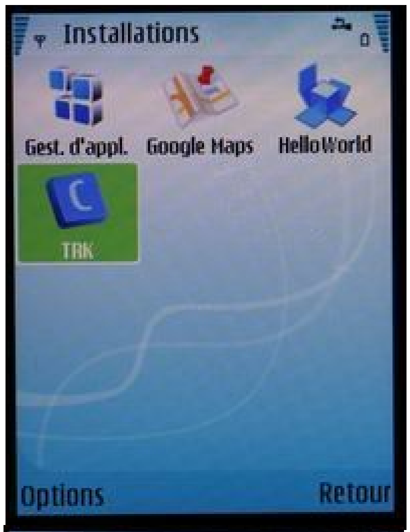
Debugging Symbian applications with IDA Pro

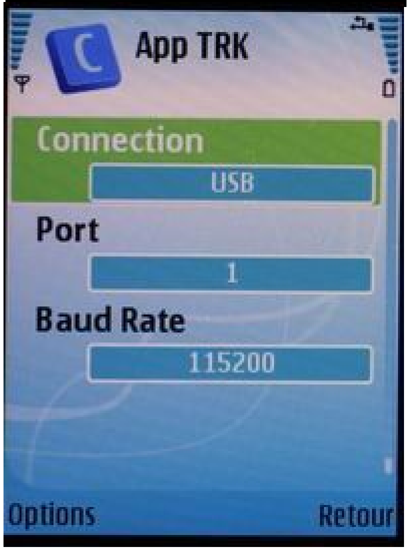
Starting from version 5.3 IDA adds the Symbian platform as a debugger target. User applications and DLLs can be debugged with IDA Pro Standard Edition under MS Windows.

Before we debug any application, we have to install the MetroTRK agent for the device. It can be downloaded from this web site: http://tools.ext.nokia.com/agents

We install the MetroTRK .sisx file on the phone. This icon should appear in the applications folder:



By default MetroTRK is configured to connect over Bluetooth. For a USB connection, the settings must be changed:

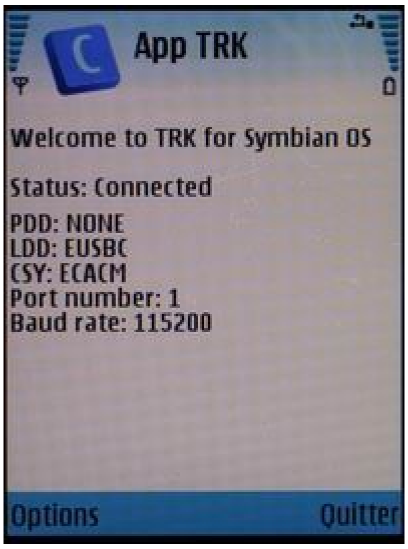

App TRK

Connection
USB

Port
1

Baud Rate
115200

If everything done correctly, upon the « connect » command we will see the following screen:


App TRK

Welcome to TRK for Symbian OS

Status: Connected

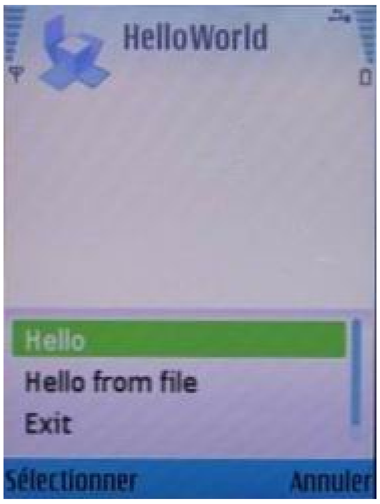
PDD: NONE
LDD: EUSBC
CSY: ECACM
Port number: 1
Baud rate: 115200

We are ready to debug Symbian applications!

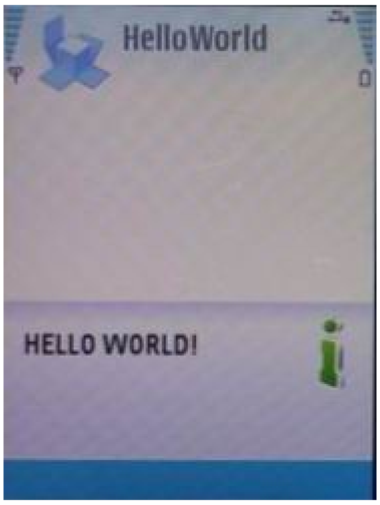
Let's start with a simple HelloWorld application. It can be downloaded from this link: http://www.hex-rays.com/idapro/epoc/helloworldbasic.sis It is not signed, so you might need to change the installation options before installing it. The following steps disable the signature verification:

1. open the “Application Manager”
2. click the left soft key to open the options menu
3. from the list choose the point “Settings”
4. switch the option “Software Installation” to “All”
5. switch the option “Online certif. check” to “Off”

The sample application has the following look:

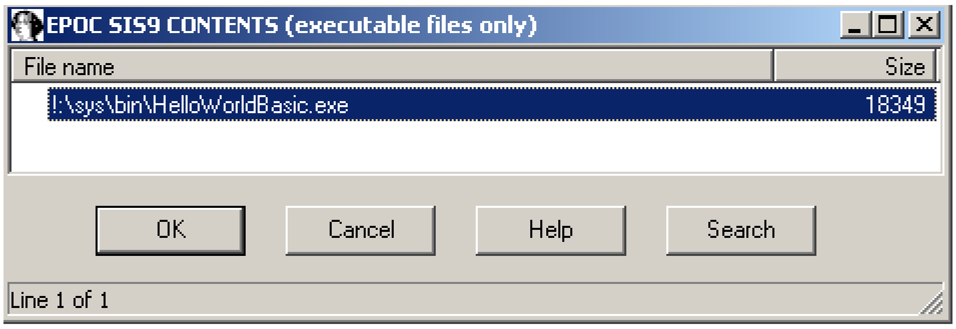


Upon a menu selection it will display a trivial message:



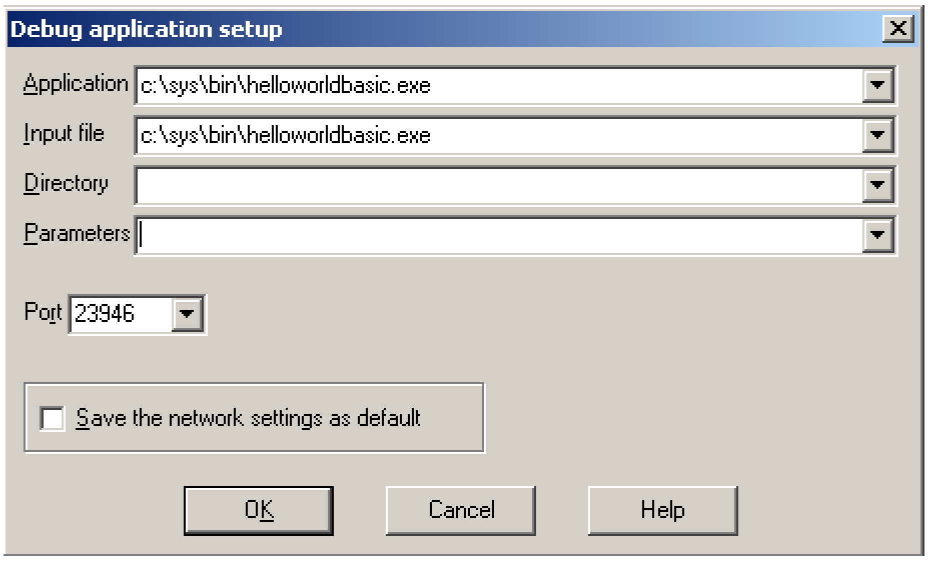
HELLO WORLD!

Now let's create a database for the application and debug it. IDA can parse .SIS files and extract binary files, so we just accept the only file from the list:


EPOC SIS9 CONTENTS (executable files only)

File name	Size
!:\sys\bin\HelloWorldBasic.exe	18349

We have to tell IDA where the application is located on the device and how to connect to it. This is done in the Debugger, Process options dialog box:


Debug application setup

Application	c:\sys\bin\helloworldbasic.exe
Input file	c:\sys\bin\helloworldbasic.exe
Directory	
Parameters	
Port	23946

☐ Save the network settings as default

Since we are debugging an application, the application and input file fields are the same (they are different if we debug DLLs). Usually Symbian applications are installed on c:\sys\bin, but if your device has memory extension cards, the device letter can be different.

IDA uses a serial port connection to talk to Symbian devices. The port number is specified in the « port » input field. The default number proposed by IDA (23946) has a special meaning. It directs IDA to enumerate all serial ports and find the one with the word « Nokia » in its description[1]. You are free to specify a fixed serial port numbere in this field as well.

