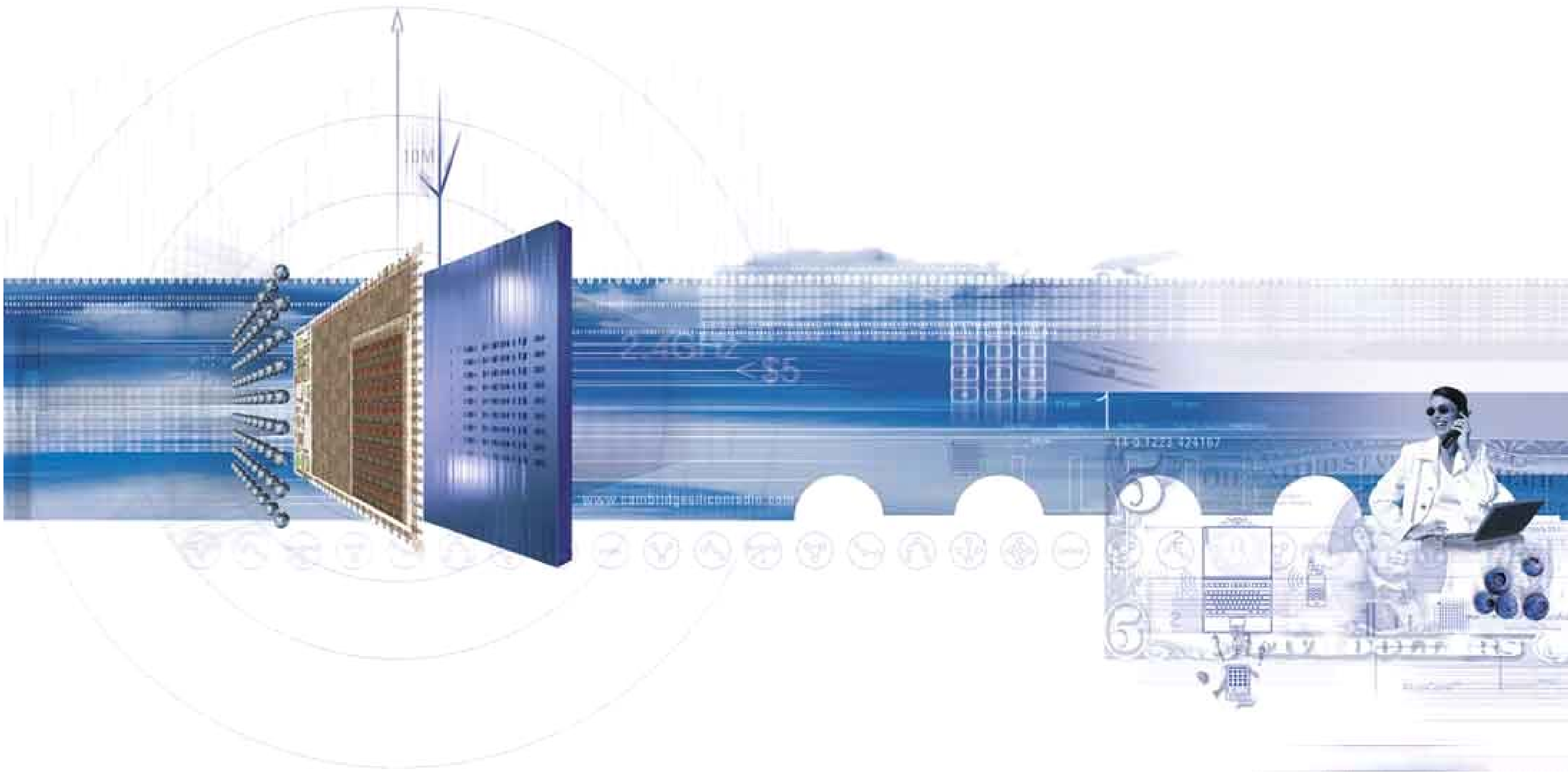
BlueCore™

Windows® CE Driver for BlueCore ROM Devices

Application Note

May 2004



CSR

Cambridge Science Park
Milton Road
Cambridge
CB4 0WH
United Kingdom

Registered in England 3665875

Tel: +44 (0)1223 692000
Fax: +44 (0)1223 692001
www.csr.com

© Cambridge Silicon Radio Limited 2004
This material is subject to CSR's non-disclosure agreement.

Contents

List of Figures

List of Tables

© Cambridge Silicon Radio Limited 2004
This material is subject to CSR's non-disclosure agreement.

1 Introduction

1.1 ROM Devices

CSR produces ROM variants of its **BlueCore™** devices. Each of these variants contains a complete version of the firmware. This eliminates the need for the firmware to be stored in an external flash device, significantly reducing bill of materials (BOM).

