RVDS 4.0 Introductory Tutorial

Introduction

Aim

This tutorial provides you with a basic introduction to the tools provided with the RealView Development Suite version 4.0 (RVDS). This will include the use of command line and GUI tools, to build and debug projects.

The tutorial is split into two practical sessions:

Session 1 –	Command line tools and executing images in RealView Debugger.
Session 2 –	Creating projects using the ARM Workbench IDE and debugging using RealView Debugger.

Pre-requisites

This tutorial is intended for use with a Microsoft Windows version of RVDS v4.0. You should be familiar with Microsoft DOS/Windows, and have a basic knowledge of the C programming language.

Note: Explanation of File Extensions:

`.c`	C source file
`.h`	C header file
`.o`	object file
`.s`	assembly language source file
`.axf`	ARM Executable file, as produced by **`armlink`**
`.txt`	ASCII text file

Additional information

This tutorial is not designed to provide detailed documentation of RVDS. Full documentation is provided with the product.

Further help can be accessed by pressing *F1* when running RVD, from the help menu, or by using the --help switch for a command line tool. The documentation is also available in PDF format. This can be found by going to *Start → Programs → ARM → RealView Development Suite v4.0 → RVDS v4.0 Documentation Suite*. To access an online version of the documentation, you can go to http://infocenter.arm.com/help/topic/com.arm.doc.set.swdev/index.html.

Section 1: Command Line Tools and executing images in RealView Debugger (RVD)
This section covers the command line tools required to create and examine executable images from the command line, and using RealView Debugger (RVD) to configure a connection to a simulator and execute an image.

The command line tools include:

`armcc`	ARM C compiler
`armlink`	Object code linker
`armasm`	Assembler for ARM/Thumb source code
`fromelf`	File format conversion tool

Help is available from the command line for all of the tools covered in this session by typing the name of the tool followed by **`--help`**.

For more details please refer to the following documentation: *Compiler User Guide, Compiler Reference Guide, Linker User Guide, Linker Reference Guide, Utilities Guide.*

Consider the following simple C program which calls a subroutine. This file is provided as **`hello.c`** in **`c:\armprac\intro\session1\`**

```
/* hello.c Example code */

#include <stdio.h>
#include <stdlib.h> /*for size_t*/

void subroutine(const char *message)
{
  printf(message);
}


int main(void)
{
  const char *greeting = "Hello from subroutine\n";
  printf("Hello World from main\n");
  subroutine(greeting);
  printf("And Goodbye from main\n\n");
  return 0;
}
```

Exercise 1.1 - Compiling and running the example

Compile this program with the ARM C compiler:

```
armcc -g hello.c
```

The C source code is compiled and an ARM ELF object file, **`hello.o`**, is created. The compiler also automatically invokes the linker to produce an executable with the default executable filename **`__image.axf`**.

The **`-g`** option adds high level debugging information to the object/executable. If **`–g`** is not specified then the program will still produce the same results when executed but it will not be possible to perform high level language debugging operations.

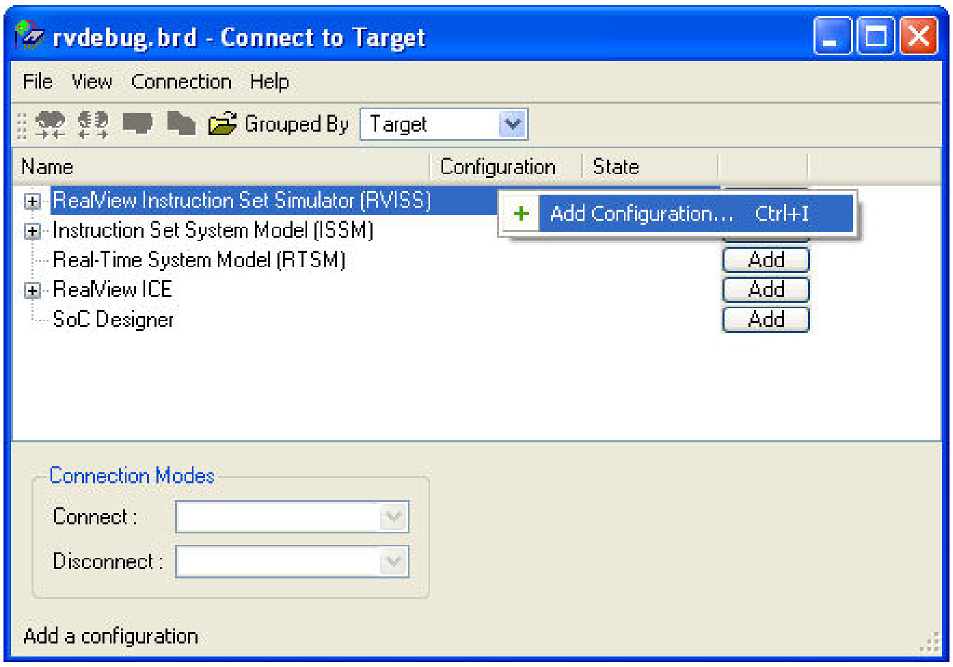
Thus this command will compile the C code, link with the default C library and produce an ARM ELF format executable called **`__image.axf`**.

Exercise 1.2 – Executing the example in RVD

Before the image can be executed it must be loaded to an appropriate target using the debugger. This example will use the RealView Instruction Set Simulator (RVISS) as the target to execute the image using RealView Debugger (RVD).

Start RVD by clicking on the icon in the Windows Start Menu folder.
ARM →RealView Development Suite v4.0 → RealView Debugger v4.0

Select *Target → Connect to Target…* from the menu, or click the hyperlink to launch the *Connection Control* window.


rvdebug.brd - Connect to Target
File View Connection Help
Grouped By Target
Name Configuration State
Add Configuration... Ctrl+I
Add
Add
Add

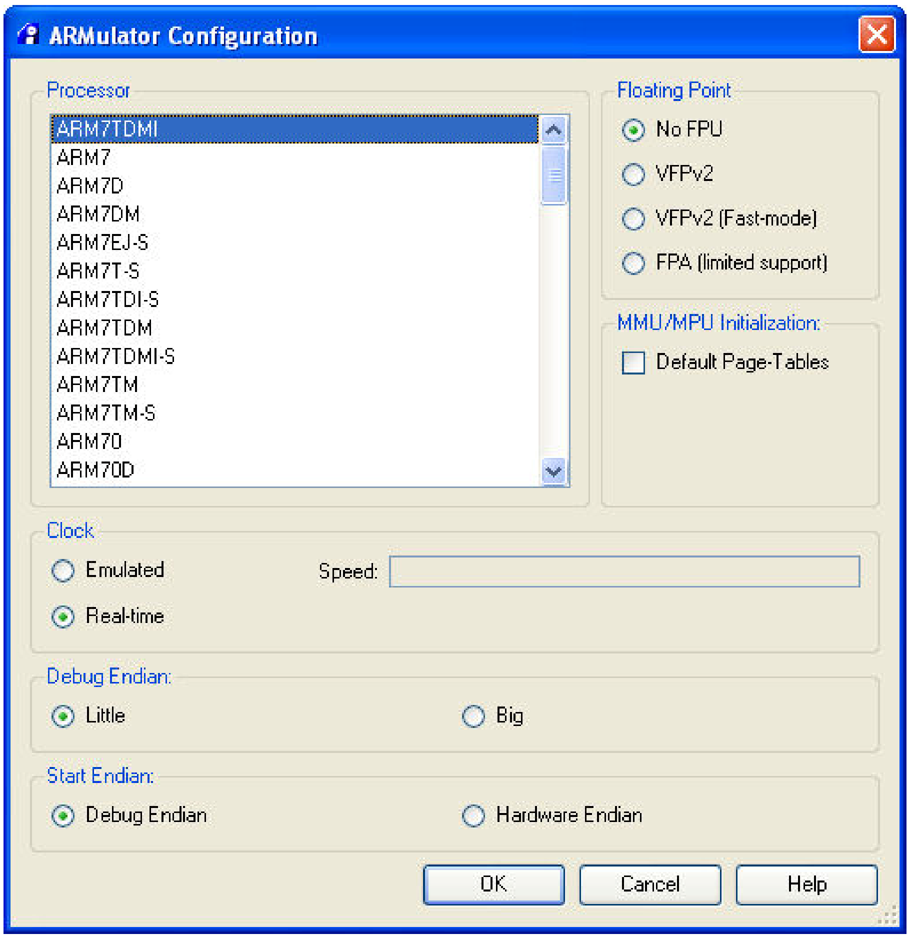
RealView Instruction Set Simulator (RVISS)
Instruction Set System Model (ISSM)
Real-Time System Model (RTSM)
RealView ICE
SoC Designer

Connection Modes
Connect :
Disconnect :

Add a configuration

Right click on the RealView Instruction Set Simulator branch, and select *Add Configuration…*

The *ARMulator Configuration* window appears.