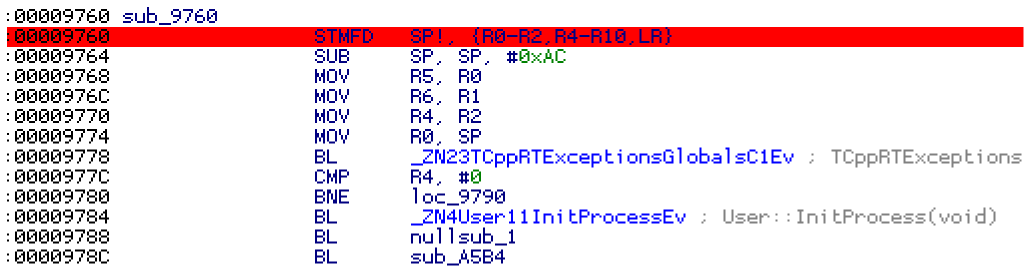
This completes the initial setup. We are ready to press F9 and run the application. However, it is really difficult to suspend a Symbian application once it runs, so we better set some breakpoints before launching it[2]. We can use only software breakpoints, hardware breakpoints are not available.

The following function is a good candidate because it will be executed at the initialization time.

1 If you want to use the « automatic port » feature of IDA for a phone from another vendor, you will need to define the EPOC_PORT_DESCRIPTION environment variable with the desired substring.

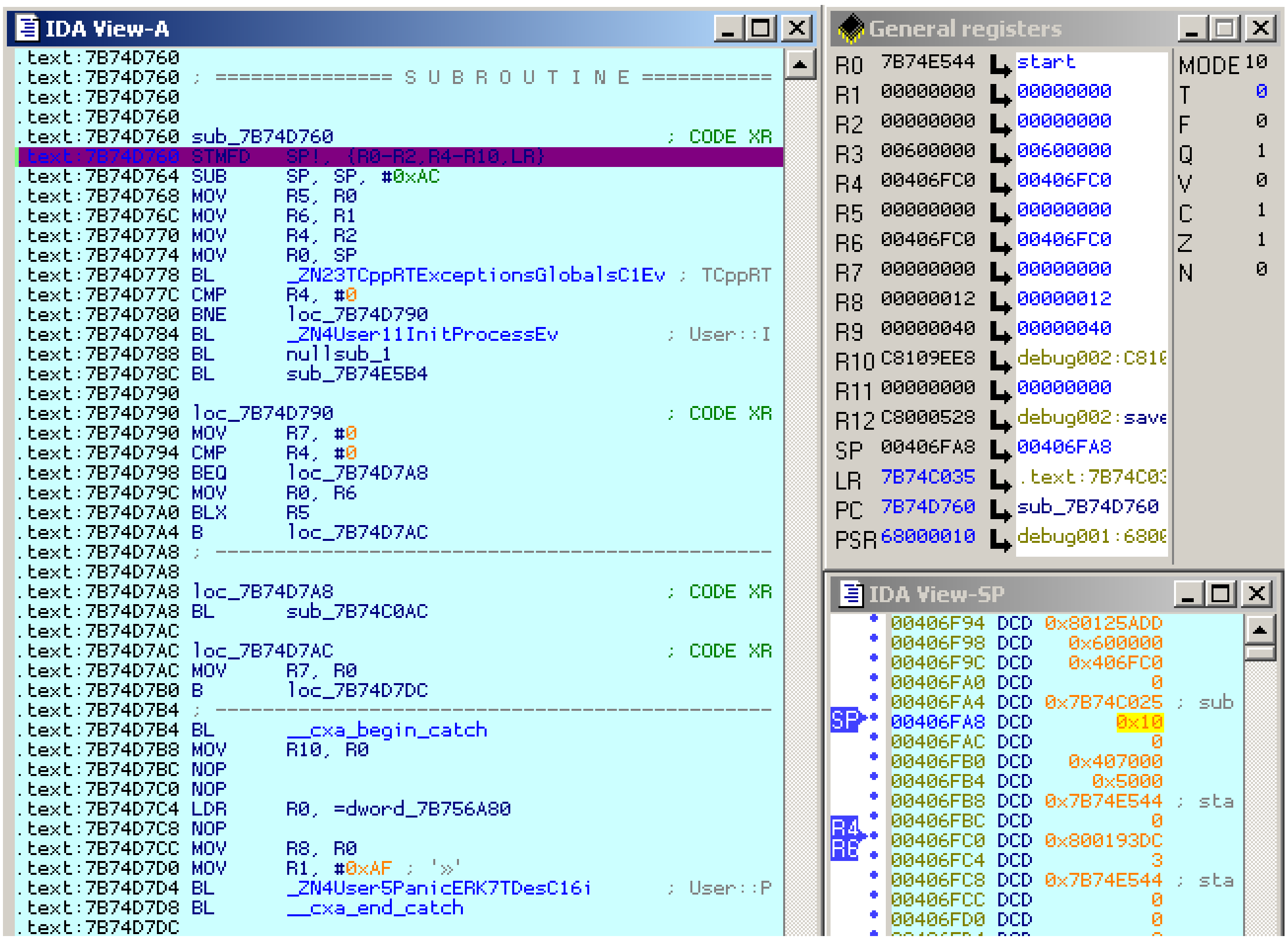
2 The debugged application stops somewhere in the kernel upon « pause ». MetroTRK does not allow to single step in the kernel area. Also, the kernel code is invisible. So, we may suspend the application but we will be in the middle of nowhere.