BlueCore devices require some device-specific information stored with them, namely the Bluetooth® address, crystal frequency trim value and possibly some information about the radio power. Traditionally this would be stored in the flash device. A list of keys is available in the CSR document Selection of I^2C™ EEPROMs for Use with BlueCore. Some of this information will be unique to the specific BlueCore device itself. Other elements of this information will be used to change the ROM defaults (e.g., host transport or boot mode).

In ROM variants, there is no place within the ROM device for this information to be stored. The information must be stored outside the device. There are two possible mechanisms for sending this extra information to the device.

- Store the device-specific information on an EEPROM accessible over BlueCore's I^2C interface. BlueCore will read this information at its boot time.
- Send down the device-specific information at BlueCore's boot time from some external application on a host microcontroller.

This document deals with the latter of these two mechanisms.

1.2 Application

CSR's BlueCore devices can be used in many applications, including smart phones, headsets, PCs and dongles. Some of these applications use an external stack (e.g., PDAs and PCs), while others require the BlueCore device to contain the entire Bluetooth stack, including the profiles (e.g., headsets and HID devices).

In some instances where an external stack is used, the Bluetooth chip may never move away from the stack, (such as in a PDA or a smart phone). It is, therefore, possible for this external stack to hold the Bluetooth device-specific information and have an application to send down the device specific information from a configuration file.

Unit costs are lowered by keeping the device information with the host operating system; this eliminates the cost of an additional EEPROM from the overall BOM.

1.3 Function

The Bluetooth stacks used are often monolithic or provide no easy hook on which to hang some form of configuration function. However, the Bluetooth stacks all use a serial driver for communication with the BlueCore device. This serial driver has a standard interface, so the configuration module can be inserted between the Bluetooth stack and the serial driver.

This document describes **SerialCSR**, a wrapper library for a Windows® CE (WinCE) serial driver for use with devices containing CSR ROM chips.

The primary function of SerialCSR is to send down configuration information to BlueCore. It does this by intercepting calls to the real serial driver and, when triggered, it reads a .psr file. For each key in the psr file, it sends a command to BlueCore using the BlueCore Command (BCCMD) protocol. SerialCSR utilises "Micro" BlueCore Serial Protocol (µBCSP) to send down the BlueCore commands. ROM chips used in CE devices will tend to have BCSP as their default protocol.

© Cambridge Silicon Radio Limited 2004
This material is subject to CSR's non-disclosure agreement.

The driver must send this information whenever BlueCore initialises, which may be after a suspend, power-up or stack initialisation. The driver will send the contents of the .psr when triggered by the following commands:

- On every CreateFile
- On any FileRead or FileWrite, when immediately following a Suspend or Init

After sending down the configuration information, SerialCSR will send commands to warm-reset BlueCore, so the new PS Keys will take effect. After that, SerialCSR will act transparently, passing through any calls to the underlying serial driver.

By performing these operations, SerialCSR can cope with several types of power cycle where the Bluetooth stack needs to have seamless communication with the BlueCore device.

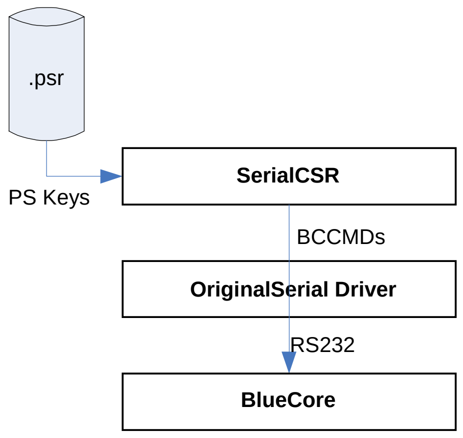


Figure 1.1: SerialCSR Driver Performing a Configuration

Apart from sending down the commands, the SerialCSR driver will pass through all other calls to the underlying serial driver. It does not alter the behaviour of the underlying serial driver in any other way.

The same approach of storing the PS Keys with the operating system and using a ROM chip without EEPROM would also work on alternative platforms such as Palm® and Symbian®, as well as desktop operating systems. There are no implementations other than Windows CE available from CSR. Please check with your CSR representative or distributor for more information.

1.4 Development Environment

The development environment used was the Microsoft® Embedded Visual C++ v3.0. The following support matrix summarises the operating system support of current Microsoft Embedded Visual Development Tools.