Processor: ARM7TDMI (selected), ARM7, ARM7D, ARM7DM, ARM7EJ-S, ARM7T-S, ARM7TDI-S, ARM7TDM, ARM7TDMI-S, ARM7TM, ARM7TM-S, ARM70, ARM70D

Floating Point: [✓ No FPU] VFPv2 VFPv2 (Fast-mode) FPA (limited support)

MMU/MPU Initialization: ☐ Default Page-Tables

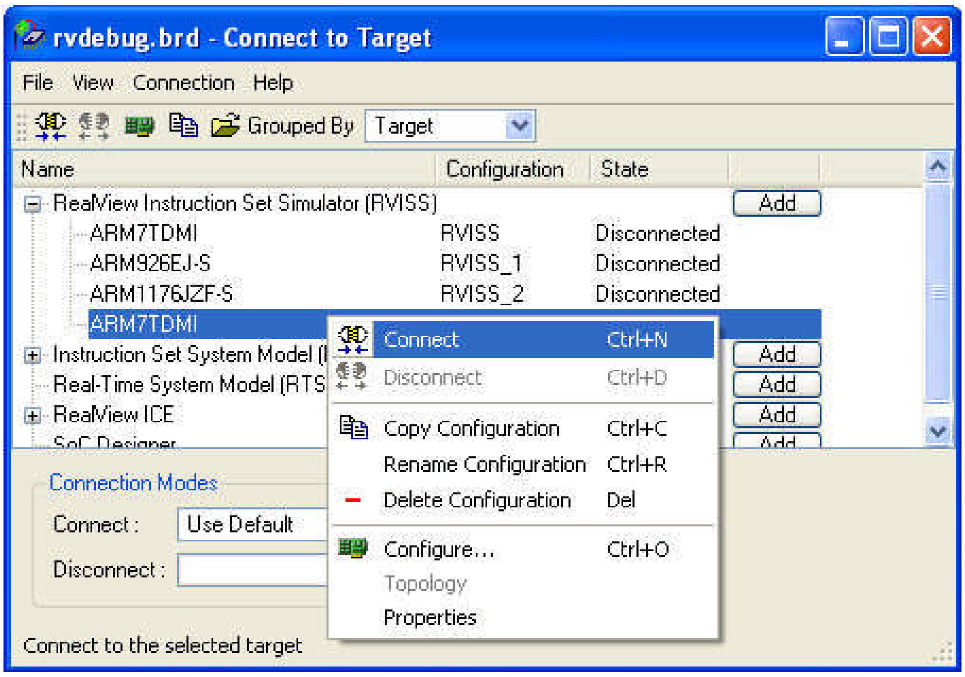
Clock: Emulated [✓ Real-time] Speed:

Debug Endian: [✓ Little] Big

Start Endian: [✓ Debug Endian] Hardware Endian

Select ARM7TDMI as shown above and click *OK* to return to the Connection Control window.

Right click on the new ARM7TDMI entry in the connection control window and click *Connect*.


rvdebug.brd - Connect to Target
File View Connection Help
Grouped By Target
Name Configuration State
RealView Instruction Set Simulator (RVISS) Add
ARM7TDMI RVISS Disconnected
ARM926EJ-S RVISS_1 Disconnected
ARM1176JZF-S RVISS_2 Disconnected
ARM7TDMI
Instruction Set System Model (I Add
Real-Time System Model (RTS Add
RealView ICE Add
SoC Designer Add
Connect Ctrl+N
Disconnect Ctrl+D
Copy Configuration Ctrl+C
Rename Configuration Ctrl+R
Delete Configuration Del
Configure... Ctrl+O
Topology
Properties
Connection Modes
Connect : Use Default
Disconnect :
Connect to the selected target

RealView Debugger is now connected to the ARM7TDMI RVISS target.

Select *Target* → *Load Image* and navigate to *\intro\session1*, select *__image.axf* and click *Open* to load the image into the debugger.

Select *Debug* → *Run* from the menu *(F5).*

RVD displays the following in the *StdIO* tab:

```
Hello World from main
Hello from subroutine
And Goodbye from main
```

Leave the debugger open for use later in the exercise.

Exercise 1.3 - Compilation options

Different arguments can be passed to the compiler from the command line to customize the output generated. A list of the more common options, together with their effects, can be viewed by entering **`armcc --help`** at the command line. Some of these options are listed below:

`-c`	Generate object code only, does not invoke the linker
`-o <filename>`	Name the generated output file as ‘filename’
`-S`	Generate an assembly language listing
`-S --interleave`	Generate assembly interleaved with source code
`--thumb`	Generate Thumb code

When the compiler is asked to generate a non-object output file, for example when using -c or -S, the linker is **not** invoked, and an executable image will not be created. These arguments apply to both the ARM and Thumb C compilers.

RVCT uses a `--` prefix for multi character switches like **`interleave`**.

Use the compiler options with **`armcc`** to generate the following output files from **`hello.c`**:

`image.axf`	An ARM executable image
`source.s`	An ARM assembly source
`inter.s`	A listing of assembly interleaved with source code
`thumb.axf`	A Thumb executable image
`thumb.s`	A Thumb assembly source

Use a suitable text editor to view the interleaved source file.

To use Notepad from the command line type **`notepad <filename>`**.

Note the sections of assembly source that correspond to the interleaved C source code.

Exercise 1.4 - armlink

In previous exercises we have seen how the compiler can be used to automatically invoke the linker to produce an executable image. **`armlink`** can be invoked explicitly to create an executable image by linking object files with the required library files. This exercise will use the files, **`main.c`** and **`sub.c`** which can be linked to produce a similar executable to the one seen in the previous exercises.

Use the compiler to produce ARM object code files from each of the two source files.

Remember to use the **`-c`** option to prevent automatic linking.

Use **`armlink main.o sub.o -o link.axf`** to create a new ARM executable called **`link.axf.`**

`armlink` is capable of linking both ARM and Thumb objects.
If the **`-o`** option is not used an executable with the default filename, __**`image.axf`**, will be created.

Load the executable into RVD and run - check that the output is the same as before.

The ability to link files in this way is particularly useful when link order is important, or when different C source modules have different compilation requirements. It is also useful when linking with assembler object files.

Exercise 1.5 - fromelf

ARM ELF format objects and ARM ELF executable images that are produced by the compilers, assembler and/or linker can be decoded using the **`fromelf`** utility and the output examined. Shown below is an example using the **`-c`** option to produce decoded output, showing disassembled code areas, from the file **`hello.o`**:

```
fromelf -c hello.o
```

Alternatively re-direct the output to another file to enable viewing with a text editor:

```
fromelf -c hello.o > hello.txt
```

Use the **`fromelf`** utility to produce and view disassembled code listings from the **`main.o`** and **`sub.o`** object files.

A complete list of options available for '**`fromelf`**' can be found from the command line using fromelf **`--help`**, or by consulting the online documentation.

Section 1 - Review

We have now seen how the command line tools can be used to compile and link and link simple projects.

`armcc`	The compiler can be called with many different options. The **`-g`** option is required to enable source level debugging. The compiler can be used to generate executable images, object files and assembly listings.
`armasm`	The assembler can be used to construct object files directly from assembly source code.
`armlink`	The linker can be used to produce executable images from ARM or Thumb object files.
`fromelf`	The **`fromelf`** utility can be used to generate disassembled code listings from ARM or Thumb object or image files.

Help is available from the command line. Alternatively, consult the online documentation for further information.

We have seen how RVD can be used to:

- Set up and connect to an RVISS target.
- Load and execute an image.

Section 2: Creating projects using ARM Workbench and debugging using RVD

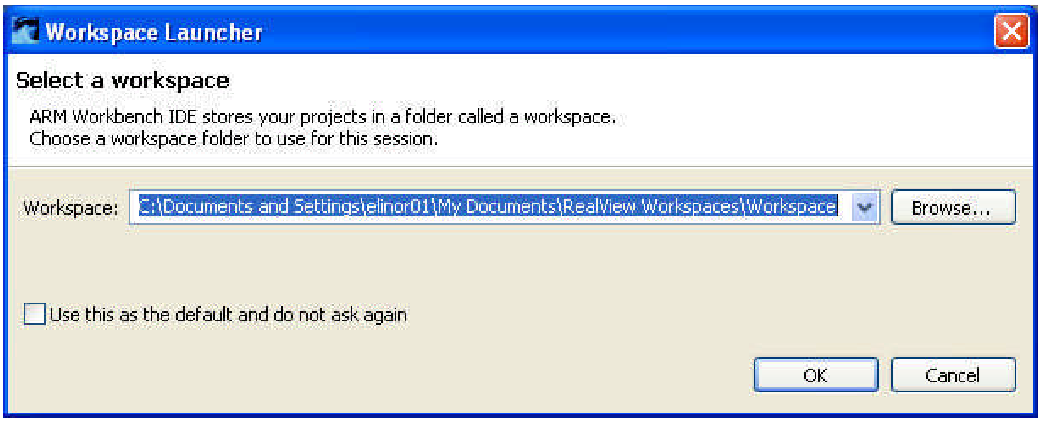
In this session we will see how the ARM Workbench Integrated Development Environment can be used with RealView Debugger (RVD) to create and develop projects.

Exercise 2.1 - Creating a new project

In this exercise, we will create a new project in the ARM Workbench.

Start ARM Workbench IDE by clicking on the icon in the Windows Start Menu folder: *ARM→ ARM Workbench IDE v4.0*

In the Workspace launcher, browse to the desired workspace, or stay with the default. Make a note of this directory, and click *OK*.

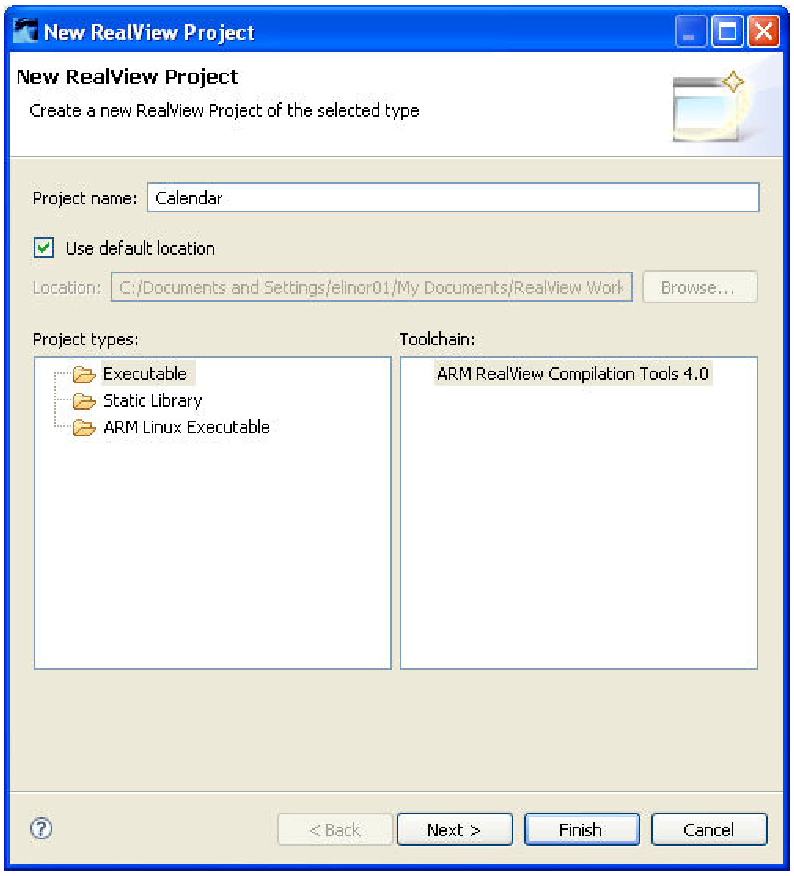

Workspace Launcher

Select a workspace

ARM Workbench IDE stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Documents and Settings\elinor01\My Documents\RealView Workspaces\Workspace

Select *File → New → RealView Project* from the menu.
Enter **`Calendar`** as the Project name. Click *Next*.