Pressing F2 adds the breakpoint:



```
:00009760 sub_9760
:00009760                    STMFD   SP!, {R0-R2,R4-R10,LR}
:00009764                    SUB     SP, SP, #0xAC
:00009768                    MOV     R5, R0
:0000976C                    MOV     R6, R1
:00009770                    MOV     R4, R2
:00009774                    MOV     R0, SP
:00009778                    BL      _ZN23TCppRTExceptionsGlobalsC1Ev ; TCppRTExceptions
:0000977C                    CMP     R4, #0
:00009780                    BNE     loc_9790
:00009784                    BL      _ZN4User11InitProcessEv ; User::InitProcess(void)
:00009788                    BL      nullsub_1
:0000978C                    BL      sub_A5B4
```

We are ready to run the application, so we press F9. If everything goes well, IDA connects to the device, launches the program, and switches to the debugger desktop (this one is custom made):



```
.text:7B74D760
.text:7B74D760 ; =============== S U B R O U T I N E ===========
.text:7B74D760
.text:7B74D760
.text:7B74D760 sub_7B74D760                              ; CODE XR
.text:7B74D760 STMFD    SP!, {R0-R2,R4-R10,LR}
.text:7B74D764 SUB      SP, SP, #0xAC
.text:7B74D768 MOV      R5, R0
.text:7B74D76C MOV      R6, R1
.text:7B74D770 MOV      R4, R2
.text:7B74D774 MOV      R0, SP
.text:7B74D778 BL       _ZN23TCppRTExceptionsGlobalsC1Ev ; TCppRT
.text:7B74D77C CMP      R4, #0
.text:7B74D780 BNE      loc_7B74D790
.text:7B74D784 BL       _ZN4User11InitProcessEv          ; User::I
.text:7B74D788 BL       nullsub_1
.text:7B74D78C BL       sub_7B74E5B4
.text:7B74D790
.text:7B74D790 loc_7B74D790                              ; CODE XR
.text:7B74D790 MOV      R7, #0
.text:7B74D794 CMP      R4, #0
.text:7B74D798 BEQ      loc_7B74D7A8
.text:7B74D79C MOV      R0, R6
.text:7B74D7A0 BLX      R5
.text:7B74D7A4 B        loc_7B74D7AC
.text:7B74D7A8 ; ---------------------------------------------
.text:7B74D7A8
.text:7B74D7A8 loc_7B74D7A8                              ; CODE XR
.text:7B74D7A8 BL       sub_7B74C0AC
.text:7B74D7AC
.text:7B74D7AC loc_7B74D7AC                              ; CODE XR
.text:7B74D7AC MOV      R7, R0
.text:7B74D7B0 B        loc_7B74D7DC
.text:7B74D7B4 ; ---------------------------------------------
.text:7B74D7B4 BL       __cxa_begin_catch
.text:7B74D7B8 MOV      R10, R0
.text:7B74D7BC NOP
.text:7B74D7C0 NOP
.text:7B74D7C4 LDR      R0, =dword_7B756A80
.text:7B74D7C8 NOP
.text:7B74D7CC MOV      R8, R0
.text:7B74D7D0 MOV      R1, #0xAF ; '»'
.text:7B74D7D4 BL       _ZN4User5PanicERK7TDesC16i       ; User::P
.text:7B74D7D8 BL       __cxa_end_catch
.text:7B74D7DC
```

```
R0  7B74E544  start          MODE 10
R1  00000000  00000000       T     0
R2  00000000  00000000       F     0
R3  00600000  00600000       Q     1
R4  00406FC0  00406FC0       V     0
R5  00000000  00000000       C     1
R6  00406FC0  00406FC0       Z     1
R7  00000000  00000000       N     0
R8  00000012  00000012
R9  00000040  00000040
R10 C8109EE8  debug002:C81
R11 00000000  00000000
R12 C8000528  debug002:save
SP  00406FA8  00406FA8
LR  7B74C035  .text:7B74C03
PC  7B74D760  sub_7B74D760
PSR 68000010  debug001:680
```