Note:

This product includes Microsoft Embedded Visual C++ 5.0

Visual Development Tool	Operating System Supported
Embedded Visual C++ v5.0	WinCE.NET devices only
Embedded Visual C++ v4.0	WinCE v4.0 devices only
Embedded Visual C++ v3.0	WinCE v3.0 and earlier only

Table 1.1: Support Matrix for Embedded Visual Tools

© Cambridge Silicon Radio Limited 2004
This material is subject to CSR's non-disclosure agreement.

2 Deliverables and Scope of Additional Work

CSR ships the source code for SerialCSR. The source code (as shipped) is tailored for use with the Widcomm device driver and Bluetooth stack. If the driver is to be used with another stack, some source code modifications will be required.

The source code reads the configuration information from a .psr file. If it is not deemed appropriate to store configuration information in a .psr file, then it is entirely feasible to use the registry or another format. In this case, some source code modifications will be required.

SerialCSR (as shipped) makes certain assumptions about the ROM's default baud rate and UART configuration on the BlueCore device. If these do not match the variant of firmware on the ROM device, then the default values will need to be edited in the source. Using an HCI transport other than BCSP entails extensive modifications to the source code, replacing uBCSP with another transport.

© Cambridge Silicon Radio Limited 2004
This material is subject to CSR's non-disclosure agreement.

3 Overview of Software

3.1 Functional and Data Description

This section describes the overall function of SerialCSR from initialisation through to Persistent Store update. As a shim layer above the serial driver, the information domain within which SerialCSR operates is well defined by the functions it wraps. In a typical stream interface driver, the functions outlined in Table 3.1 are exported for use by the WINAPI. `COM_IOControl()` is typically used as the handler for all direct hardware access and for all access not handled by the other exported functions.

3.1.1 System Architecture

SerialCSR is developed as a wrapper library to any WinCE compliant serial driver. Figure 3.1 outlines the context level architecture within which SerialCSR operates.