New RealView Project

Create a new RealView Project of the selected type

Project name: Calendar

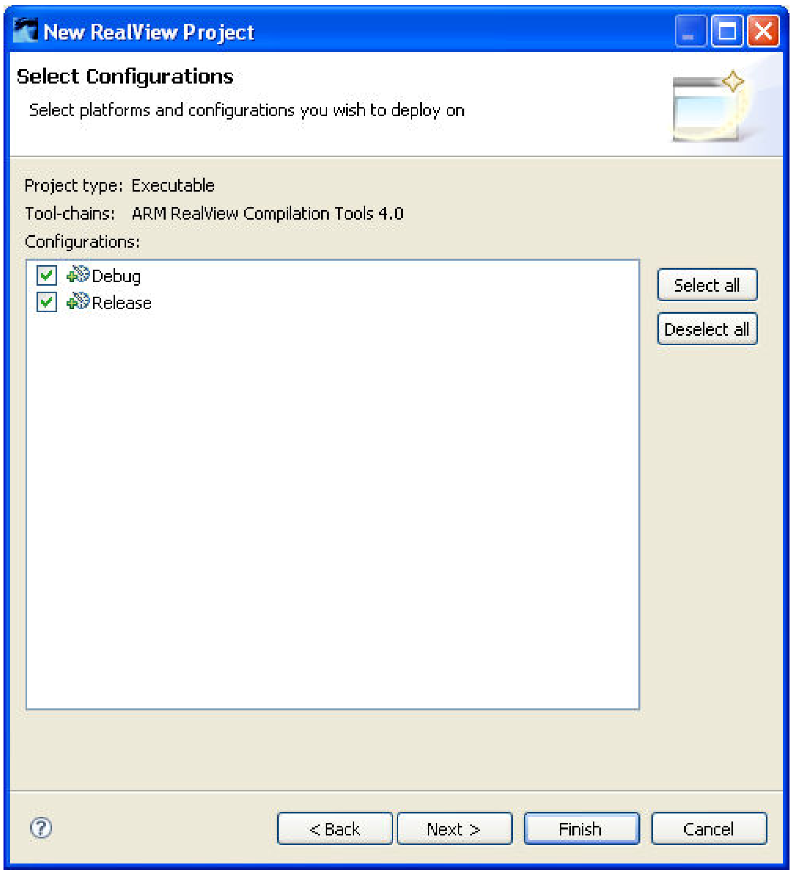
☑ Use default location

Location: C:/Documents and Settings/elinor01/My Documents/RealView Work

Project types:
- Executable
- Static Library
- ARM Linux Executable

Toolchain: ARM RealView Compilation Tools 4.0

You can browse through a set of windows to pre-configure your project using the *Next* button. However, we will use the default configuration for now, so just click *Finish*.



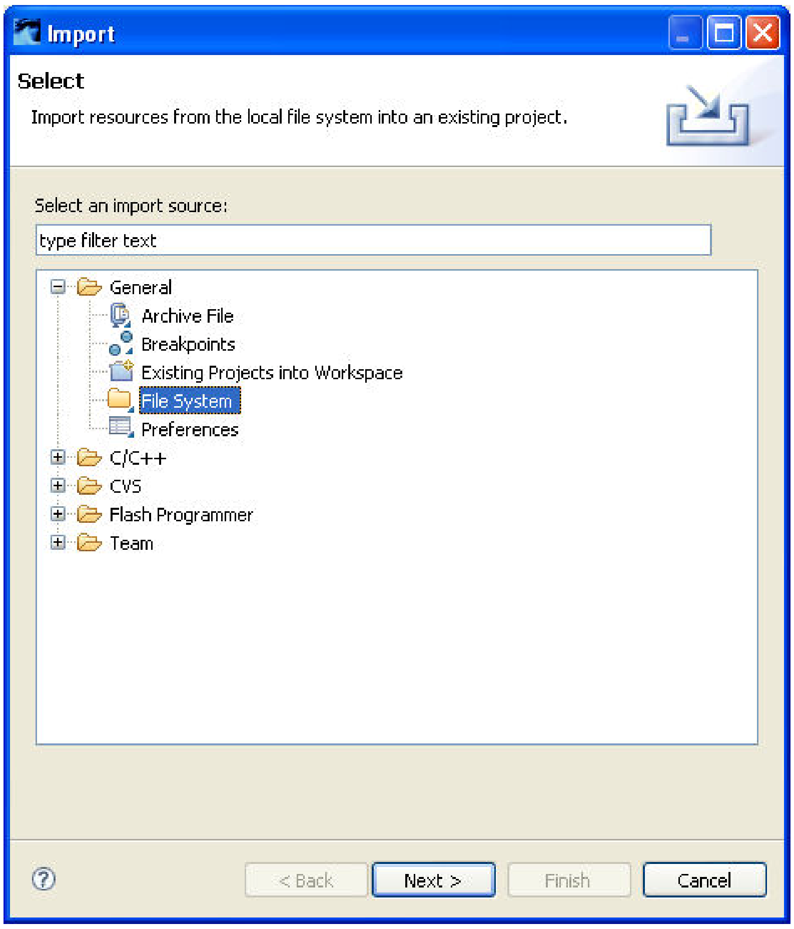
Select Configurations

Select platforms and configurations you wish to deploy on

Project type: Executable
Tool-chains: ARM RealView Compilation Tools 4.0
Configurations:
☑ Debug
☑ Release

We have now created a new blank project in ARM Workbench. Our next step is to import our existing source file into the project.

Select *File* → *Import*…and in the *Import* window, expand the *General* entry. Select *File System* and click *Next*.



Import

Select

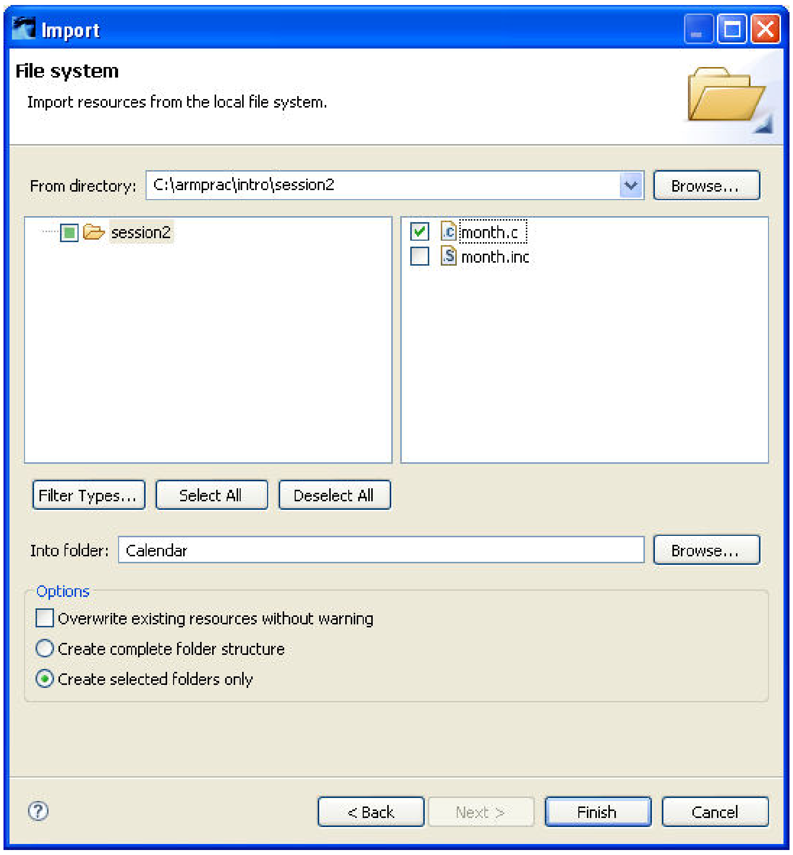
Import resources from the local file system into an existing project.

Select an import source:

type filter text

- General
  - Archive File
  - Breakpoints
  - Existing Projects into Workspace
  - File System
  - Preferences
- C/C++
- CVS
- Flash Programmer
- Team

Adjacent to *From directory:* click Browse and navigate to **`c:\armprac\intro\session2\`**. A list of files should appear on the right, and the subfolder *session2* should appear on the left. Check the box next to **`month.c`** in the right-hand list to specify the file to import.



From directory: C:\armprac\intro\session2

[■] session2

[✓] month.c
[ ] month.inc

Into folder: Calendar

Options
[ ] Overwrite existing resources without warning
○ Create complete folder structure
● Create selected folders only

If **`Calendar`** is not already listed under *Into folder:*, click Browse next to *Into folder:*, and select **`Calendar`** from the list that appears. Click OK, then click Finish in the Import window.

