S60 Platform: Thread And Active Objects Example

Version 1.1
November 25, 2005

S60 platform

Legal Notice

Disclaimer

The information in this document is provided “as is,” with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. Furthermore, information provided in this document is preliminary, and may be changed substantially prior to final release. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time, without notice.

License

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein.

Contents

Change History

April 20, 2005	Version 1.0	Initial document release
November 25, 2005	Version 1.1	Compatibility with the updated example checked. Minor terminology update.

1. System Architecture

Figure 1 shows a simplified class diagram of the system. The system is divided into two blocks which show the responsibilities of each thread. The main program is `MainThread`. `ThreadOne` is a worker thread that is created by `MainThread` when the user starts discovering devices. The `ThreadOne` and `MainThread` communication is described in detail in Figure 3.

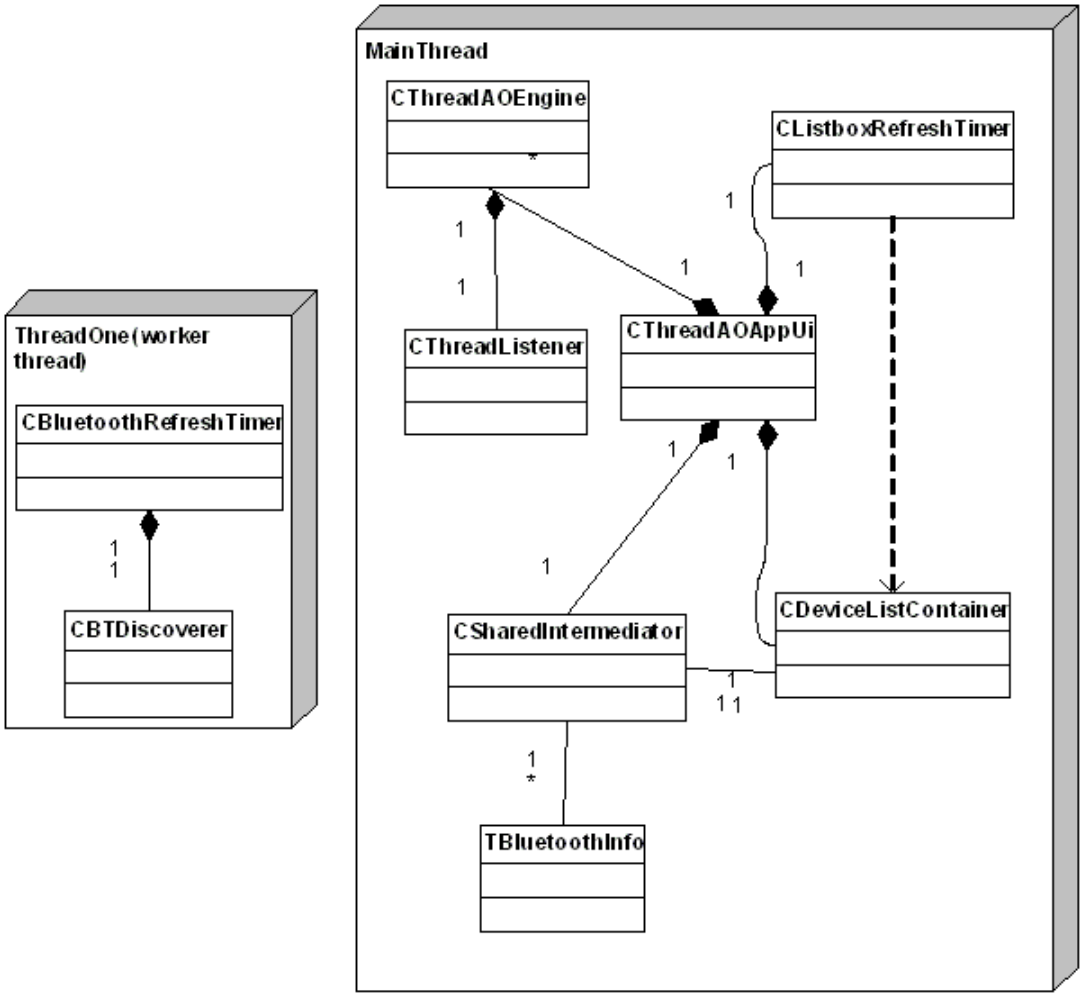

Figure 1: Class diagram of the system. The relationships between `ThreadOne` (worker thread) and `MainThread` are not presented in this diagram.

