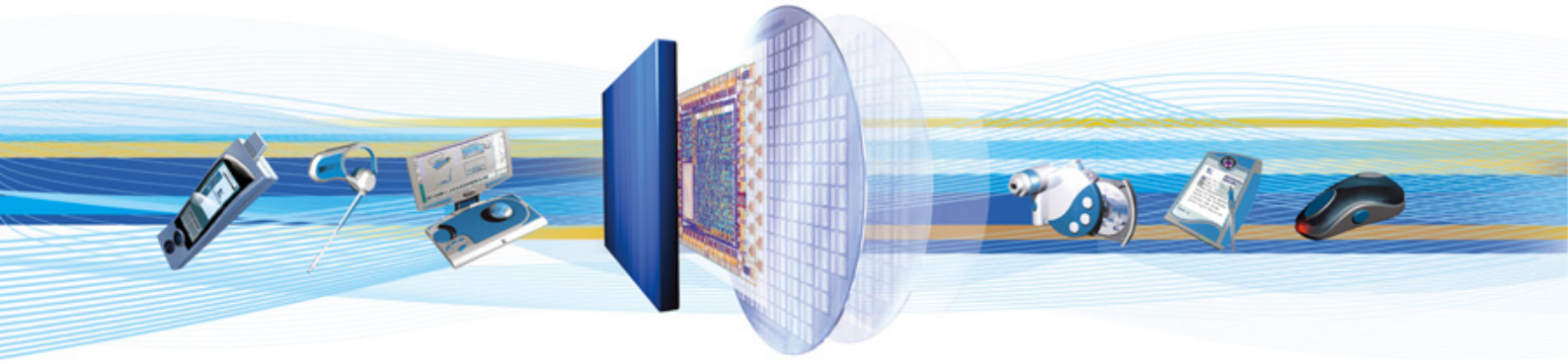
CSR Synergy Bluetooth 18.2.0

vCard

API Description

November 2011



Cambridge Silicon Radio Limited

Churchill House
Cambridge Business Park
Cowley Road
Cambridge CB4 0WZ
United Kingdom

Registered in England and Wales 3665875

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

Contents

List of Tables

1 Introduction

The vCard library consists of a parser and a generator for vCard 2.1 and vCard 3.0. Both the parser and the generator are designed to work with “on demand” decoding and encoding with some limitation. This allows the application to either decode large documents divided into smaller blocks of data one block at the time or encode vCard into multiple small buffers, making the vCard library suitable for sending and receiving vCard on network connections. The other advantage of the “on demand” technique is that the memory management does not have to support large block of memory allocation. To get a reliable result from the parser all the data must be parsed before the decoded data can be used in the application.

The vCard library uses a common structure for the data which the parser stores the parsed data into and the generator encodes vCard based on this structure. The data structure is based on linked list of different types of elements representing the data for the vCard. Multiple vCard is also linked together in the structure. The data elements are accessible fully decoded and all vCard encodings are removed. Also the generator internally handles all the vCard specific encoding such as print-quotable, base64 and other dedicated vCard encoding styles. The application can also choose to have all the strings in the entire structure represented as UTF-8 or UTF-16 (UCS-2). This can be specified as a parameter to the parser also when generating vCard the actual encoding should be specified.

The parser is by default in strict parsing mode where the vCard must be fully compliant with the vCard specification. Since the vCard v3.0 specification is stricter than the vCard v2.1 specification some implementations of vCard v3.0 generators do not fully comply with the specification causing the parser to fail the parsing. To avoid this problem multiple flags can be set to removed some of the construct rules that are specified by the specification. Also it is possible to parse documents that are constructed based on both vCard v2.1 constructs and vCard v3.0 constructs. Using all the possible flags would increase the possibility to decode all of types of vCard constructs.

The vCard library supports most of the vCard specification with few exception/limitations:

- No non-standard attribute names are available
- Grouping is parsed, but the information is not returned
- Only UTF-8 encoding input is supported
- Language and char set parameters are ignored
- “X-“ attribute names are not supported except X_IRMC_CALL_DATETIME
- “X-“ parameter names are not supported

Memory management is handled internally by the vCard parser and provides functionality to cleanup the parsed data. Do not free any of the structures returned by the parser, but use the provided function.

2 Concept

The vCard library uses a number of different concepts. Each of there concepts are detailed in this section starting with basic function constructs following with general parameter details and error codes.

2.1.1 Function Construct

Both the parser and generator use some internal structures to manage details of the decoding or encoding. These structures must be initialized prior to using the parser or generator. To this end a initializing function is provided for both the parser (**CsrBtVcardParseInitialize(…)**) and the generator (**CsrBtVcardGeneratorInitialize(…)**).

The generator provides a writing function where the data structure can be written as a vCard to an application provided buffer.

The parser provides a reading function for parsing vCard from an application provided buffer. In addition a special version of the reading function to only detect the vCard version is provided. This function only scans for the version attribute and does not store any data making it much faster. After the parsing is complete the function **CsrBtVcardParseComplete** must be called to clean internal data structure and gain access to the parsed data. This function must be called even if the reading function fails during parsing.

2.1.2 Parameter Details

The parser and generator have a common flags concept used for controlling the parsing and generation. This parameter is based on a bit pattern allowing multiple flags to be specified. Not all of the flags are available in both the parser and generator.