It is possible to import whole subdirectories by checking the box next to the subfolder on the left-hand side. However, in this example we only require the single **`.c`** file.

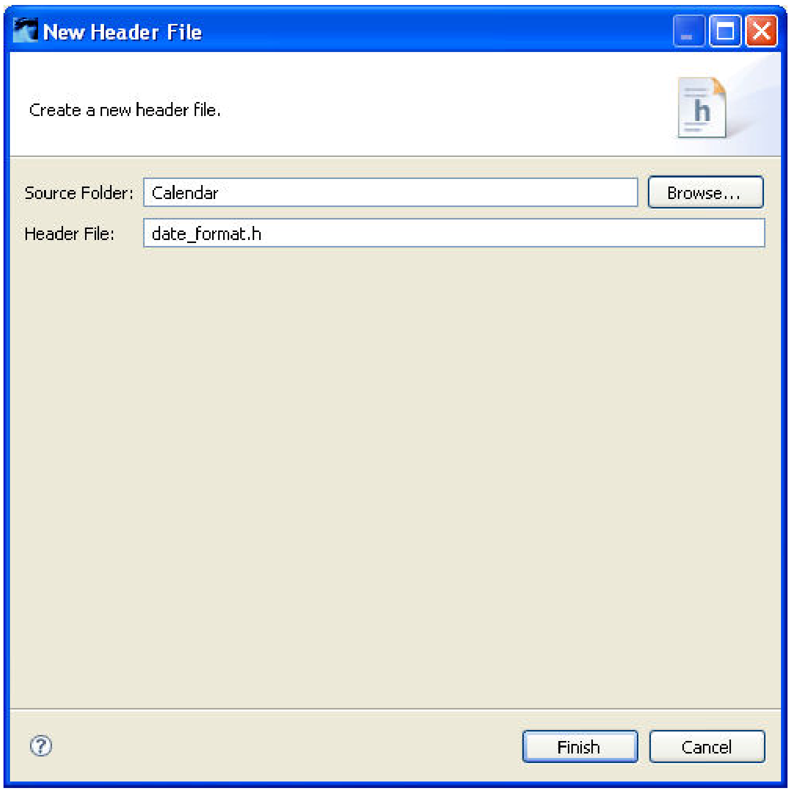
You have now created a new ARM project in ARM Workbench, and added the source file **`month.c`** to it.

Leave the editor window open for use in the next exercise, where we will add a header file to the project.

Exercise 2.2 - Creating a header file

In this exercise, we will create a new header file using ARM Workbench.

Select *File* → *New* → *Header File*. The *New Header File* window appears. If it is not there already, enter **`Calendar`** as the Source Folder. Enter **`date_format.h`** as the Header File. Click *Finish*.

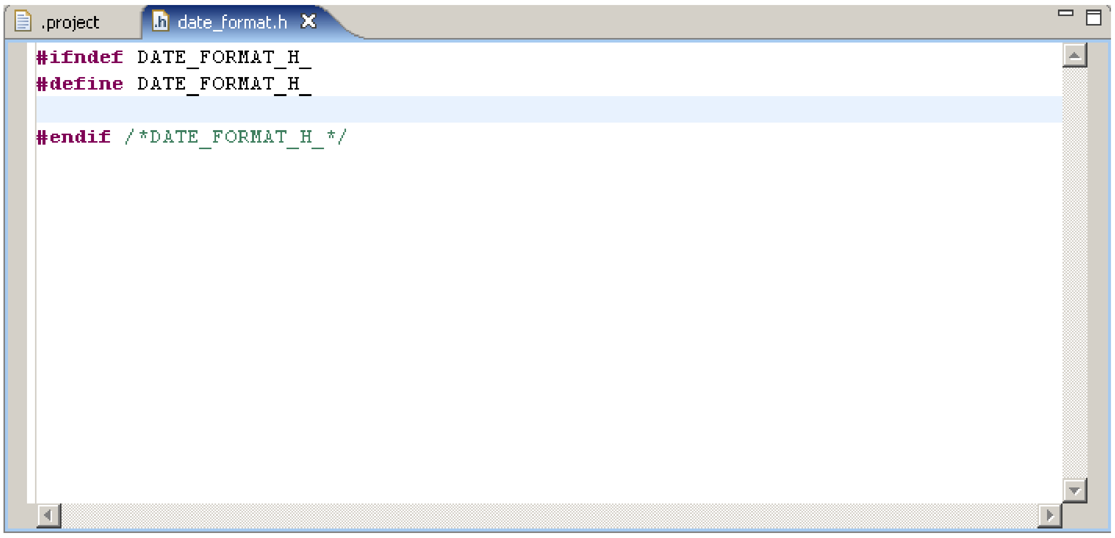


Create a new header file.

Source Folder: Calendar

Header File: date_format.h

The file **`date_format.h`** now appears in ARM Workbench. Note that this file has now been automatically added to the **`Calendar`** project directory.



```
#ifndef DATE_FORMAT_H_
#define DATE_FORMAT_H_

#endif /*DATE_FORMAT_H_*/
```

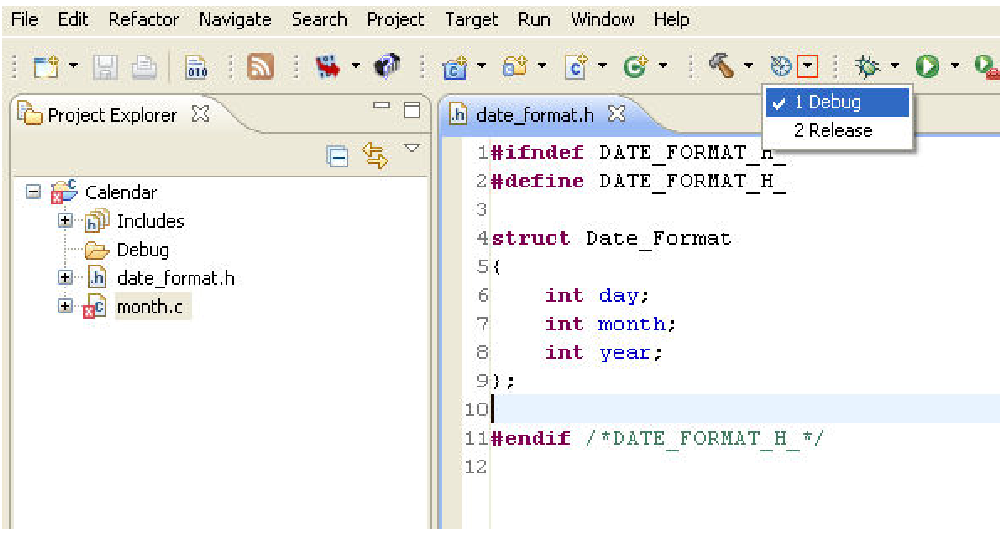
We need to add some code to this file. Modify the file to include the following C struct definition:

```
#ifndef DATE_FORMAT_H_
#define DATE_FORMAT_H_

struct Date_Format
{
      int day;
      int month;
      int year;
};

#endif /*DATE_FORMAT_H_*/
```

Select *File* → *Save* from the menu. This will cause the project to rebuild. Ignore any project error messages for the time being.

Finally, we must check that the correct build target is selected.

Ensure the *Debug* target is the active target in the project window, by selecting the Active Build Configuration from the list box shown below:



```
#ifndef DATE_FORMAT_H_
#define DATE_FORMAT_H_

struct Date_Format
{
    int day;
    int month;
    int year;
};

#endif /*DATE_FORMAT_H_*/
```

The two default targets available refer to the level of debug information contained in the resultant image.

Debug	Contains full debug table information and very limited optimization.
Release	Enables full optimization, resulting in a worse debug view.

It is the *Debug* build target that we shall use for the remainder of this tutorial.

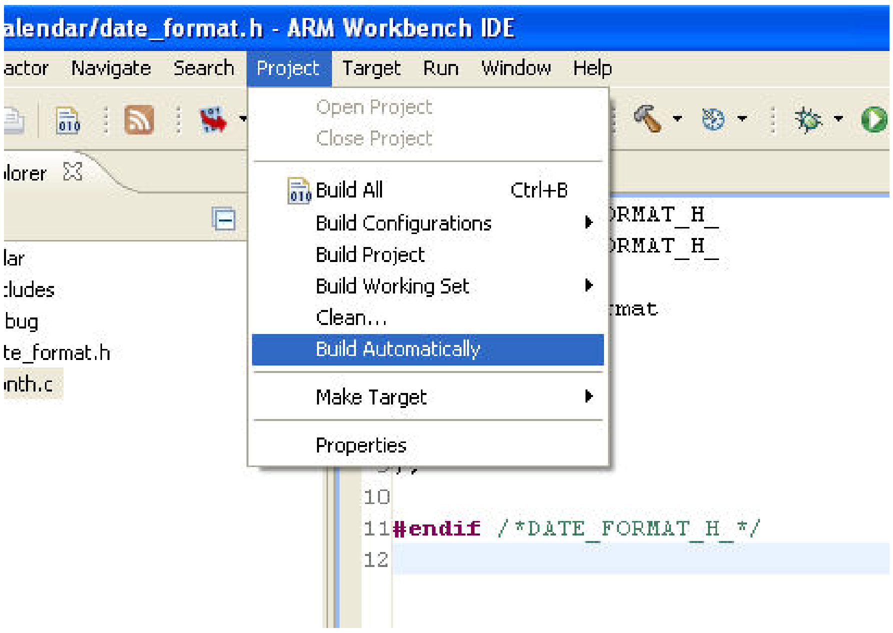
You have now created a very simple header file as part of your project.

Leave the editor window open for use later in the exercise, where we will build the project in ARM Workbench.

Exercise 2.3 - Building the project (Debug target)

In this exercise, we will build our Calendar Project.

