the full comfort of a graphical environment during the program development targeting small systems. The separation between an efficient user interface and a small debugger kernel assures the almost undisturbed program behavior on the target system.

With a connection to the target system through standard networks, even remote debugging over the internet is possible.

Source level

Project-Type

[✓ CREST-C] PEARL 90

The analysis of program behavior is constantly done on source level. Whether a program is written using Crest-C or UH-PEARL, **RT-Debug** has access to all program objects and considers the characteristics of the individual languages.

The program flow is shown in the source code. Access to variables is strongly typed, also for user-defined data types. **RT-Debug** is aware of the multitasking environment and hands full control of the programs execution under the realtime operating system RTOS-UH to the user.

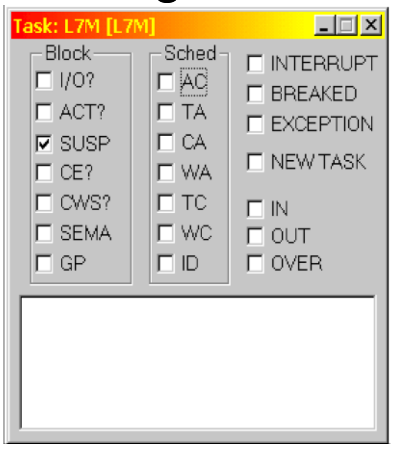
Break- and watchpoints provide for detailled examination of a programs state and flow with minimal disruptions.

Crash analysis

Even in the case of catastrophic program aborts such as bus- or address errors **RT-Debug** offers support.

Callstack and backtrace allow to inspect the program behavior before a crash. At each point in the callstack, the program state is displayed in the correct context. Seeing the valid values of variables eases the detection of either algorithmic or tasking-based programming errors as far as possible. For special cases, access to assembler code, register contents and administrative task data is provided.

Tasking control



```
Task: L7M [L7M]
Block        Sched        
[ ] I/O?     [ ] AC      [ ] INTERRUPT
[ ] ACT?     [ ] TA      [ ] BREAKED
[✓] SUSP     [ ] CA      [ ] EXCEPTION
[ ] CE?      [ ] WA      [ ] NEWTASK
[ ] CWS?     [ ] TC      [ ] IN
[ ] SEMA     [ ] WC      [ ] OUT
[ ] GP       [ ] ID      [ ] OVER
```

RT-Debug shows the current state of the task under control in a task state window.

The continuous display of the task state provides a precise insight into the runtime behavior. Special events are logged in a message pane.

A task can be interrupted at any time. The current program position is graphically shown in the source code window.

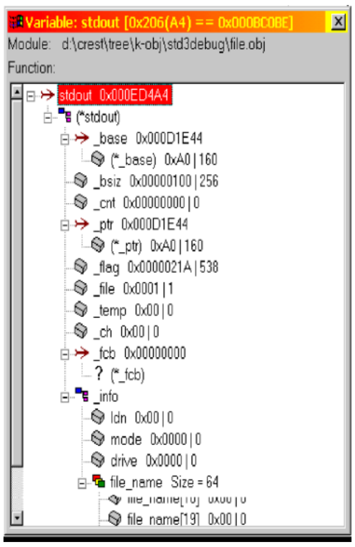
For exact control of the program flow, controlled program execution is provided by the instructions:

- Step in – executes the program in a single step mode
- Step Out – stops at the return from the current procedure
- Step Over – stops after returning from a procedure call

Breakpoints for systematic interruption of program execution are set directly in the source code window.

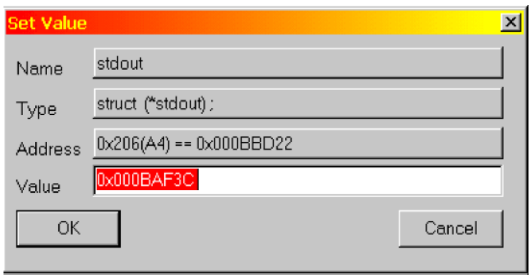
Watchpoints allow to take a snapshot of all variables currently in scope with minimal disruption of program flow.