Class	Main responsibility
CBTDiscoverer	Finds Bluetooth devices. Sends found devices to `CSharedIntermediator`.
CBluetoothRefreshTimer	Counts the time and refreshes the Bluetooth device search when refreshing occurs.
CThreadAOEngine	Creates `ThreadOne` (worker thread).
CThreadListener	Observes `ThreadOne` and ends the main program when the worked thread dies.

Class	Main responsibility
CListboxRefreshTimer	Calls `CDeviceListContainer::HandleItemAdditionL()` periodically because that function cannot be called from `ThreadOne`. This class makes sure that items are shown after `ThreadOne` adds new devices into listbox.
CThreadAOAppUi	Command handling and most of the objects are created inside this class.
CSharedIntermediator	Communication class between `ThreadOne` and `MainThread`. Stores also a list of the found Bluetooth devices (names and addresses). See `TBluetoothInfo`.
TBluetoothInfo	Represents one `BluetoothDevice` that has a name and an address.
CDeviceListContainer	Listbox for the Bluetooth device names which have been found. Asks the chosen Bluetooth device's address from `CSharedIntermediator` and shows the address.

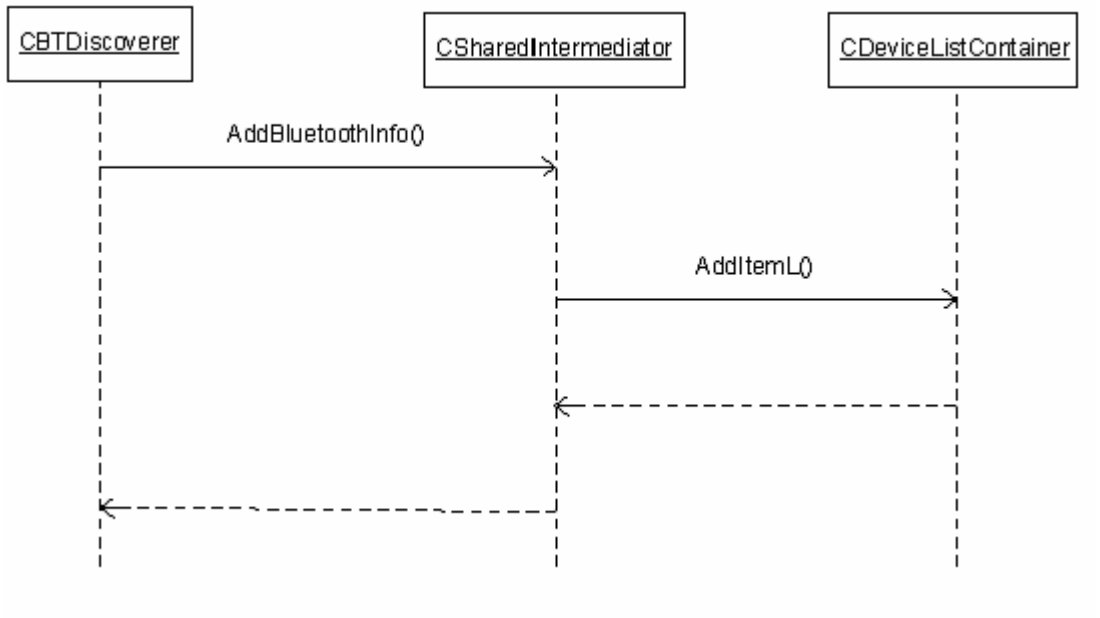

Figure 2: Adding one Bluetooth device name into ListBox

2. Inter-Thread Communication

Communication between `ThreadOne` and `MainThread` is handled using an instance of `CSharedIntermediator`, which is shown in Figure 3. An object is created in `MainThread` and given to `ThreadOne` (`TAny*` parameter of `ThreadFunction`) when the worker thread is created (`CThreadAOEngine::CreateThreadsL()`). `CSharedIntermediator` contains all the pointers and data structures which are needed in inter-thread communication. It is important to understand that the communication is done with the help of only one object. `CThreadListener` is an observer that listens when the `ThreadOne` is killed. This way it is possible to get information of the `ThreadOne` state, even in error situations.