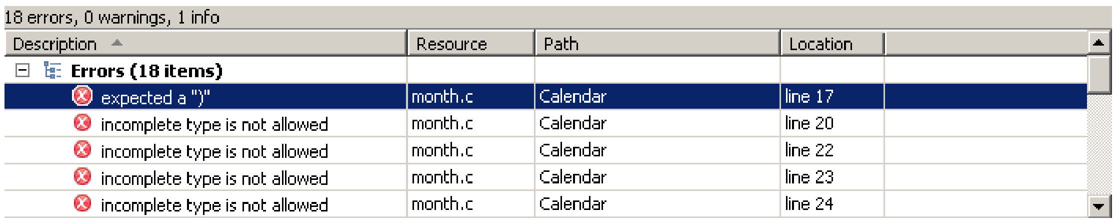
ARM Workbench is set to automatically build our project by default. Disable automatic build, by deselecting *Build Automatically* in the Project menu.



```
RMAT_H_
RMAT_H_
rmat
#endif /*DATE_FORMAT_H_*/
```

The project should already have been built from the previous exercise. If not, select *Project→ Build Project* from the menu.

The *Problems* window appears with the several Errors and Warnings messages generated as a result of the attempted build:



18 errors, 0 warnings, 1 info

Description	Resource	Path	Location
Errors (18 items)			
expected a ")"	month.c	Calendar	line 17
incomplete type is not allowed	month.c	Calendar	line 20
incomplete type is not allowed	month.c	Calendar	line 22
incomplete type is not allowed	month.c	Calendar	line 23
incomplete type is not allowed	month.c	Calendar	line 24

The first error is at line 17, in **`month.c`**, and reads **`expected a ")"`**. Double click on it to open the relevant source file in the code pane.

There is something wrong with the code; a close bracket is missing. The line should read:

```
printf("\ne.g. 1972 02 17\n\n");
```

Correct the error by adding the missing bracket and then save the updated source file. Rebuild the project.

The *Errors & Warnings* window again shows the errors associated with the failed build. The first error message is:

```
Error    :  #70: incomplete type is not allowed
month.c line 20
```

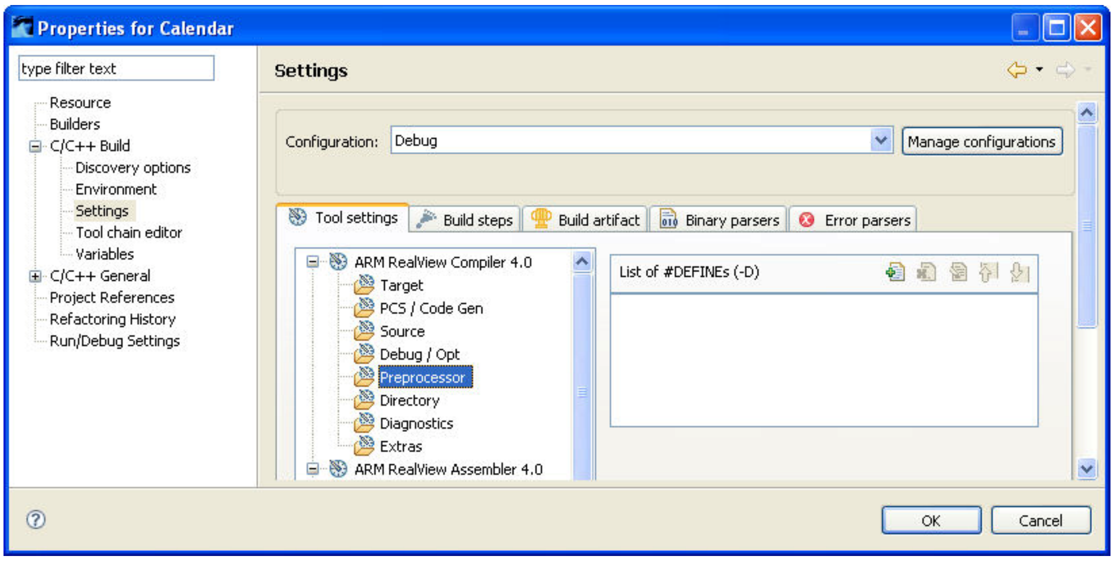
Once again, the code pane of the *Problems* window displays the relevant source file and an arrow highlights the line of code associated with the first error message. You will find that there is nothing wrong with the code on this line!

Towards the top of the file, the preprocessor directives contain a reference to the macro **`INCLUDE_DATE_FORMAT`**, which has not been defined in any of the source files. Normally a command line parameter would have to be supplied to the C compiler, **`armcc`**, to specify:

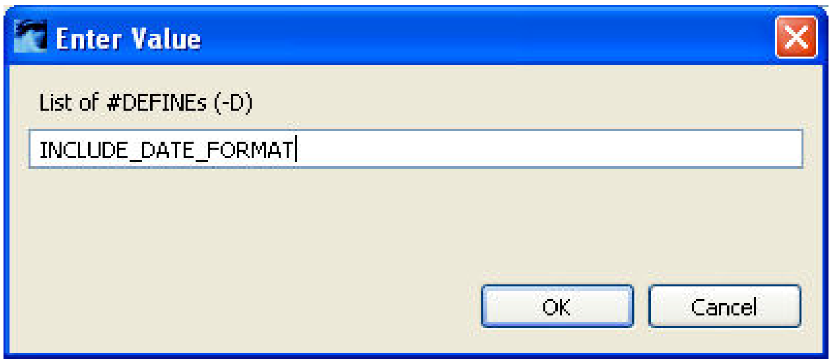
```
-D INCLUDE_DATE_FORMAT
```

We must edit the command line arguments for this project's settings.

Open the *Project Properties* window by selecting *Project*→ *Properties* from the menu. Expand *C/C++ Build* from the side menu, and select *Settings*, and browse to the *Preprocessor* entry.



Click the *Add* button above the list box, and in the window that appears, enter **`INCLUDE_DATE_FORMAT`**. Click *OK*.



We also need to change the optimization level to –O1. This is so that variables are stored in registers automatically, while still retaining a reasonable debug view.

Select the *Debug/Opt* entry. In the list of optimization levels, select Restricted optimization, good debug (1). Click *OK* to close the *Project Properties* window.

Our command line options have now been correctly set.

Select *Project→ Build Project* from the menu.

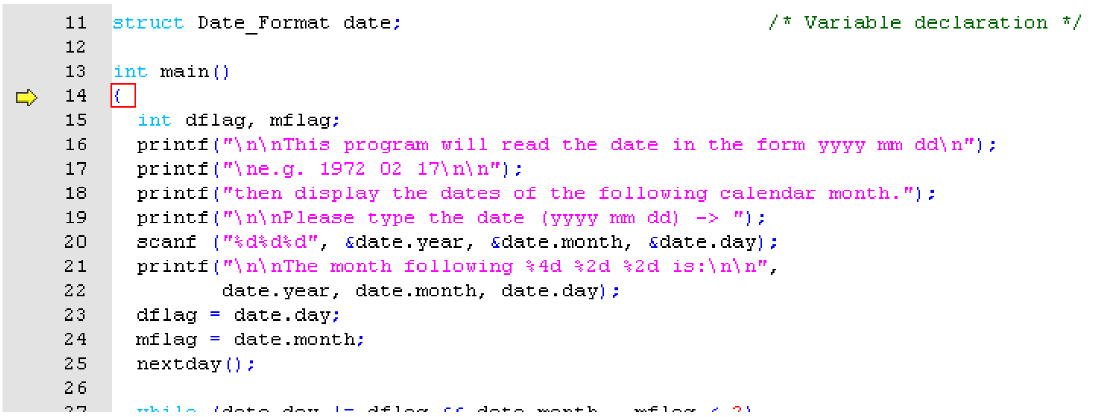
If a project is already up-to-date then nothing will be done by the IDE when it is requested to build a project. If you wish to do a forced rebuild of all the source files then select *Project → Clean…* to delete the relevant object files.

Exercise 2.4 - Executing the example in RVD

Ensure that there are no existing RVD windows running. From the ARM Workbench main menu, select *Run* → *Debug As* → *Load into RealViewDebugger* to start RVD.

In RVD, connect to the ARM7TDMI RVISS target. Select *Target* → *Load Image…* and browse to **`/Calendar/Debug/Calendar.axf`** in your ARM Workbench workspace folder. Click *Open*.

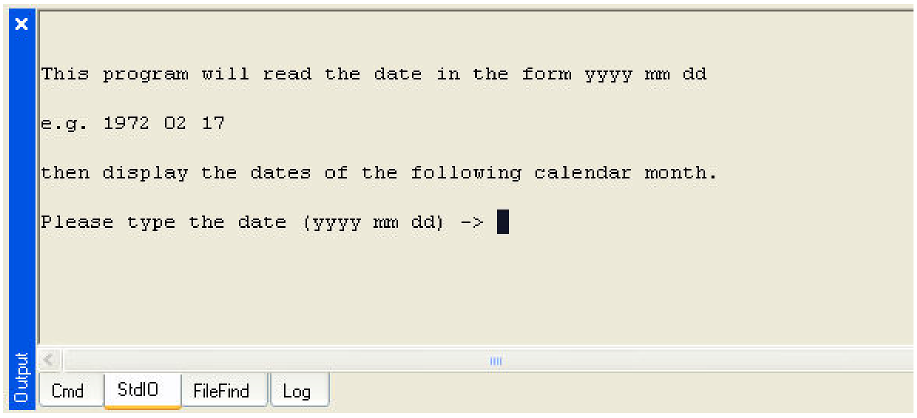
The *Code* pane now shows that the image is loaded and the red box indicates the current execution position in the source view.



```
struct Date_Format date;                              /* Variable declaration */

int main()
{
  int dflag, mflag;
  printf("\n\nThis program will read the date in the form yyyy mm dd\n");
  printf("\ne.g. 1972 02 17\n\n");
  printf("then display the dates of the following calendar month.");
  printf("\n\nPlease type the date (yyyy mm dd) -> ");
  scanf ("%d%d%d", &date.year, &date.month, &date.day);
  printf("\n\nThe month following %4d %2d %2d is:\n\n",
         date.year, date.month, date.day);
  dflag = date.day;
  mflag = date.month;
  nextday();
```

Select *Debug* → *Run* from the menu *(F5)*.