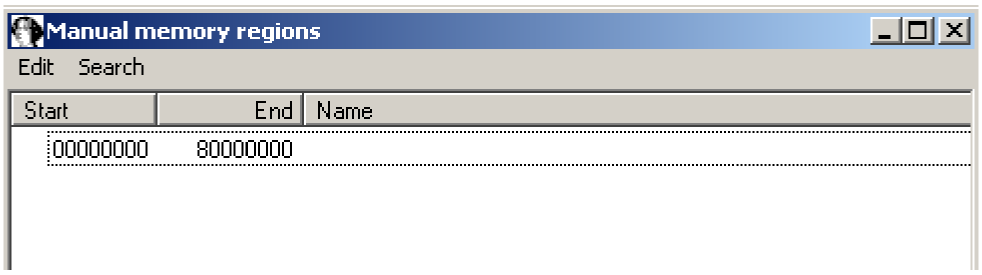
```
      00406F94 DCD 0x80125ADD
      00406F98 DCD   0x600000
      00406F9C DCD   0x406FC0
      00406FA0 DCD          0
      00406FA4 DCD 0x7B74C025 ; sub
SP    00406FA8 DCD       0x10
      00406FAC DCD          0
      00406FB0 DCD   0x407000
      00406FB4 DCD     0x5000
      00406FB8 DCD 0x7B74E544 ; sta
      00406FBC DCD          0
R4 R6 00406FC0 DCD 0x800193DC
      00406FC4 DCD          3
      00406FC8 DCD 0x7B74E544 ; sta
      00406FCC DCD          0
      00406FD0 DCD          0
```

Please note that the addresses have been changed. This happens because applications rarely load to the zero address. The system relocates them somewhere else. IDA detects this and rebases the database. Usually rebasing is transparent and straightforward but if you notice that your code disappeared after F9, then you will need to locate it in the memory and manually rebase the database (Edit, Segments, Rebase program).

By default IDA is able to display addresses up to 0x8000000. If your device rebases applications to higher addresses, then you will need to modify the memory layout. It can be done with the Debugger, Manual memory regions.menu item[3].

3 MetroTRK provides no information on the memory layout, so it must be done manually. IDA automatically determines the memory layout for most platforms.

The following list enumerates the defined memory regions:



Start	End	Name
00000000	80000000	

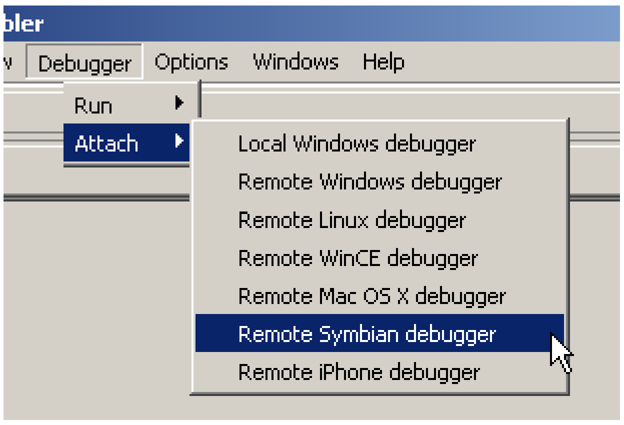
If you do not see anything meaningful on the screen after launching the application, try to increase the upper limit of the defined memory region to 0xFFFFFFFE. You are free to manipulate memory regions as you wish. IDA will use this information to create the program segmentation.

Once we suspended the process, we can use all regular debugger commands like single step, step over, etc. All usual IDA commands are available. You may write scripts and automate things with plugins. However, the following limitations exist:

- The TRK module refuses to single step into system area, we can not do anything with it. This means that tracing will go up to a system call and then stop because of a failed single step command
- Hardware data breakpoints are not supported
- TRK can not properly handle an exception which happens on an instruction with active breakpoint. The exception is generated but it is impossible to resume the application.
- There is no notion of 'handled/unhandled exception' in TRK, so modifying this setting in the exception definitions will not change anything.