Quick Watch



```
Variable: stdout [0x206(A4) == 0x000BC0BE]
Module:   d:\crest\tree\k-obj\std3debug\file.obj
Function:
stdout  0x000ED4A4
  (*stdout)
    _base  0x000D1E44
      (*_base)  0xA0 | 160
    _bsiz  0x00000100 | 256
    _cnt  0x00000000 | 0
    _ptr  0x000D1E44
      (*_ptr)  0xA0 | 160
    _flag  0x0000021A | 538
    _file  0x0001 | 1
    _temp  0x00 | 0
    _ch  0x00 | 0
    _fcb  0x00000000
      ? (*_fcb)
    _info
      ldn  0x00 | 0
      mode  0x0000 | 0
      drive  0x0000 | 0
      file_name  Size = 64
        file_name[10]  0x00 | 0
        file_name[19]  0x00 | 0
```

The values of all local and global variables can be inspected using the quickwatch window. Variables of complex data types are shown in a tree view, selective opening of sub-elements gives a quick overview and simple access to member variables. Different symbols for different data types give a concise view, even when working on larger projects.

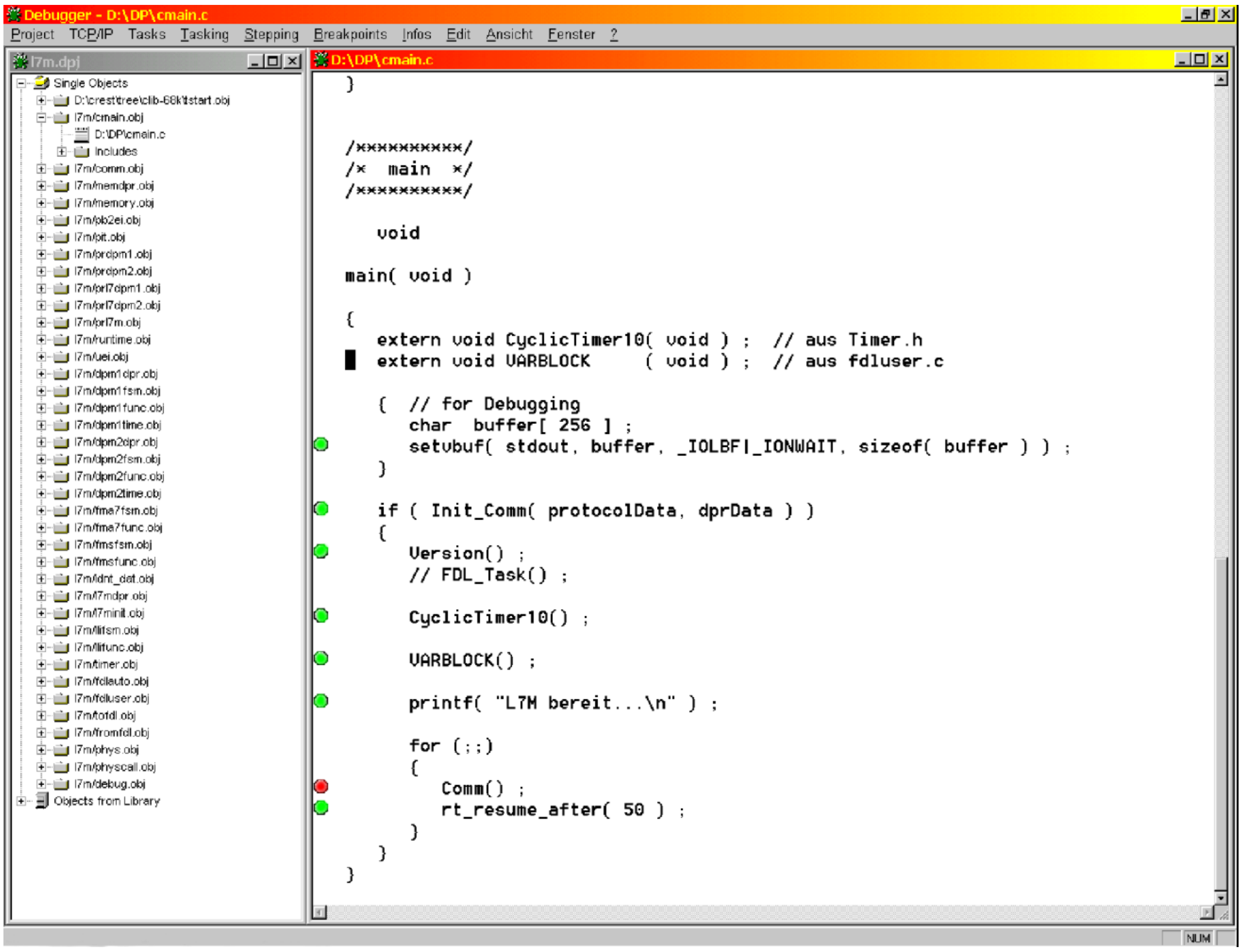


```
Set Value
Name      stdout
Type      struct (*stdout) ;
Address   0x206(A4) == 0x000BBD22
Value     0x000BAF3C
OK                       Cancel
```

A Quick Watch dialog shows all available information about individual variables or members of structured data and allows to change their values.