Execution begins. The *Output* pane at the bottom of the window shows the *StdIO* tab which performs console I/O operations for the current image. The program is now awaiting user input.



```
This program will read the date in the form yyyy mm dd

e.g. 1972 02 17

then display the dates of the following calendar month.

Please type the date (yyyy mm dd) -> 
```

Enter today's date in the format described, e.g. **`2009 01 11`**

The program will display the dates for the following calendar month and then terminate.

Note that there is no source code available for the system exit routines and RVD displays **`Stopped at 0x0000A728 SYS_S\_sys_exit`** in the CMD tab.

The disassembled project code can be viewed by selecting the *Disassembly* tab in the *Code* pane.

All windows can be resized by clicking and dragging at the edges.

Docked panes can be changed to floating windows by dragging and dropping.

Exercise 2.5 - Debugging the example

	Select *Target* → *Reload Image to Target* from the menu.

RVD will load the image ready for debugging. Again the current execution position is shown at the image entry point.

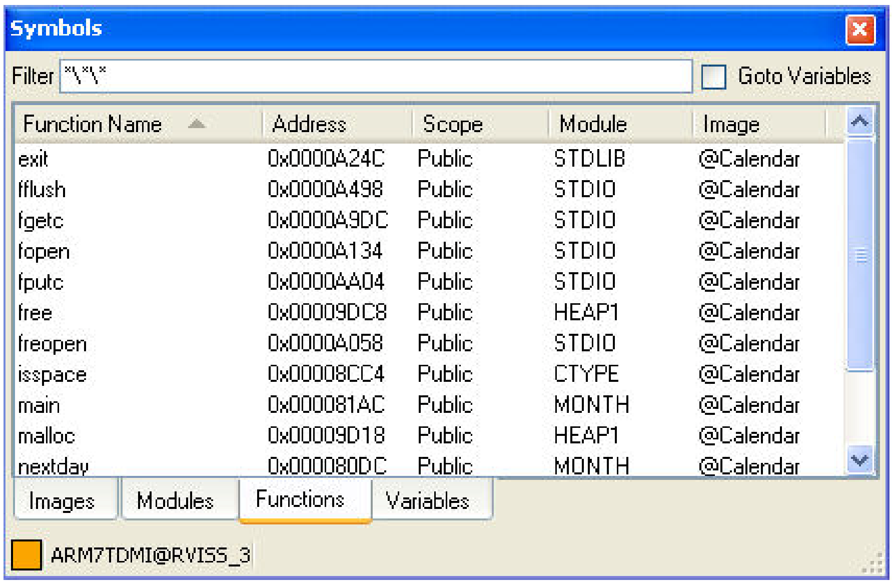
	Select *Debug* → *Run* from the menu *(F5)*.

	You will once again be prompted to enter a date. This time enter **`2009 11 30`**. The program will terminate after it has output the set of dates for the following month.

	Use the scroll bar at the edge of the *Output Pane* to view the dates at the end of November. You will find that there is an extra day!

	Reload the image into the debugger.

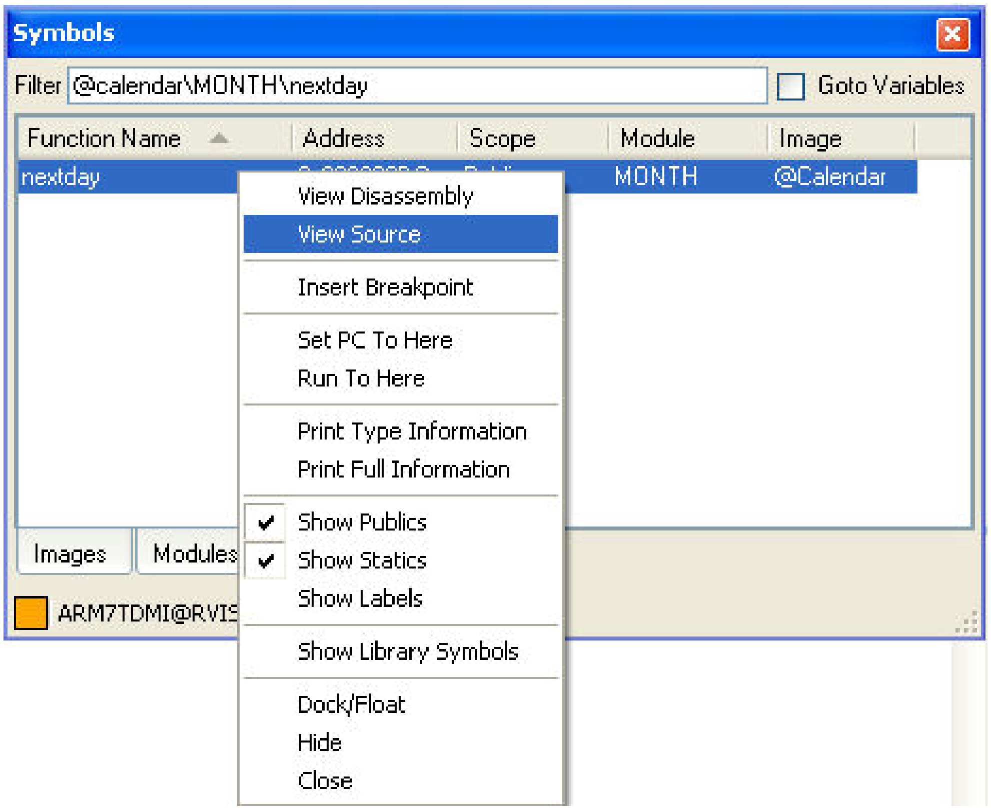
	Select *View* → *Symbols* from the menu to open a *Symbols* pane. Select the *Functions* tab.



Function Name	Address	Scope	Module	Image
exit	0x0000A24C	Public	STDLIB	@Calendar
fflush	0x0000A498	Public	STDIO	@Calendar
fgetc	0x0000A9DC	Public	STDIO	@Calendar
fopen	0x0000A134	Public	STDIO	@Calendar
fputc	0x0000AA04	Public	STDIO	@Calendar
free	0x00009DC8	Public	HEAP1	@Calendar
freopen	0x0000A058	Public	STDIO	@Calendar
isspace	0x00008CC4	Public	CTYPE	@Calendar
main	0x000081AC	Public	MONTH	@Calendar
malloc	0x00009D18	Public	HEAP1	@Calendar
nextday	0x000080DC	Public	MONTH	@Calendar

Enter the string **`@calendar\MONTH\nextday`** in the *Filter* textbox and press *Enter*.

The basic filter strings are of the form: **`@image_name\module_name\function_name`**, and you can also use wildcards. For example, to list all of the functions in the month module, you can specify: **`@calendar\MONTH\*`**. Full details of additional filters that can be used can be found in the RVD User Guide.



Highlight the **`nextday`** entry, right-click it and select *View Source* from the context menu to locate this function in the source file. Then close the *Symbols* pane.

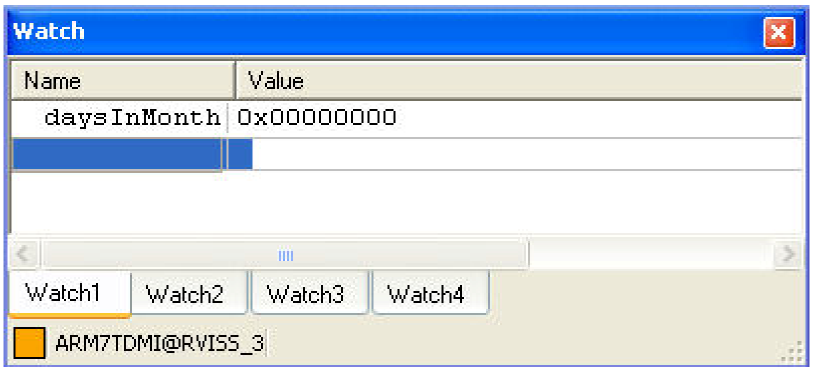
Set a breakpoint on the **`switch`** statement on line **`40`** by double-clicking to the left of the line number.

The line will receive a red breakpoint marker.

Resume execution and enter the date **`2009 11 30`** again. The program will stop at the breakpoint.

Right click on the variable name **daysInMonth** on line **38** and select *Add Watch* from the context menu.

A new entry appears in the *Watch* pane showing **daysInMonth**. Its value has not been determined yet and it is currently set to **0**.



Name	Value
daysInMonth	0x00000000

Right click on the word **date** in the variable name **date.month** on line **40** and select *Add Watch* from the context menu.

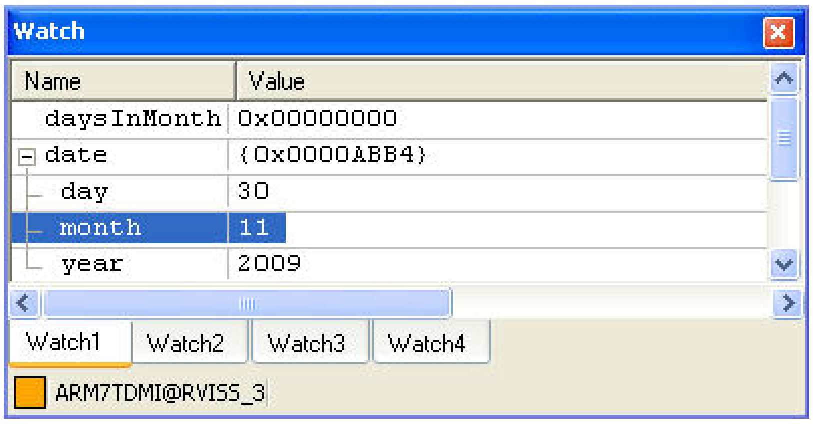
The display in the *Watch* pane is updated. The **date** struct is now visible.