Attaching to running applications

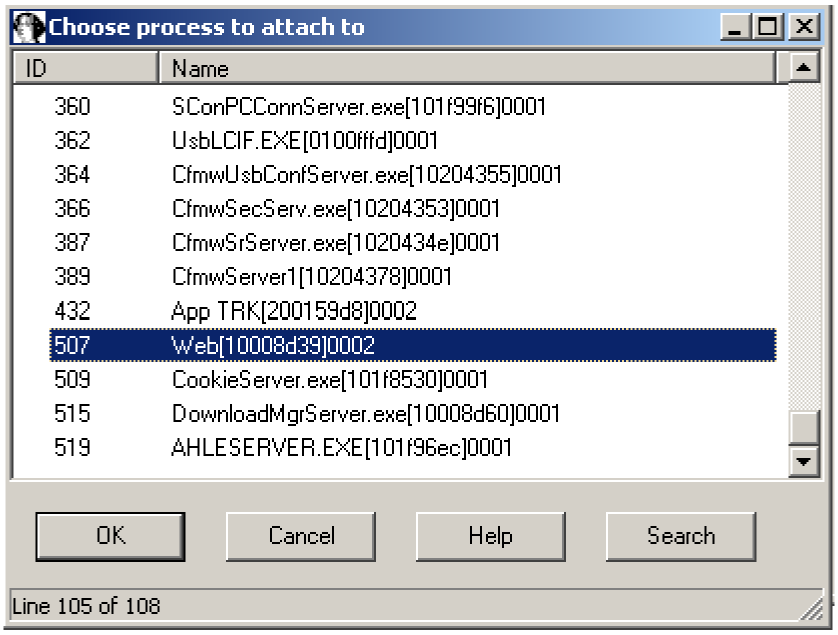
IDA can also attach to running applications. It can be done with an active database or without it. To attach to a process without a database, select the following menu item:



(You can also run a new application using the Debugger, Run menu.)

If you have an open database, select Debugger, Attach to process.

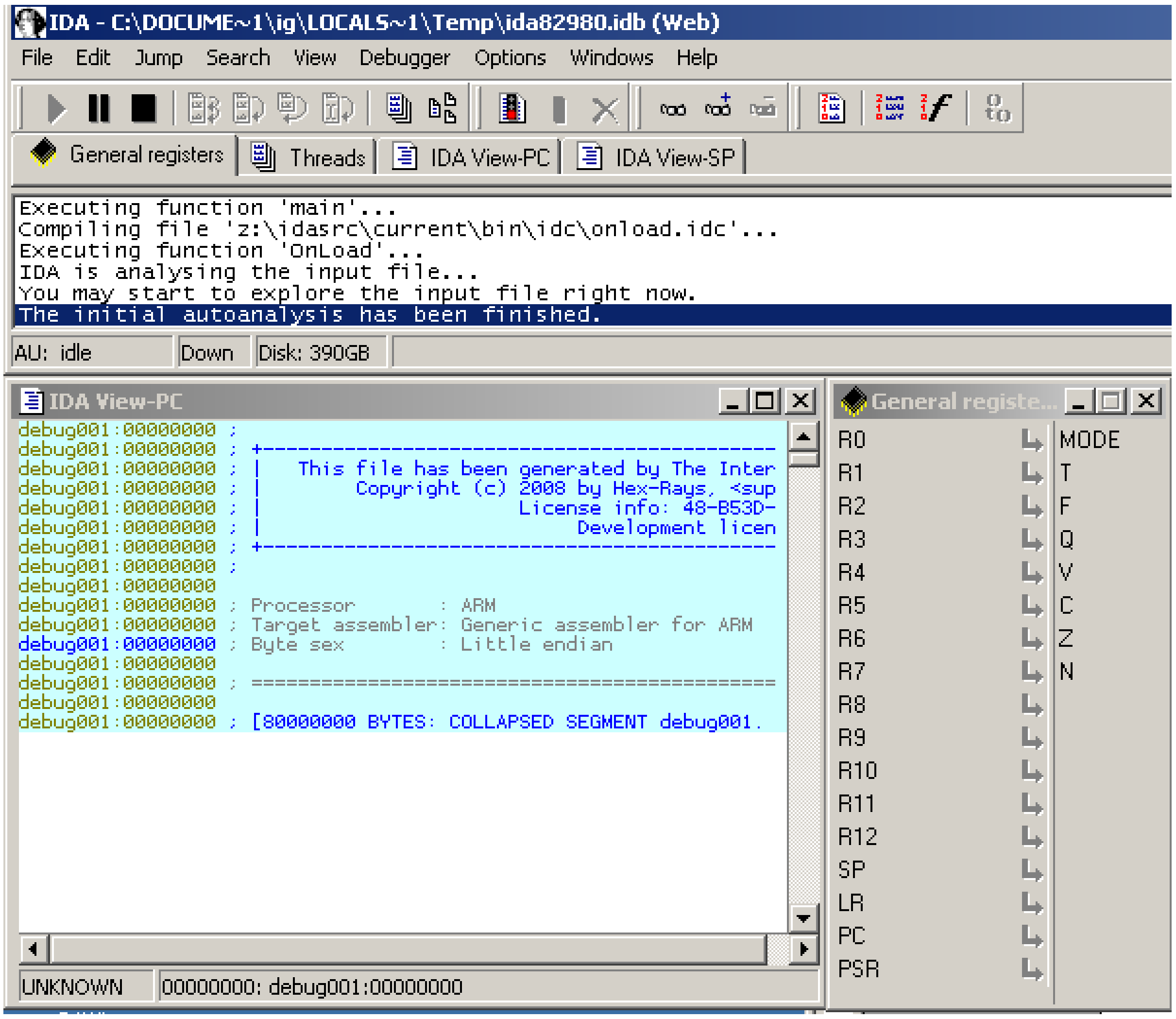
In both cases the following list will appear:


Choose process to attach to

ID	Name
360	SConPCConnServer.exe[101f99f6]0001
362	UsbLCIF.EXE[0100fffd]0001
364	CfmwUsbConfServer.exe[10204355]0001
366	CfmwSecServ.exe[10204353]0001
387	CfmwSrServer.exe[1020434e]0001
389	CfmwServer1[10204378]0001
432	App TRK[200159d8]0002
507	Web[10008d39]0002
509	CookieServer.exe[101f8530]0001
515	DownloadMgrServer.exe[10008d60]0001
519	AHLESERVER.EXE[101f96ec]0001

Line 105 of 108

We have a web browser running on the phone, so let's attach to it and debug it. After clicking ok, IDA attaches to the process and switches the debugger desktop:


IDA - C:\DOCUME~1\ig\LOCALS~1\Temp\ida82980.idb (Web)

```
Executing function 'main'...
Compiling file 'z:\idasrc\current\bin\idc\onload.idc'...
Executing function 'OnLoad'...
IDA is analysing the input file...
You may start to explore the input file right now.
The initial autoanalysis has been finished.
```

AU: idle   Down   Disk: 390GB

IDA View-PC

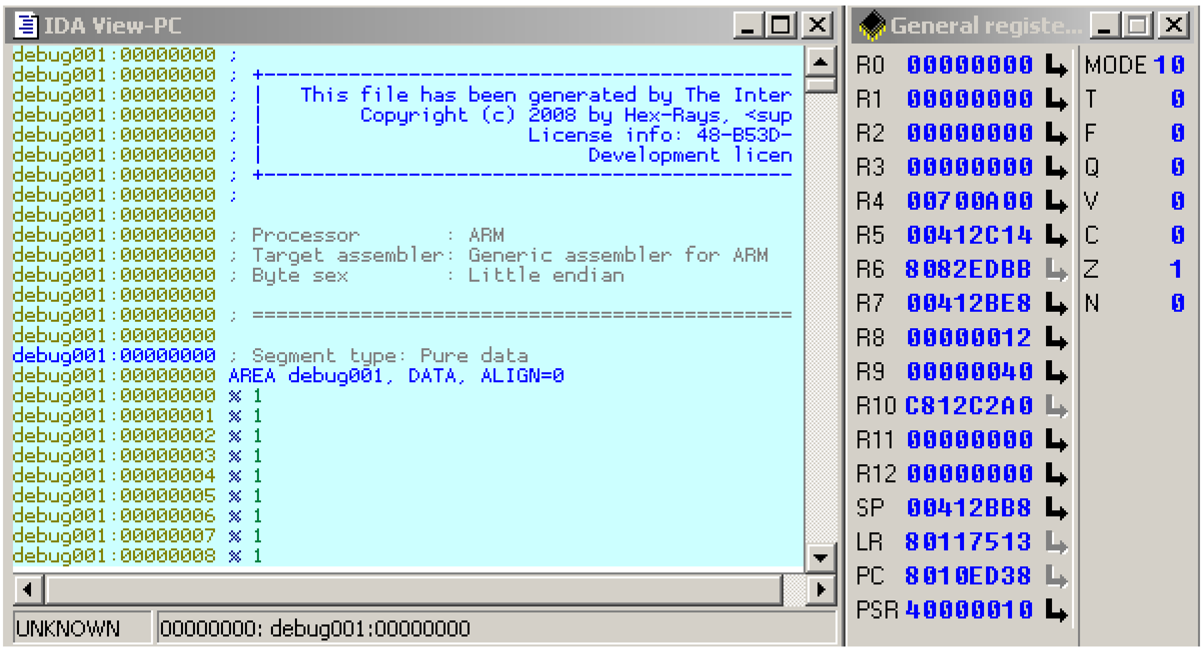
```
debug001:00000000 ;
debug001:00000000 ; +-------------------------------------------
debug001:00000000 ; |   This file has been generated by The Inter
debug001:00000000 ; |      Copyright (c) 2008 by Hex-Rays, <sup
debug001:00000000 ; |                     License info: 48-B53D-
debug001:00000000 ; |                            Development licen
debug001:00000000 ; +-------------------------------------------
debug001:00000000 ;
debug001:00000000
debug001:00000000 ; Processor       : ARM
debug001:00000000 ; Target assembler: Generic assembler for ARM
debug001:00000000 ; Byte sex        : Little endian
debug001:00000000
debug001:00000000 ; ============================================
debug001:00000000
debug001:00000000 ; [80000000 BYTES: COLLAPSED SEGMENT debug001.
```