Project Organization

RT-Debug presents all available project information in clear, hierarchical form. Dependencies between the individual translation units as well as the linked libraries are visualized in order to ease the navigation in the source code.



Debugger - D:\DP\cmain.c
Project TCP/IP Tasks Tasking Stepping Breakpoints Infos Edit Ansicht Fenster ?
l7m.dpj
D:\DP\cmain.c
NUM

```
Single Objects
    D:\crest\tree\clib-68k\tstart.obj
    l7m/cmain.obj
        D:\DP\cmain.c
        Includes
    l7m/comm.obj
    l7m/memdpr.obj
    l7m/memory.obj
    l7m/pb2ei.obj
    l7m/pit.obj
    l7m/prdpm1.obj
    l7m/prdpm2.obj
    l7m/prl7dpm1.obj
    l7m/prl7dpm2.obj
    l7m/prl7m.obj
    l7m/runtime.obj
    l7m/uei.obj
    l7m/dpm1dpr.obj
    l7m/dpm1fsm.obj
    l7m/dpm1func.obj
    l7m/dpm1time.obj
    l7m/dpm2dpr.obj
    l7m/dpm2fsm.obj
    l7m/dpm2func.obj
    l7m/dpm2time.obj
    l7m/fma7fsm.obj
    l7m/fma7func.obj
    l7m/fmsfsm.obj
    l7m/fmsfunc.obj
    l7m/idnt_dat.obj
    l7m/l7mdpr.obj
    l7m/l7minit.obj
    l7m/llifsm.obj
    l7m/llifunc.obj
    l7m/timer.obj
    l7m/fdlauto.obj
    l7m/fdluser.obj
    l7m/tofdl.obj
    l7m/fromfdl.obj
    l7m/phys.obj
    l7m/physcall.obj
    l7m/debug.obj
Objects from Library
```

```
}


/**********/
/*  main  */
/**********/

    void

main( void )

{
    extern void CyclicTimer10( void ) ;  // aus Timer.h
    extern void VARBLOCK     ( void ) ;  // aus fdluser.c

    {  // for Debugging
       char  buffer[ 256 ] ;
       setvbuf( stdout, buffer, _IOLBF|_IONWAIT, sizeof( buffer ) ) ;
    }

    if ( Init_Comm( protocolData, dprData ) )
    {
       Version() ;
       // FDL_Task() ;

       CyclicTimer10() ;

       VARBLOCK() ;

       printf( "L7M bereit...\n" ) ;

       for (;;)
       {
          Comm() ;
          rt_resume_after( 50 ) ;
       }
    }
}
```

The individual source files are accessable simply and quickly by navigation in the project tree. The simultaneous display of several source code panes gives an optimal view to the program flow.

Break- and Watchpoints are shown in the source code, different colors allow to differentiate between possible, active and hitten points.

When a breakpoint is hit, the program execution is suspended, all quickwatch windows are refreshed and the current point of execution is shown graphically in the source code.

On the hit of a watchpoint, program execution is interrupted only to refresh the quickwatch windows and is resumed immediately thereafter.

Target systems

RT-Debug is available for all systems based on RTOS-UH using either PowerPC or the 68xxx-family. A small debugger kernel in the target system communicates with the comfortable user interface using the TCP/IP protocol. The target can be connected either serial or by network.

The debugger kernel provides basic debugging functions. Apart from the manipulation of storage areas he sets or resets breakpoints, observes task condition changes and recognizes special events during task execution. All actions are initiated by the development computer.

The separation of the debugger into a kernel with elementary basic functions and a comfortable user interface on a commonly used workstation leads to a very small load of the target system. The target system does not have to fulfill special requirements regarding available memory or computational power. Even programs targeting small systems with little or no disks can be debugged comfortably without special hardware support. No external debugging tools are needed.

Development system

The development computer presents the main functionality of the debugger. It translates the source files, analyzes and interprets the debug informations of the compilers and gives a concise view of the program flow. The user interface follows the Look and Feel of the operating system.

All versions of the Microsoft Windows desktop or server operating system since Windows 95 are supported.

Coupling the target systems to the development system by network allows to separate the location of target from the workstation. Debug sessions can be made even over the Internet.