Click on the cross to the left of **date** to view the struct's fields.

Right click on the **date** field and select *Format…* from the context menu to change the display format of the variables.

Select *Signed Decimal* from the *List Selection* dialog and click *OK*.

The *Watch* pane is updated again:



Name	Value
daysInMonth	0x00000000
⊟ date	{0x0000ABB4}
day	30
month	11
year	2009

Select *Debug* → *Step Over (next)* (*F10*) to perform the next step in the program.

You will note that the case statements have been skipped and that the **`default`** path will be taken.

The **`default`** path assumes the month has 31 days. This is not correct for November. There is a fragment of code, "**`case 11:`**", missing from line **`51`**. To rectify this permanently we would have to edit the source file. For the purposes of this example we will modify the variable **`daysInMonth`** to produce the desired result.

Double-click on the breakpoint set on line **`40`** to remove it.

Set a new breakpoint on line **`58`** after the block of code containing the **`switch`** statement.

Resume program execution, the debugger will stop at the new breakpoint.

Right click on the **`daysInMonth`** variable in the *Watch* pane and change the format of this variable to *Signed Decimal*.

You will see that the value of **`daysInMonth`** is **`31`**, but we require it to be **`30`**.

Click on the value to edit it and change the value to **`30`**, then press enter.

Remove the breakpoint on line **`58`**.

Resume the program and finish executing the example.

Note that the output generated by the program is now correct.

Exercise 2.6 – Viewing registers and memory

	Reload the image into the debugger.

RVD will load the image ready for debugging. Again the current execution position is shown at **`main`**.

	Set a breakpoint on the **`printf`** statement on line **`29`** by double clicking in the region to the left of the statement.

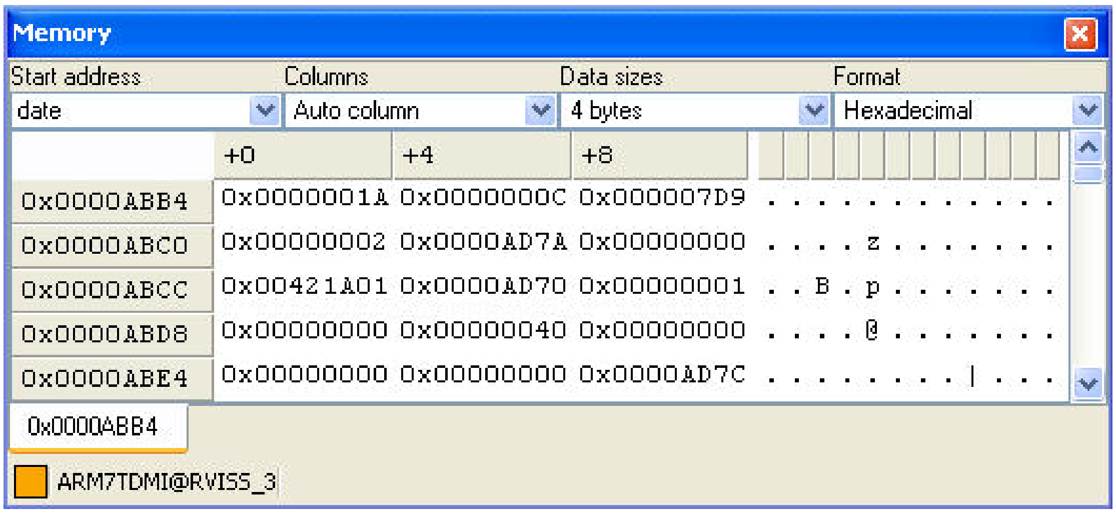
	Select *Debug* → *Run* from the menu *(F5)*.

You will once again be prompted to enter a date.

	This time enter **`2009 12 25`**.

The program will stop at the breakpoint on the printf statement.

	Select *View* → *Memory* to open a *Memory* Pane. In the top-left-hand corner of the *Memory* pane is an address box. Type **`date`** and press Enter.

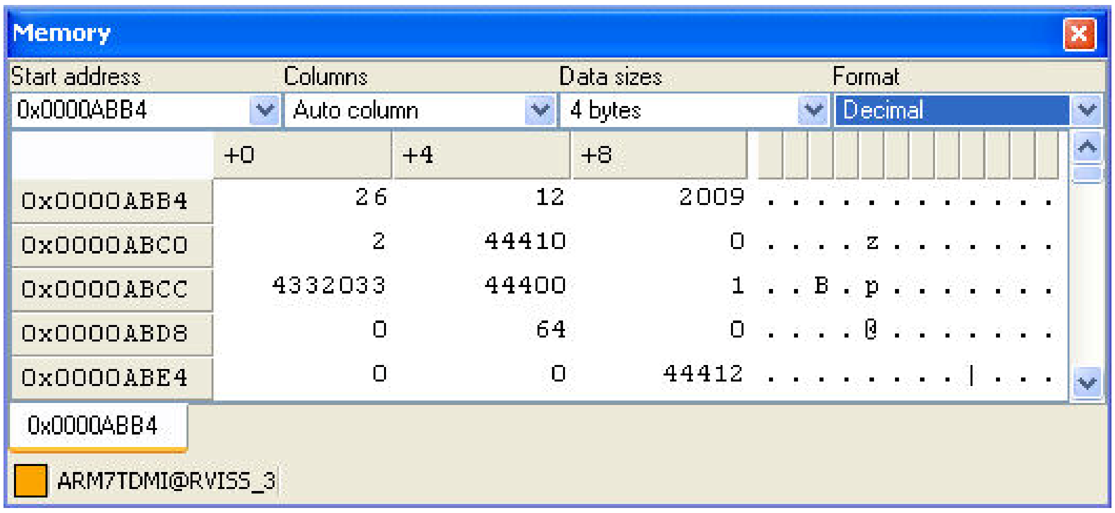


Start address: date
Columns: Auto column
Data sizes: 4 bytes
Format: Hexadecimal

```
            +0         +4         +8
0x0000ABB4  0x0000001A 0x0000000C 0x000007D9  . . . . . . . . . . . .
0x0000ABC0  0x00000002 0x0000AD7A 0x00000000  . . . . z . . . . . . .
0x0000ABCC  0x00421A01 0x0000AD70 0x00000001  . . B . p . . . . . . .
0x0000ABD8  0x00000000 0x00000040 0x00000000  . . . . @ . . . . . . .
0x0000ABE4  0x00000000 0x00000000 0x0000AD7C  . . . . . . . . | . . .
```

The *Memory* pane is updated and now shows memory beginning at the address of the **`date`** struct.

Select *Decimal* from the *Format* list, and select 4 bytes from the *Data Sizes* List.



Memory

Start address: 0x0000ABB4 | Columns: Auto column | Data sizes: 4 bytes | Format: Decimal

```
              +0        +4        +8
0x0000ABB4    26        12        2009   ............
0x0000ABC0    2         44410     0      .....z......
0x0000ABCC    4332033   44400     1      ..B.p.......
0x0000ABD8    0         64        0      .....@......
0x0000ABE4    0         0         44412  .........|...
```

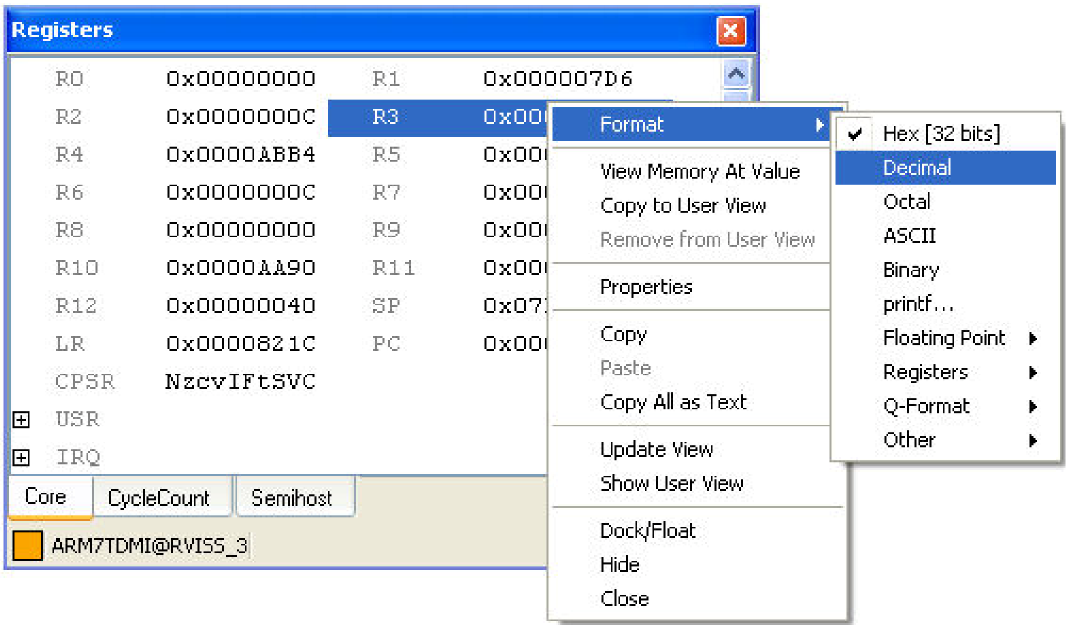
0x0000ABB4

ARM7TDMI@RVISS_3

Note how the three successive words in memory correspond to the three fields in the date struct (**`26/12/2009`**).

Resume the program, execution will stop at the breakpoint again.