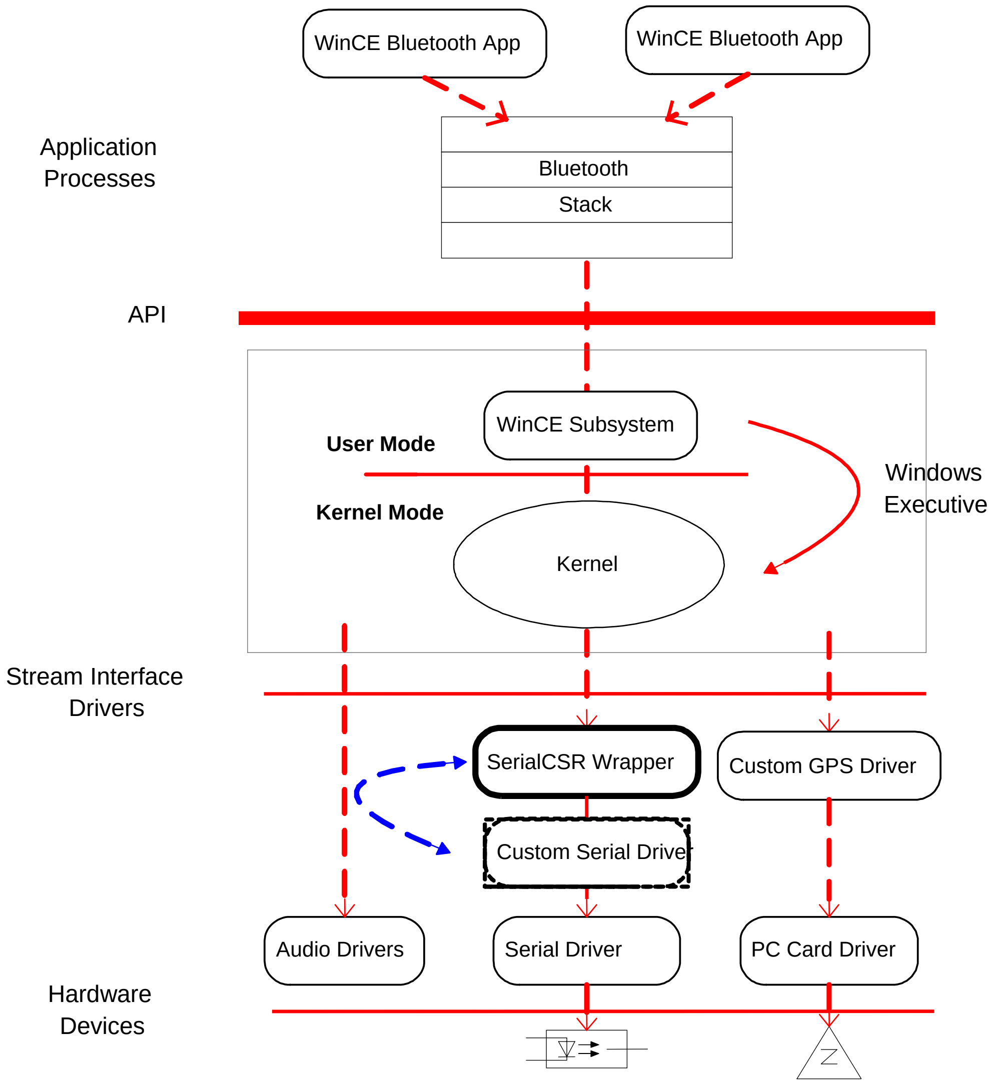


Figure 3.1: SerialCSR Overview

© Cambridge Silicon Radio Limited 2004
This material is subject to CSR's non-disclosure agreement.

Serial drivers used to communicate to the UART on a BlueCore are implemented as stream interface drivers and can be loaded under a number of circumstances.

- Boot Time

  Typically, Bluetooth serial drivers are loaded at boot time by the Device Manager. Registry entries indicate which driver to load.for those drivers.

- Device Detection

  When WinCE detects the connection of some additional hardware, such as a Bluetooth PC card, the appropriate driver is loaded.

- Application Initialisation

  Applications can also ship with their own custom stream interface drivers and load the drivers on the system when the application is run. Such drivers generally sit on top of an existing driver and present the underlying driver's services to the application.

Figure 3.2 outlines the context level architecture within SerialCSR

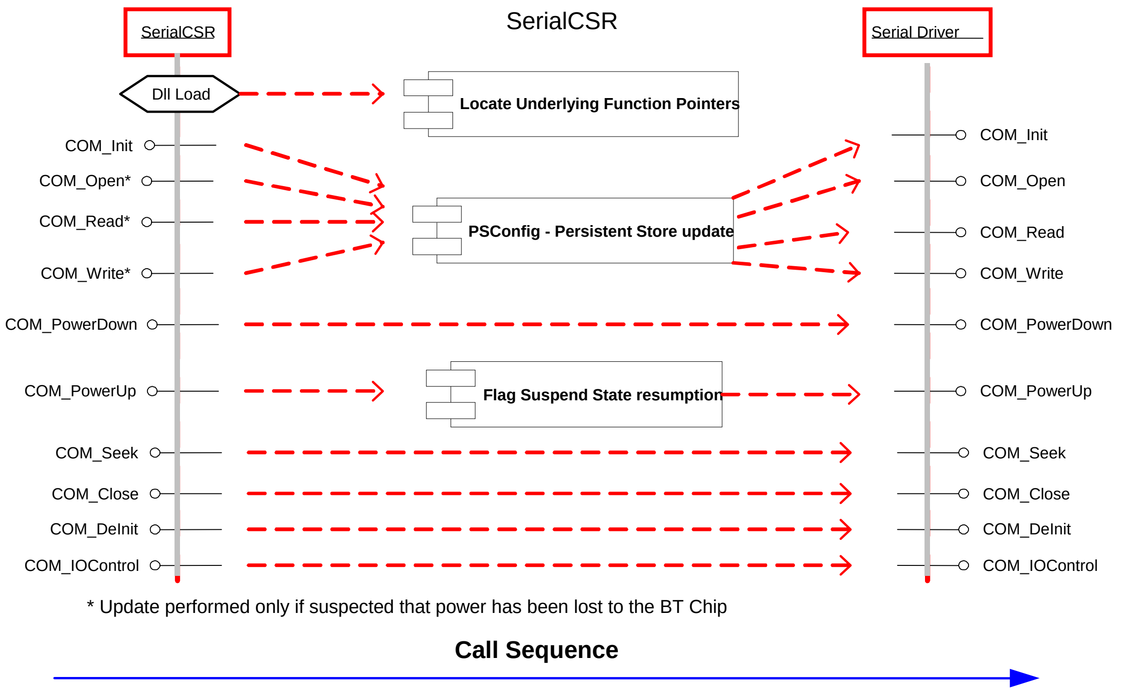


Figure 3.2: Architecture of SerialCSR

3.1.2 Subsystem Overview

All WinCE applications accessing Bluetooth Devices driven through Serial ports make use of standard serial driver windows API calls such as `CreateFile()`, `ReadFile()` and `WriteFile()`. The WinCE Executive marshalls these calls and filters them based on the type of device access requested, the security context of the caller and the resources available to the system. Calls to the API functions such as `CreateFile()`, `ReadFile()` and `WriteFile()` are mapped to functions like `COM_Open()`, `COM_Read()` and `COM_Write()`.