UNKNOWN   00000000: debug001:00000000

General registe...

```
R0      MODE
R1      T
R2      F
R3      Q
R4      V
R5      C
R6      Z
R7      N
R8
R9
R10
R11
R12
SP
LR
PC
PSR
```

Please note that we attached to the process but it did not suspend. It is still running, we do not have any register values, no code is displayed on the disassembly view. This is normal. If we suspend it by clicking on the pause ⏸ button[4], we will be in the middle of nowhere (in the kernel).



IDA View-PC

```
debug001:00000000 ;
debug001:00000000 ; +-------------------------------------------------
debug001:00000000 ; |    This file has been generated by The Inter
debug001:00000000 ; |          Copyright (c) 2008 by Hex-Rays, <sup
debug001:00000000 ; |                         License info: 48-B53D-
debug001:00000000 ; |                              Development licen
debug001:00000000 ; +-------------------------------------------------
debug001:00000000 ;
debug001:00000000
debug001:00000000 ; Processor       : ARM
debug001:00000000 ; Target assembler: Generic assembler for ARM
debug001:00000000 ; Byte sex        : Little endian
debug001:00000000
debug001:00000000 ; ===============================================
debug001:00000000
debug001:00000000 ; Segment type: Pure data
debug001:00000000 AREA debug001, DATA, ALIGN=0
debug001:00000000 % 1
debug001:00000001 % 1
debug001:00000002 % 1
debug001:00000003 % 1
debug001:00000004 % 1
debug001:00000005 % 1
debug001:00000006 % 1
debug001:00000007 % 1
debug001:00000008 % 1
```

```
UNKNOWN    00000000: debug001:00000000
```

General registe...

```
R0  00000000   MODE 10
R1  00000000   T     0
R2  00000000   F     0
R3  00000000   Q     0
R4  00700A00   V     0
R5  00412C14   C     0
R6  8082EDBB   Z     1
R7  00412BE8   N     0
R8  00000012
R9  00000040
R10 C812C2A0
R11 00000000
R12 00000000
SP  00412BB8
LR  80117513
PC  8010ED38
PSR 40000010
```

Unfortunately, MetroTRK gives us no information on the memory layout, exported functions, and other niceties like other debugger modules do. We will have to find out the code ourselves. First ,we need to locate accessible memory. You may use the module list (Debugger, Debugger windows, Module list) to get the list of loaded DLLs. You may also run a script like this:

```
auto ea;
for (ea=0x1000; ea < 0x8000000; ea=ea+0x10000)
{
 if ( Byte(ea) != 0xFF )
   Message("%a: readable memory\n", ea);
 // print something from time to time
 if ( (ea % 0x1000000) == 0x1000 )
   Message("%a: running\n", ea);
}
```

to determine readable memory.

Again, this is not enough, we also need to locate the code and put some breakpoints there. Only after that we can intercept control and examine registers, single step, etc.

The following script can be used to convert the import table into code and add a breakpoint for each imported function. You will need to find the import table yourself, though:

```
auto ea,i;
i=0;
for(ea=here; Dword(ea)==0xE51FF004; ea=ea+8)
{
  MakeCode(ea);
  AddBpt(ea);
}
```

4 In fact, we always need to suspend the attached application at least once because of a bug in IDA v5.3. If we do not suspend the application, IDA will not suspend on breakpoints.

Troubleshooting

If anything does not work, use -z10000 command line switch to get more information on the screen. In particular, communication with the TRK module will be displayed in detail.

If you do not see any code after launching the application, check out the memory region setup (Debugger, Manual memory regions). MetroTRK provides no information on the memory layout and it must be congifured manually.

If one big memory region is defined for the entire application, IDA might delete code at each pause. You can fix this by disabling the stack reconstruction in the Debugger, Debugger options menu.