Open a register pane by selecting *View* → *Registers* from the menu. Right click on register r3 and select *Format* → *Decimal* to change the register display format.



Registers

```
R0    0x00000000   R1   0x000007D6
R2    0x0000000C   R3   0x000
R4    0x0000ABB4   R5   0x000
R6    0x0000000C   R7   0x000
R8    0x00000000   R9   0x000
R10   0x0000AA90   R11  0x000
R12   0x00000040   SP   0x07
LR    0x0000821C   PC   0x000
CPSR  NzcvIFtSVC
⊞ USR
⊞ IRQ
```

At this point in the program **`r3`** holds the value stored in the **`day`** field of the **`date`** variable in the *Memory* window (the value of **`day`** is now **`27`** as the **`nextday`** function has been called.).

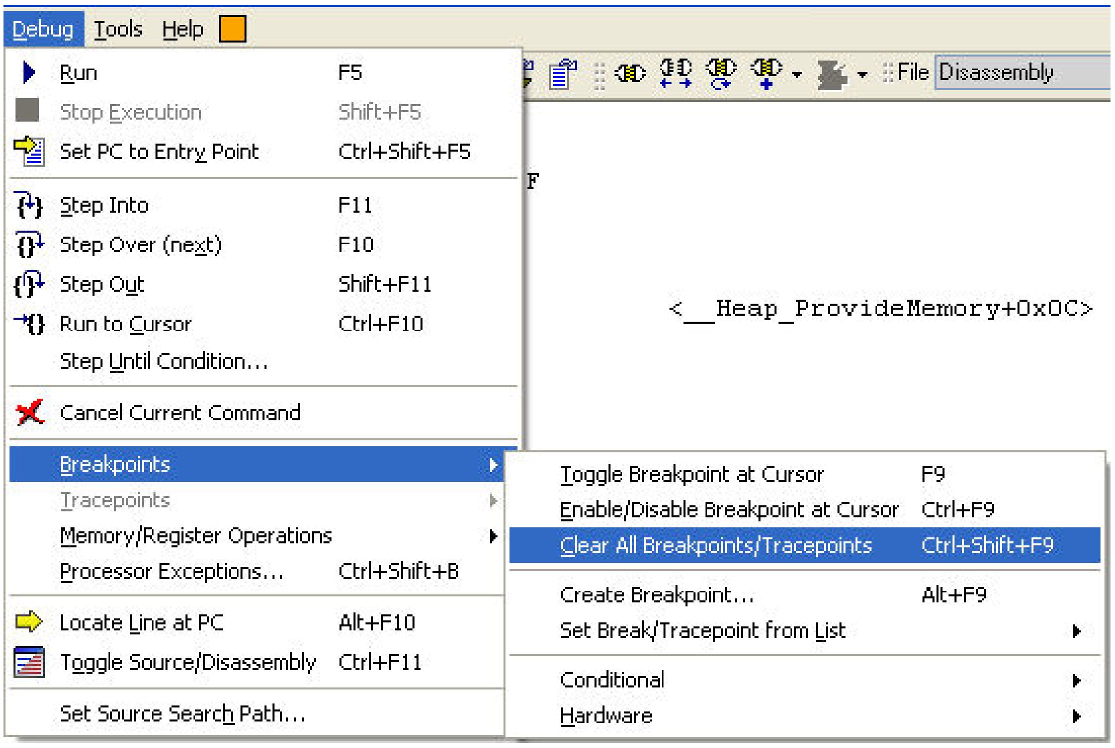
Use the *Go* button to execute the while loop until **`r3`** has the value **`2`**.

Double click on the **`date.day`** value (**`2`**) in the *Watch* pane to edit it. Change it to **`22`** and press Enter.

Use the *Go* button to pass through the while loop until the program ends.

Note how the value entered in the variable watch pane affects the value in the register **`r3`**, the corresponding entry in the memory window and the program output.

Remove all current breakpoints from the image by selecting *Debug* → *Breakpoints* → *Clear All Breakpoints/Tracepoints* from the menu:

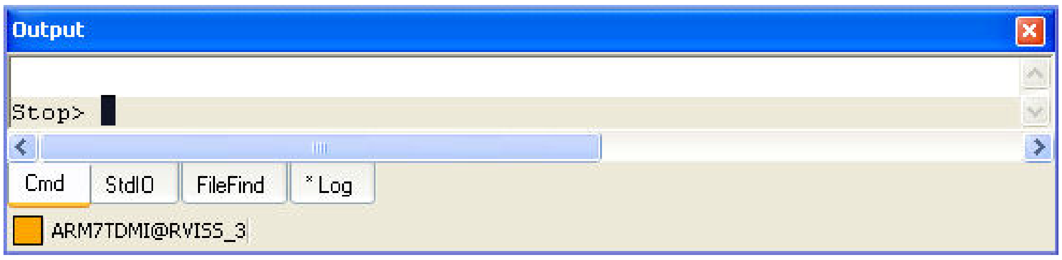

Debug
Tools
Help
File
Disassembly
Run F5
Stop Execution Shift+F5
Set PC to Entry Point Ctrl+Shift+F5
Step Into F11
Step Over (next) F10
Step Out Shift+F11
Run to Cursor Ctrl+F10
Step Until Condition...
Cancel Current Command
Breakpoints
Tracepoints
Memory/Register Operations
Processor Exceptions... Ctrl+Shift+B
Locate Line at PC Alt+F10
Toggle Source/Disassembly Ctrl+F11
Set Source Search Path...
Toggle Breakpoint at Cursor F9
Enable/Disable Breakpoint at Cursor Ctrl+F9
Clear All Breakpoints/Tracepoints Ctrl+Shift+F9
Create Breakpoint... Alt+F9
Set Break/Tracepoint from List
Conditional
Hardware
F
<__Heap_ProvideMemory+0x0C>

Exercise 2.7 – Using the command line

Reload the image into the debugger.

RVD will load the image ready for debugging. Again the current execution position is shown at **`main`**.

Click on the *Cmd* tab of the *Output* pane to view the command line interface, and click in the grey command line bar to give it the focus.



```
Stop>
```

Re-size any other windows as necessary to ensure the command line interface is in clear view.

Set a breakpoint on line **`40`** of the source file by using the **`break`** command then start program execution using **`go`**:

```
Stop> break #40
Stop> go
```

Enter the date **`2009 11 30`** in the *Console* window when prompted.

Execution will stop at the breakpoint. Now check the values of the program variables.

You will need to click on the *Cmd* tab of the *Output* pane to switch focus.

```
Stop> print daysInMonth
```

Note that as it is a C variable the name is case sensitive. The value of **`daysInMonth`** is zero as it is a *static* variable and has not yet been initialized.

```
Stop> print date
```

Remove the breakpoint on line **`40`** using the **`clear`** command.

The **`clear`** command will clear all breakpoints. If you need to clear a specific breakpoint, you can find the reference for the breakpoint using the **`break`** command. This will print a list of current breakpoints. For example to clear the first breakpoint listed type: **clear 1**.

```
Stop> clear
```

Set another breakpoint immediately after the **`switch`** statement then resume program execution.

```
Stop> break #58
Stop> go
```

Check the value of the **`daysInMonth`** variable.

```
Stop> print daysInMonth
```

It is possible to use the cursor keys, ↑ and ↓, to recall recent commands.

Correct the value from **`31`** to **`30`** using the **`ce`** command.

```
Stop> ce daysInMonth=30
```

`ce` is an abbreviation of the **`CExpression`** command. This can be used on its own to view the value of an expression, or with a modifier, such as **`=`** in this case, to change the value of an expression.

Use the **`go`** command to pass through the while loop until the output displays the date **`2009 12  3`**

```
Stop> go
```

You will need to toggle between the *StdIO* and *Cmd* tabs of the *Output* pane.

Use the **dump** command to view the **date** variable in memory.

```
Stop> dump /w &date
```

The **/w** argument specifies how to display the area of memory, in this case as words. **&date** specifies the address of the **date** variable.

Note how the successive words in memory correspond to the fields in the **date** struct.

Use the step command to execute the next two instructions.

```
Stop> step
Stop> step
```

Use the **dump** command to view the **date** variable in memory again.

```
Stop> dump /w &date
```

Note how the value of **date.day** has been incremented.

Remove the breakpoint on line **58** and resume program execution.

```
Stop> clear
Stop> go
```

The program terminates normally.

Exercise 2.8 – Using include files in RVD

In this exercise we will see how multiple commands can be combined in an *include command file* to control execution within the debugger.

Consider the file **`month.inc`** found in **`c:\armprac\intro\session2`**:

```
break #40
go
print daysInMonth
print date.day
print date.month
print date.year
clear
break #58
go
print daysInMonth
ce daysInMonth=30
go
go
go
dump /w &date
step
step
dump /w &date
clear
go
```

The file consists of a simple selection of commands which will perform the same task that was performed in the previous exercise.

	Reload the image into the debugger.

	Invoke the include file by selecting *Tools → Include Commands from File* from the menu.

	Use the *Select File* dialog to locate and open the file **`c:\armprac\intro\session2\month.inc`**

	Enter the date **`2009 11 30`** in the *Console* window when prompted.

When the program has terminated use the *Cmd* tab to view the values of the variables displayed by the script file.

	Check the output is correct then quit the debugger to finish the exercise.

Section 2 - Review

We have seen how the ARM Workbench IDE can be used to:

- Create source files and projects
- Invoke the compiler and linker to generate executable images
- Automatically open files for editing from a project, or a compilation warning or error message
- Invoke the compiler and linker to generate executable images
- Automatically open files for editing from a project, or a compilation warning or error message

We have seen how RVD can be used to:

- Control and modify execution of code
- View and modify locals, globals and memory
- View and modify the contents of registers
- Accept commands via the CLI or from a script file to automate debugging

A complete range of debugging facilities is available within RVD. Consult the online documentation for complete information.