The serial driver provides a standard interface for communicating with a UART. The UART in turn is used to communicate with other devices, both internal to the platform, (e.g. a BlueCore device) or on a separate device, such as a PC. SerialCSR lies between the underlying lower level Bluetooth device’s serial driver and either the NT Executive or an upper level custom driver, depending on the implementation desired by the OEM.

OEMs may change the position of SerialCSR, but that may involve source code modifications based on the APIs to be adapted. The default implementation, however, should be adequate for most implementations.

© Cambridge Silicon Radio Limited 2004
This material is subject to CSR’s non-disclosure agreement.

3.2 Data Description

As supplied, SerialCSR reads a .psr file containing Persistent Store values.

```
// PSKEY_BDADDR
&0001 = 0001 2821 005b 6789
// PSKEY_ANA_FTRIM
&01f6 = 0025
// PSKEY_HOST_INTERFACE
&01f9 = 0001
// PSKEY_UART_BAUD_RATE
&0204 = 01d8
// PSKEY_ANA_FREQ
&01fe = 0004
// PSKEY_UART_CONFIG
&0205 = 0006
```

Persistent Store Keys (PS Keys) are identified by a single 16-bit word. They are delimited from values by the = symbol; the value can be up to 64 words in length. SerialCSR reads the data from the specified file and loads the values into a BCCMD PDU, which is sent to the chip over BCCMD. Serial CSR will skip comments in the .psr file.

© Cambridge Silicon Radio Limited 2004
This material is subject to CSR's non-disclosure agreement.

3.2.1 Interface Description

The WinCE API defines the following interfaces for higher-level access.

Function	Description
`COM_Close`	Closes the serial device. It is called in response to an application's call to the `CloseHandle` function.
`COM_Deinit`	De-initialises the serial port.
`COM_Init`	Initialises the serial device.
`COM_IOControl`	Implements the serial port's I/O control routine. It is called by serial port functions such as `GetComState`, which is a wrapper around this function.
`COM_Open`	Initialises the serial port driver. This function is exported to applications through `CreateFile`.
`COM_PowerDown`	Indicates to the serial port driver that the platform is about to go into suspend mode.
`COM_PowerUp`	Indicates to the serial port driver that the platform is resuming from suspend mode.
`COM_Read`	Enables an application to receive characters from the serial port. This function is exported to users through the `ReadFile` function. This function must obey time-out values set for the serial port.
`COM_Write`	Enables an application to transmit bytes to the serial port. This function is exported to users through the `WriteFile` function. This function must obey flow-control and time-out values.

Table 3.1: Exported Functions from a Typical Serial Driver Implementation

Aside from the configuration function, SerialCSR passes all calls from the Windows Executive above to the serial driver below.

Table 3.1 describes the complete list of WinCE exposed driver functional interfaces that SerialCSR implements.

© Cambridge Silicon Radio Limited 2004
This material is subject to CSR's non-disclosure agreement.

3.3 Updating the Persistent Store: PSConfig()

The core of SerialCSR is the code to update the Persistent Store using the minimal CSR BCSP implementation known as μBCSP. A description of μBCSP is outside the scope of this document. Refer to the μBCSP User Guide.

Figure 3.3 denotes the high-level flow of information through the update function.