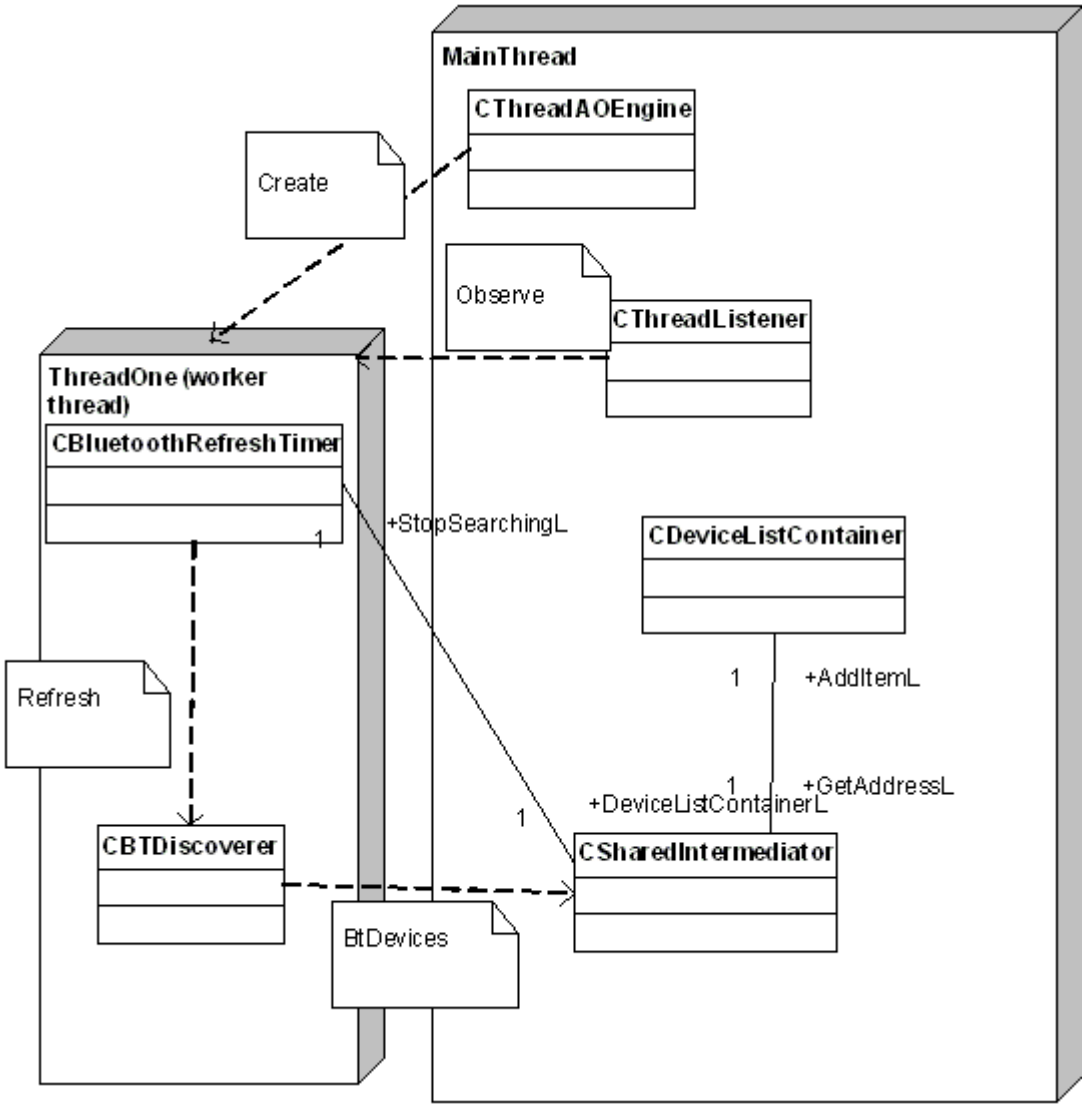

Figure 3: Relationships between `MainThread` and `ThreadOne` (worker thread)

3. Worker Thread Cleanup

Exiting the program is not so trivial, because the worker thread is run down in a controlled way. Exiting starts when the user presses the Exit button. Notification of ending is transmitted to the worked thread. The worked thread polls one `TBool` variable continuously just to see when the main program wants to shut down. Figure 4 shows how `CSharedIntermediator` is updated.

