Trace Function Enter, Exit and Leave

Many times during debugging it is useful to know what functions are called, and in what order. However, adding **RDebug::Print**s in the beginning and end of interesting functions is tedious and time consuming. Also, early return statements in the function cause the **Exit** print to be skipped.

16 Nov 2008

Automatic variables (stack based objects) are destroyed when the function goes out of scope. Placing a logging object in the stack is an easy way to get **Enter** and **Exit** prints without missing the **Exit** print when the function returns early.

The function can also exit with a **Leave** and it is useful to get a log print from that too. In S60 3rd Edition and newer releases where leaves are implemented as exceptions, this is very simple. C++ provides a function **std::uncaught_exception()** to check if an exception is currently active.

Function return value can also be logged, because in a 32-bit system it is always a 32-bit value. All it takes is some inline assembly code. The downside is that the assembly code is not portable across different build targets (for example winscw or ARM), so different implementations need to be defined.

Function name can be retrieved automatically using the **__PRETTY_FUNCTION__** compiler macro. The compiler provides classname, function name and its signature in this macro and with a little bit of descriptor juggling it can be printed to the log.

Simple code example

This is a simple tracer that writes the function **Enter**, **Exit** and **Leave** to RDebug or a file. It can also log the function return value. It defines a simple T class that can be used as an automatic variable in a function. In its constructor it logs the function **Enter** and in the destructor it logs either the normal **Exit** or a **Leave**. So far it can only log the function return value for a **winscw** target.

File: tracer.h

```
#ifndef TTRACER_H
#define TTRACER_H

#include <e32base.h>

// Define tracer logging method
// 0    = Logging off
// 1    = Log to RDebug
// 2    = Log to file (RFileLogger)
#define TRACER_LOG_METHOD 2

// ==============================================================================

// Logging off, define empty macros and skip all the rest
#if TRACER_LOG_METHOD == 0

    #define TRACER(func)
    #define TRACER_RET(func,format)

#else                       // Logging on

    // Macro to print function entry, exit and leave.
    // Example: TRACER("CMyClass::MyFunction");
    #define TRACER(func) TTracer function_tracer( _S(func), _S("") );

    // Macro to print function return value in addition to entry, exit
    // and leave conditions Second parameter is a formatting string used
    // to print the return value Example to print an integer return value:
    // TRACER_RET("CMyclass::MyFunction", "%d");
    #define TRACER_RET(func,format) TTracer func_tracer( _S(func), _S(format) );

    // Macro to print function entry, exit and leave.
    // Gets the function name automatically
    // Example: TRACER_AUTO;
    // Substract 1 from MaxLength() because PtrZ() adds the zero terminator
    #define TRACER_AUTO \
        TPtrC8 __ptr8((const TUint8*)__PRETTY_FUNCTION__); \
        TBuf<150> __buf; \
        __buf.Copy( __ptr8.Left( __buf.MaxLength() - 1 ) ); \
        TTracer function_tracer( __buf.PtrZ(), _S("") )

    // Macro to print function entry, exit and leave.
    // Gets the function name automatically
    // Example: TRACER_AUTO_RET("%d");
    // Substract 1 from MaxLength() because PtrZ() adds the zero terminator
    #define TRACER_AUTO_RET(format) \
        TPtrC8 __ptr8((const TUint8*)__PRETTY_FUNCTION__); \
        TBuf<150> __buf; \
        __buf.Copy( __ptr8.Left( __buf.MaxLength() - 1 ) ); \
        TTracer function_tracer( __buf.PtrZ(), _S(format) )

    #if TRACER_LOG_METHOD == 1      // Print to RDebug

        #include <e32debug.h>
        #define TRACER_PRINT(a)         RDebug::Print(a,&iFunc);
        #define TRACER_PRINT_RET(a,b)   RDebug::Print(a,&iFunc,b);

    #elif TRACER_LOG_METHOD == 2    // Print to file

        #include <flogger.h>
        _LIT( KLogDir,  "tracer" );     // Log directory: C:\logs\tracer
        _LIT( KLogFile, "tracer.txt" ); // Log file: c:\logs\tracer\tracer.txt
        #define TRACER_PRINT(a)         RFileLogger::WriteFormat(KLogDir, \
                                KLogFile,EFileLoggingModeAppend,a,&iFunc);
        #define TRACER_PRINT_RET(a,b)   RFileLogger::WriteFormat(KLogDir, \
                                KLogFile,EFileLoggingModeAppend,a,&iFunc,b);

    #endif
```


(C) Copyright Nokia 2013. All rights reserved.

```
    _LIT( KLogEnter,    "%S: ENTER" );
    _LIT( KLogExit,     "%S: EXIT" );
    _LIT( KLogLeave,    "%S: LEAVE!" );
    _LIT( KLogExitRet,  "%S: EXIT, Returning " );

    /**
     * Simple tracer class that logs function enter, exit, or leave
     */
    class TTracer
        {
    public:

        /**
         * inline constructor to write log of entering a function
         */
        TTracer( const TText* aFunc, const TText* aRetFormat )
            : iFunc( aFunc )
            , iRetFormat( aRetFormat )
            {
            TRACER_PRINT( KLogEnter );
            }

        /**
         * inline destructor to write log of exiting a function
         * normally or with a leave
         */
        ~TTracer()
            {
            if ( std::uncaught_exception() ) // Leave is an exception
                {
                // The function exited with a leave
                TRACER_PRINT( KLogLeave );
                }
            else
                {
                // The function exited normally
                if ( iRetFormat.Length() == 0 )
                    {
                    TRACER_PRINT( KLogExit );
                    }
                else
                    {
                    // Log the return value
                    #ifdef __WINS__
                        TInt32 retVal = 0;

                        // The assembly bit. This needs to be reimplemented
                        // for every target.
                        _asm( mov retVal, ebx );

                        TBuf<100> format( KLogExitRet );
                        format.Append( iRetFormat );
                        TRACER_PRINT_RET( format, retVal );
                    #else
                        TRACER_PRINT( KLogExit );
                    #endif
                    }
                }
            }

    private:

        /**
         * Pointer descriptor to function signature that is to be logged.
         */
        TPtrC iFunc;

        /**
         * Formatting string used to print the function return value
         */
        TPtrC iRetFormat;

        };

#endif // TRACER_LOG_METHOD == 0

#endif // TTRACER_H
```