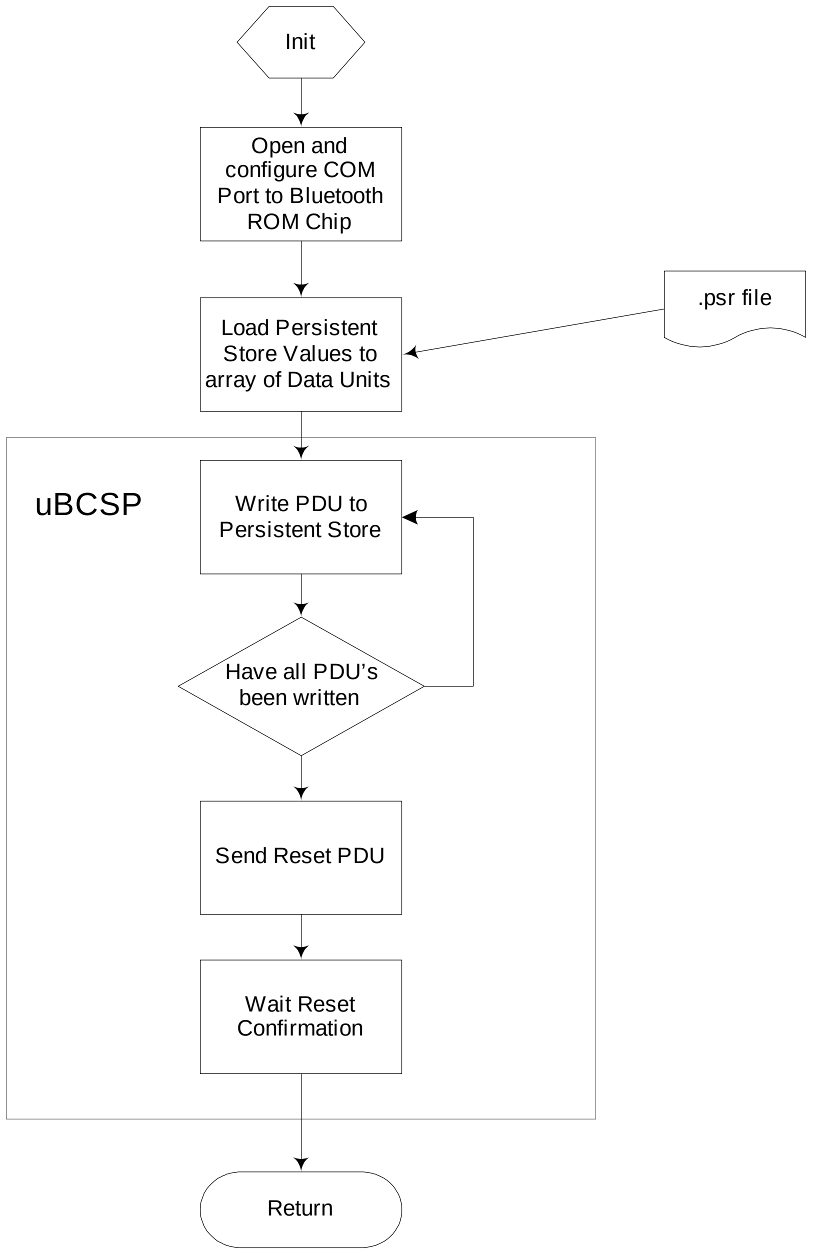

Init
Open and configure COM Port to Bluetooth ROM Chip
Load Persistent Store Values to array of Data Units
.psr file
uBCSP
Write PDU to Persistent Store
Have all PDU's been written
Send Reset PDU
Wait Reset Confirmation
Return

Figure 3.3: PSConfig Flow Diagram

© Cambridge Silicon Radio Limited 2004
This material is subject to CSR's non-disclosure agreement.

Figure 3.3 indicates the flow of information through the `PSConfig()` function and the transformation that it undergoes. PS Keys are loaded to an array of modified standard CSR Protocol Data Units (PDUs) defined as follows:

```
// Define PDU with static memory
// - Enables array handling
typedef struct {
   uint16      id;
   uint16      len;
   uint16      stores;
   uint16      psmem[BCCMDPDU_MAXBUFSIZ_PC];
 } BCCMDPDU_PS;

typedef struct {
   uint16      type;
   uint16      pdulen;
   uint16      seqno;
   uint16      varid;
   uint16      status;
   union
   {
      BCCMDPDU_PS ps;
   }d;
} BCCMDPDU;
```

The standard CSR PDU was modified to support static memory allocation rather than the common dynamic memory allocation of data. This allows an array of PDUs to be constructed before managing the update of the writing of PS Keys.

© Cambridge Silicon Radio Limited 2004
This material is subject to CSR's non-disclosure agreement.

3.4 Instantiation

In order to use SerialCSR instead of the regular device driver, changes must be made to the registry entries for the serial driver. Using the Widcomm stack as an example, under COM4 there is a reference to BT.dll. This reference should be altered to SerialCSR.dll. SerialCSR.dll will then load and instantiate the underlying serial driver, BT.dll or Serial.dll. For more information, see section 4, Installation.

The `PSConfig()` function is wrapped with the function `UpdatePersistentStore()`. This function is called based on the value of the flag (gbUpdatePS) set within the power handlers described in the next section.

The following example code describes a typical implementation of the wrapped COM port WINAPI function and the usage of the Persistent Store update code.

```
typedef ULONG (WINAPI *PFNCOM_Read)(HANDLE, PUCHAR, ULONG, PULONG);

//////////////////////////////////////////////////////////////////////
// COM_Read Wrapper
//
// The CSR implementation of this function ensures the persistent
// store of the CSR chip is updated on first access to the device
//
SERIALCSR_API ULONG COM_Read(HANDLE pContext, PUCHAR pTargetBuffer,
                      ULONG BufferLength, PULONG pBytesRead)
{
   ULONG lResult = NULL;

   if(ghLibInst)
   {
      if(gpfnCOM_Read)
      {
         // Update Persistent Store
         if(gbUpdatePS)
         {
            // Reset update flag
            gbUpdatePS = FALSE;
            UpdatePersistentStore(pContext);
         }
         lResult = gpfnCOM_Read(pContext, pTargetBuffer,
               BufferLength, pBytesRead);
      }
   }
   return lResult;
}
```

Similar handlers are used in `COM_Open()` and `COM_Write()`.