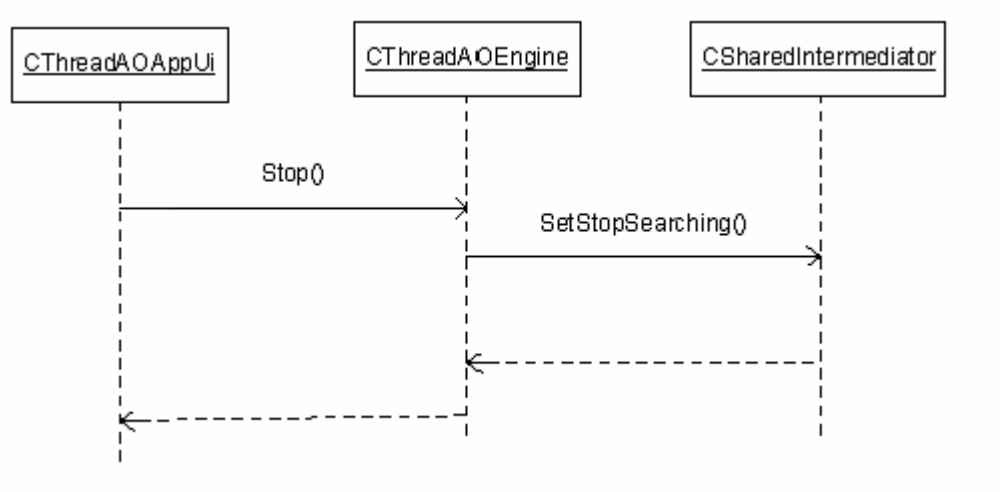

Figure 4: User presses Exit.

Figure 5 describes what happens after `SetStopSearching()` has been called. `CBluetoothRefreshtimer` polls `CSharedIntermediator` whether the worked thread should be closed down with the `StopSearching()` method. If `ETrue` is returned, the worked thread is run down. The rundown includes stopping `CBTDiscoverer`, ending the `CActiveSchedulerWait` loop, and ending `CBluetoothRefreshTimer` itself. Once the `CActiveSchedulerWait` loop is ended, the worker thread dies (`ExecuteThreadOne(TAny *aPtr)` returns 0). After the worker thread dies, `CThreadListener` is notified (`Logon()`) and the main program is exited.

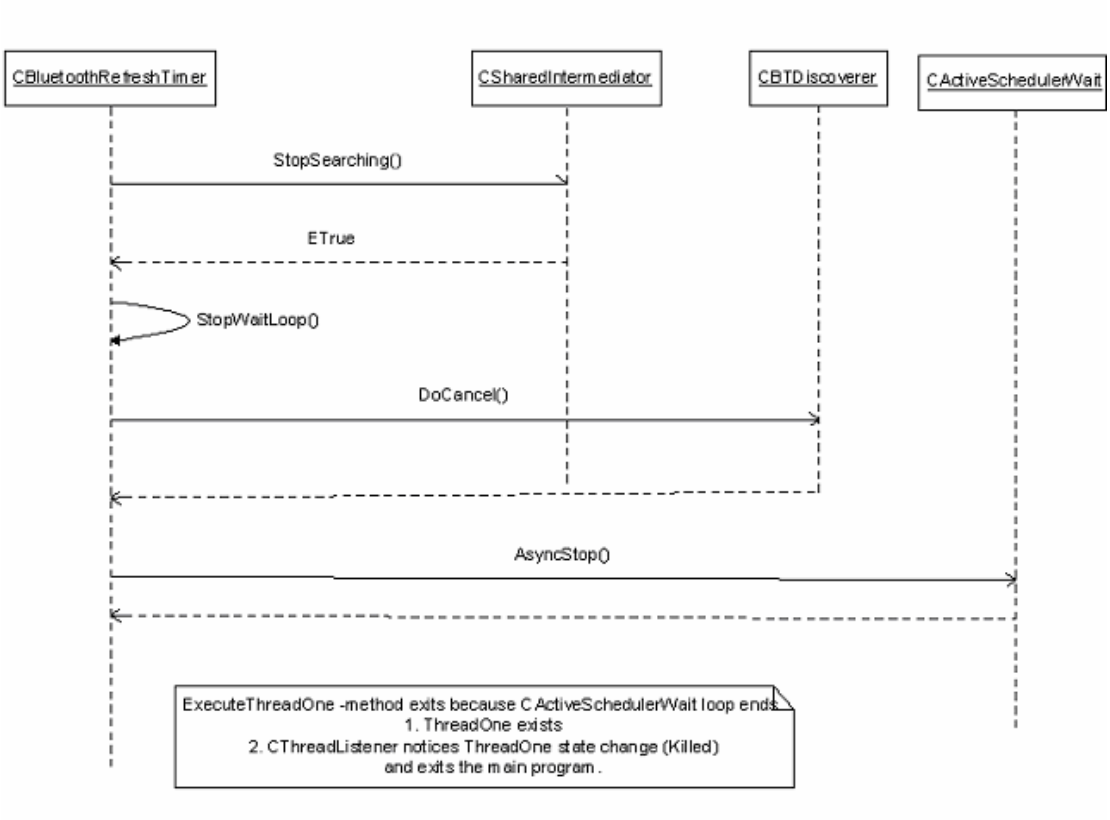

Figure 5: Cleanup after exit

4. Evaluate This Resource

Please spare a moment to help us improve documentation quality and recognize the resources you find most valuable, by rating this resource.