Usage

```
#include "tracer.h"

// Log Enter, Exit or Leave
void CSomeClass::SomeFunctionL()
    {
    TRACER( "CSomeClass::SomeFunctionL" );
    ...
    }
// Prints:
// CSomeClass::SomeFunctionL: ENTER
// CSomeClass::SomeFunctionL: EXIT
// or
// CSomeClass::SomeFunctionL: LEAVE

// Log Enter, Exit or Leave and the integer return value
TInt CSomeClass::ReturnSomeInt()
    {
    TRACER_RET( "CSomeClass::ReturnSomeInt", "%d" );
    ...
    return 42;
    }
// Prints:
// CSomeClass::ReturnSomeInt: ENTER
// CSomeClass::ReturnSomeInt: EXIT, Returning 42
// or
// CSomeClass::ReturnSomeInt: LEAVE

// Log Enter, Exit or Leave and the descriptor return value
HBufC* CSomeClass::ReturnSomeDescL()
    {
    TRACER_RET( "CSomeClass::ReturnSomeDescL", "%S" );
    return _L("Test data").AllocL();
    }
// Prints:
// CSomeClass::ReturnSomeDescL: ENTER
```


(C) Copyright Nokia 2013. All rights reserved.

```
// CSomeClass::ReturnSomeDescL: EXIT, Returning Test data
// or
// CSomeClass::ReturnSomeDescL: LEAVE

// Log Enter, Exit or Leave without specifying function name
void CSomeClass::SomeFunctionL( TBool aFlag )
    {
    TRACER_AUTO;
    ...
    }
// Prints:
// CSomeClass::SomeFunctionL(int): ENTER
// CSomeClass::SomeFunctionL(int): EXIT
// or
// CSomeClass::SomeFunctionL(int): LEAVE

// Log Enter, Exit or Leave and the integer return value without specifying function name
TInt CSomeClass::ReturnSomeInt( TInt aArg )
    {
    TRACER_AUTO_RET( "%d" );
    ...
    return 42;
    }
// Prints:
// CSomeClass::ReturnSomeInt(int): ENTER
// CSomeClass::ReturnSomeInt(int): EXIT, Returning 42
// or
// CSomeClass::ReturnSomeInt(int): LEAVE
```

Helper script

This is a simple perl script that reads through a CPP file and adds TRACER macros in the beginning of each function. It keeps the original file as a backup under a different name. The script will not add TRACER_RET macros into functions that return values – that needs to be done manually.

add_traces.pl

```
# add_traces.pl (C) Marko Kivijärvi 2006
# Dummy checks
die "Specify an input file!\n" if $ARGV[0] eq "";
die "File not found!\n" unless -e $ARGV[0];
die "Incorrect file extension for a C/C++ file!\n"
                if ( $ARGV[0] !~ /(.*)\.(c|cpp)$/ );

# Constants
my $INC_TRACER_H = "#include \"tracer.h\"\n";
my $TRACER = "TRACER";

# Parse output filename from the input filename
my $file = $ARGV[0];
my $origFile = $1."-orig.".$2;
system( "copy $file, $origFile" );

# Reset the input record separator (newline) so the entire file is read at once
undef $/;

# Read the input file
$_ = <>; # All there

# Adds a tracer macro after each function definition
s/
    (\b\w*?\b[&*]?)?            # Possible function return type
    \s+                         # One or more empty spaces
    (\b\w+?\b)                  # Class name
    \s*?                        # Possible empty space
    ::                          # ::
    \s*?                        # Possible empty space
    (~?\b\w+?\b)                # Function name
    \s*?                        # Possible empty space
    \(                          # Opening brace
    ([^)]*?)                    # Possible function parameters
    \)                          # Closing brace
    \s*?                        # Possible empty space
    (const)?                    # Possible 'const'
    [^{;\/]*?                   # Possible empty space or constructor
                                # initializer list
    \{                          # Opening curly brace
/
    Parse($&,$1,$2,$3,$4,$5)    # Print the match and add the macro
/gxe; # g = repeat, x = split regex to multiple lines, e = evaluate substitution

open OUT, ">$file" or die "Cannot open file $file $!\n";
print OUT $INC_TRACER_H;
print OUT;
close OUT;

exit 0;

sub Parse {
    my $match   = shift;
    my $ret     = shift;
    my $class   = shift;
    my $func    = shift;
    my $param   = shift;
    my $const   = shift;

    foreach ( $ret, $class, $func, $param ) {
        s/^\s+|\s+$//g;
        s/\n//g;
    }

    my $debug = $match."\n    ";
    $debug .= $TRACER."(\"";
    $debug .= $ret." " if defined $ret;
    $debug .= $class."::".$func."(";
    $debug .= $param if $param ne "";
    $debug .= ")";
    $debug .= " ".$const if defined $const;
    $debug .= "\")";

    return $debug;
}
```


(C) Copyright Nokia 2013. All rights reserved.

See Using TRAP for additional information about logging leaves..

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.