© Cambridge Silicon Radio Limited 2004
This material is subject to CSR's non-disclosure agreement.

3.5 Managing the Power Suspend

Upon loss of power to a ROM chip, all Persistent Store settings are lost and the device reverts to default values. Consequently SerialCSR must manage the Persistent Store of the ROM chip throughout the suspend power cycle.

WinCE uses two messages to notify device drivers of a system power change through a suspend state and these are handled by the exported functions `COM_PowerDown()` and `COM_PowerUp()`. Calls to both of these functions are passed on to the underlying driver and (as with all power handlers) these functions cannot call functions in other .dll files, memory allocators or debugging output functions; nor can they do anything that could cause a page fault. It is, however, safe to set a flag and this is done in `COM_PowerUp()`. The set flag is used throughout SerialCSR to determine whether an update to the Persistent Store should be performed.

Unfortunately, it is not possible to know definitely whether power has been lost to the chip during a suspend because power to the chip can be controlled either by WinCE or by the hardware level of the underlying device driver, or by both. Consequently even though the `COM_PowerUp()` handler may return `TRUE`, without a failsafe test it cannot be known for certain that power has indeed been lost to the chip unless a `TRUE` was received from `COM_PowerDown()` and that power was not lost to the chip during suspend when `COM_PowerDown()` recorded `FALSE`.

If an OEM is certain of the power state through suspend, then it is possible to consider one of the following three power handling scenarios:

Case A: COM_PowerDown handling only

- If the `COM_PowerDown()` return from PDD layer = `TRUE` set gbUpdatePS = `TRUE`
- If the `COM_PowerDown()` return from PDD layer = `FALSE` set gbUpdatePS = `FALSE`
- On `COM_PowerUp()`: ignore

Case B: COM_PowerUp handling only

If it is the case that the chip controls power and handles COM_`PowerDown()` correctly but later kills power then a rewrite of PS values is required.

This requires COM_PowerUp handling as follows

- If the `COM_PowerUp()` return from PDD layer = `TRUE` set gbUpdatePS = `TRUE`
- If the `COM_PowerUp()` return from PDD layer = `FALSE` set gbUpdatePS = `FALSE`
- On `COM_PowerDown()`: ignore

In case B, it is vital that `COM_PowerUp()` return from the hardware layer of the underlying driver (the PDD layer), write `TRUE` if (and only if) power was down before it returned `TRUE` from `COM_PowerUp()`. This is not normal handling, therefore CASE A is preferred.

© Cambridge Silicon Radio Limited 2004
This material is subject to CSR's non-disclosure agreement.

Case C: Combine both handling

To combine handling with both `COM_PowerUp()` and `COM_PowerDown()` produces four possible states.

	Down		
Up		TRUE	FALSE
	TRUE	1	2
	FALSE	3	4

Actions/States:

1. Update Persistent Store again as power was switched off and then switched on.
2. Do nothing as power was always on.
3. No power restored to device, but store gbWriteAgain=`TRUE` and update Persistent Store on next access as `COM_Open()` will be called once power is restored.
4. No power off and no power on indicates indeterminate state.

Problems arise in:

State 2. Power may have been switched off during suspend.

State 4. Here power was cut when `COM_PowerDown()` message was received, but it was not bought up on `COM_PowerUp()`. This is a case of the PDD controlling the chip.

SerialCSR Implementation

The supplied code module takes a safe approach by flagging the `COM_PowerUp()` whenever it is handled. SerialCSR then ensures that (if the flag is set) on the next call through `COM_Init()`, `COM_Open()`,`COM_Read()` or `COM_Write()`, the Persistent Store values are updated before continuing. These values are written even if power has not been lost. If the baud rate is different to the default value, the update will fail cleanly.

This approach avoids any of the Cases A, B or C and guarantees that the Persistent Store is updated whenever it is needed. More efficient handling is possible and can be implemented with regard to Cases A, B or C, depending on the management of the power cycle by the OEM.

Note:

If the suspend goes on for so long that the device runs out of power, then it is rebooted and the Persistent Store is updated during the boot.

© Cambridge Silicon Radio Limited 2004
This material is subject to CSR's non-disclosure agreement.

