

Contents

- [1 Note](#)
- [2 Preparations](#)
- [3 Settings](#)
- [4 Connection](#)
- [5 Debug](#)

Note

The debugging setup process in [Carbide.c++ 2.0](#) has been improved and differs from the process documented here. For the latest information on using the debugging features in Carbide.c++ please see the [Debugging projects](#) section of the online Carbide.c++ help.

Preparations

Various editions of Carbide.c++ support emulator debugging, on-device debugging, and external debugging with JTAG. On-device debugging is a very important way for developers to run the applications on device hardware and locate issues in the code that may not be identified when using the SDK emulators.

Carbide.c++ provides two levels of on-device debugging, application debugging and system debugging. System debugging is integrated into the OEM edition of Carbide. As application debugging is used more often by third party developers, we will concentrate on on-device application debugging here.

To debug applications on device, the on-device agent is required. It is called Application TRK. The agent's installation file is included with Carbide.c++ Professional edition. It can be found in [your carbide installation path]\plugins\com.nokia.carbide.trk.support_1.3.0.020\trk\s60\.

Below are the steps to debug an S60 application. As we know, a S60 application can be a standalone GUI program or a GUI program plus engine. Application TRK is capable of debugging both types of program.

It is assumed that the Application TRK has been installed on the target device. The next step is to connect the target device with the PC to used for debugging. The connection can be setup via using a USB or Bluetooth connection. Bluetooth is the case we use here.

You need to know the port number the Bluetooth connection is using on the debugging PC. Check this in the Bluetooth configuration (usually found in Control Panel) and write down the port number. It will be used in the Carbide.c++ debugging configuration. See [\[\[Figure 1100_bluetoothport.png\]\]](#).

To illustrate debugging import the example GUIEngine project found in S60 3rd Edition SDK supporting Feature Pack 2.

Settings

The debug window is opened first, then the developers need to feed the proper settings accordingly.

Figure 2 and **Figure 3** above show how to open and start a new debugging setting. We set up the debugging configuration for Symbian OS Application TRK.

In the following three diagrams **Figure 4**, **Figure 5**, and **Figure 6**, the red arrows and under lines indicated what we have done to configure the debugging settings for GUIEngine app.

The port number is used here, make it as same as the one we got from bluetooth configuration just now.

The app needs to be packaged and signed in advance. for instruction of Symbian signed, please refer to [Symbian signed](#)

Connection

Back to target S60 device, start the Application TRK, find and connect to the debugging PC. If the connection is good, the developers shall see on the screen of mobile phone

```
Welcome to TRK for Symbian OS
```

```
Status: Connected
```

```
BT Dev Name: (PC's name)
```

```
BT Port number: 1
```

If some problems happened, check the regular bluetooth connection settings and bluetooth port number.

After the connection between the target S60 device and the debugging PC is set, press button Run in debug window, Carbide starts downloading the app, here GUIEngine to target device and debugging session.

If all steps are right, the developers should see **Figure 7**.

if not, **Figure 8**,

Debug

Meanwhile, Carbide IDE makes the debug view (**Figure 9**) visible.

As a sequence of operations, we set the break points in both GUI and engine part, continue the debug session, do debugging regularly (shown in **Figure 10 - 14**).

As one reminding, sometimes the break points are set not in right place, so we get unresolved break points. We can also disable break points on purpose (**Figure 15**).

All the figures here

Demonstration_of_Carbidе.c++_1.3_on-device_app_debug



Figure 1. Bluetooth port number



Figure 2. Open debug window 1



Figure 3. Open debug window 2



Figure 4. Debug window settings 1



Figure 5. Debug window settings 2



Figure 6. Debug window settings 3



Figure 7. Launch successfully



Figure 8. Launch failed



Figure 9. Debug viewk



Figure 10. Set break points



Figure 11. Debugging 1



Figure 12. Debugging 2



Figure 13. Debugging 3



Figure 14. Debugging 4

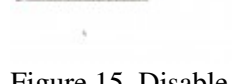


Figure 15. Disable break points