4 Installation

In order for SerialCSR to be coupled with the Bluetooth protocol stack, the Bluetooth stack must use SerialCSR instead of the usual serial driver. This is performed by editing the registry. When SerialCSR is loaded, it will (in turn) load up the original serial driver as directed by the hard-coded value in the SerialCSR source code.

All Windows CE driver locations are stored within the system registry and can be found in the section **HKEY_LOCAL_MACHINE\Drivers\Builtin\<subsection>\Driver**.

Individual OEM implementations differ in the title of <subsection>. A typical Widcomm implementation uses the <subsection> name of “Bluetooth” and the serial driver name is “BT.dll”. When using the Microsoft CE stack, the driver name is the standard “Serial.dll”

As shipped, the source code for SerialCSR links with BT.dll dynamically as the underlying serial device driver, making the implementation suitable for the Widcomm stack. If SerialCSR is used with a Microsoft stack, the reference to BT.dll in the SerialCSR source code should be changed to Serial.dll.

To implement the shim layer, the only requirement is to copy the compiled module SerialCSR.dll to the Windows directory on the device and change the above entry for “Driver” to “SerialCSR.dll”. After rebooting the device, all access to the Bluetooth chip will be passed via SerialCSR.dll through to the underlying .dll with Persistent Store settings updated as required.

© Cambridge Silicon Radio Limited 2004
This material is subject to CSR’s non-disclosure agreement.

5 Integration, Production Test and Configuration Issues

From the standard image (as released from the Platform Builder), several changes must be made. These include the following modifications.

- Modifying the image's registry so that SerialCSR driver is installed as a filter driver (see section 4, Installation).
- Incorporating a customised version of the device driver SerialCSR.dll

For every device produced, a unique .psr file must be incorporated into the ROM image. This file must contain the following information:

- Settings unique to that device, e.g., the Bluetooth address and crystal frequency trim
- Settings used by all similar devices, namely the host transport, baud rate, device name, etc.

In order to get the crystal frequency trim, various tests must be made on BlueCore. Some of these tests will involve radio and Bluetooth function. This will have a significant impact on the production plan used in creating the devices.

The .psr file needs to be blown into ROM, since it contains information that will have to persist beyond a hard reset. The information is not replaceable by a user.

© Cambridge Silicon Radio Limited 2004
This material is subject to CSR's non-disclosure agreement.

6 Document References

Document	**Reference**
Selection of I^{2}C EEPROMs for Use with BlueCore	CSR document bcore-an-008P
µBCSP User Guide	CSR document bcor-ug-001P
BCCMD Protocol	CSR document bcore-sp-002P

© Cambridge Silicon Radio Limited 2004
This material is subject to CSR's non-disclosure agreement.

Acronyms and Definitions

API	Application-Program Interface; set of functions and accessible data that define the interface between two software components
BCCMD	BlueCore Command
BCSP	BlueCore Serial Protocol
BlueCore™	Group term for CSR's range of Bluetooth wireless technology chips
Bluetooth®	Set of technologies providing audio and data transfer over short-range radio connections
BOM	Bill of Materials
BT.dll	On the Widcomm CE stacks, this is the serial driver. On the Microsoft CE stacks, this is the main part of the Bluetooth stack that exists above the serial driver
CE	Windows® CE; a small version of Microsoft Windows targeted at Consumer Electronics
CSR	Cambridge Silicon Radio
dll	Dynamically Linked Library
EEPROM	Electronically Erasable Programmable Read-Only Memory; small, low cost non-volatile memory
HID	Human Interface Device
I^2C™	Inter-Integrated Circuit
I/O	Input/Output
OEM	Original End Manufacturer
PC	Personal Computer
PDA	Personal Digital Assistant; portable device storing appointments, contacts and similar, These are increasingly able to communicate with other devices by Bluetooth
PDU	Protocol Data Unit
Persistent Store	Storage of BlueCore's configuration values in non-volatile memory
PS Key	Persistent Store Key
ROM	Read Only Memory
UART	Universal Asynchronous Receiver Transmitter
µBCSP	"Micro" BlueCore Serial Protocol
USB	Universal Serial Bus

© Cambridge Silicon Radio Limited 2004
This material is subject to CSR's non-disclosure agreement.

Record of Changes

Date	Revision	Reason for Change
10 May 04	a	Original publication of this document. (CSR reference: bcore-an-021Pa)

Windows®CE Driver for BlueCore™ ROM Devices

Application Note

bcore-an-021Pa

May 2004

Bluetooth® and the Bluetooth logos are trademarks owned by Bluetooth SIG, Inc. and licensed to CSR.

BlueCore™ is a trademark of CSR.

All other product, service and company names are trademarks, registered trademarks or service marks of their respective owners.

CSR's products are not authorised for use in life-support or safety-critical applications.