The flags are specified as followed in the **csr_bt_vcard.h** file:

```
#define CSR_BT_VCARD_FLAGS_DEFAULT                              (0x0000)
#define CSR_BT_VCARD_FLAGS_UCS2_STRINGS                         (0x0001)
#define CSR_BT_VCARD_FLAGS_ALLOW_UNSPECIFIED_ATTRIBUTE_NAMES    (0x0002)
#define CSR_BT_VCARD_FLAGS_ALLOW_UNSPECIFIED_PARAMETER_NAMES    (0x0004)
#define CSR_BT_VCARD_FLAGS_ALLOW_UNSPECIFIED_ENCODING           (0x0008)
#define CSR_BT_VCARD_FLAGS_NO_MANDATORY_TAGS                    (0x0010)
#define CSR_BT_VCARD_FLAGS_VERSION_2_1                          (0x0000)
#define CSR_BT_VCARD_FLAGS_VERSION_3_0                          (0x0020)
#define CSR_BT_VCARD_FLAGS_NO_VERSION_MATCH                     (0x0040)
#define CSR_BT_VCARD_FLAGS_VERSION_SCAN                         (0x0080)
#define CSR_BT_VCARD_FLAGS_DISABLE_2_1_TYPE_CHECK               (0x0100)
```

A detailed description of each of the flags:

- `CSR_BT_VCARD_FLAGS_DEFAULT` should be used when no flags are needed. For the parser this flag implies strict parsing and UTF-8 output and for the generator specifies that the strings in the data structure are represented as UTF-8
- `CSR_BT_VCARD_FLAGS_UCS2_STRINGS` forces the parser to produce UCS-2 string as output instead on the normal UTF-8 strings and for the generator it specifies that the strings in the data structure are represented as UTF-16 / UCS-2
- `CSR_BT_VCARD_FLAGS_ALLOW_UNSPECIFIED_ATTRIBUTE_NAMES` will instruct the parser not to fail when it encounters an unspecified attribute name. No effect on the generator
- `CSR_BT_VCARD_FLAGS_ALLOW_UNSPECIFIED_PARAMETER_NAMES` will instruct the parser not to fail when it encounters an unspecified parameter name. No effect on the generator
- `CSR_BT_VCARD_FLAGS_ALLOW_UNSPECIFIED_ENCODING` will instruct the parser not to fail when it encounters an unknown encoding. No effect on the generator
- `CSR_BT_VCARD_FLAGS_NO_MANDATORY_TAGS` will instruct the parser not to fail when N, FN or VERSION is not present in the vCard. No effect on the generator

- CSR_BT_VCARD_FLAGS_VERSION_2_1 will instruct the generator to produce vCard v2.1 output. No effect on the parser
- CSR_BT_VCARD_FLAGS_VERSION_3_0 will instruct the generator to produce vCard v3.0 output. No effect on the parser
- CSR_BT_VCARD_FLAGS_NO_VERSION_MATCH will instruct the parser not to fail when it encounters both vCard v2.1 constructs and vCard v3.0 constructs. No effect on the generator
- CSR_BT_VCARD_FLAGS_VERSION_SCAN will instruct the parser to only detect the version during parsing of the data. No effect on the generator
- CSR_BT_VCARD_FLAGS_DISABLE_2_1_TYPE_CHECK will instruct the parser to skip type checking on vCard 2.1 data which accordingly to version 3.0 of the vCard specification is invalid.

Note that all structures returned uses a type call CsrCharString which is defined as an unsigned char*, but when the flag CSR_BT_VCARD_FLAGS_UCS2_STRINGS is used this variable is a pointer to a unsigned short * instead and proper casting should be used.

2.1.3 Error Codes

The parser can return a number of error codes to identify the successfulness of the parsing. These error codes are divided into to two major groups “successful error codes” and “fatal error codes”.

The successful error codes are used when the parsing are successfully completed both with none or more incorrect constructs. When one of these error codes is returned the data structure contains all the data that was successfully parsed.

The fatal error codes are returned when the parser could complete the parsing of the input data. The parser does not return any data structure and the application must take appropriate action.

The error codes are constructed as a bitmask allowing multiple errors to occur at the same time. This is especially useful with the successful error codes since a document can have multiple errors.

The error codes are defined as followed in the **csr_bt_vcard.h** file:

```
#define CSR_BT_VCARD_ERROR_SUCCESS                      (0x00000000)
#define CSR_BT_VCARD_ERROR_UNKNOWN_PARAMETER_NAME       (0x00000001)
#define CSR_BT_VCARD_ERROR_UNKNOWN_ATTRIBUTE_NAME       (0x00000002)
#define CSR_BT_VCARD_ERROR_UNKNOWN_ENCODING             (0x00000004)
#define CSR_BT_VCARD_ERROR_INCOMPLETE                   (0x00000008)
#define CSR_BT_VCARD_ERROR_SYNTAX                       (0x80000010)
#define CSR_BT_VCARD_ERROR_INTERNAL_ERROR               (0x80000040)
#define CSR_BT_VCARD_ERROR_PARAMETER_NAME               (0x80000080)
#define CSR_BT_VCARD_ERROR_ATTRIBUTE_NAME               (0x80000100)
#define CSR_BT_VCARD_ERROR_INVALID_ENCODING             (0x80000200)
#define CSR_BT_VCARD_ERROR_INVALID_VALUE                (0x80000400)
#define CSR_BT_VCARD_ERROR_MISSING_MANDATORY_TAGS       (0x80000800)
#define CSR_BT_VCARD_ERROR_WRONG_VERSION                (0x80001000)
#define CSR_BT_VCARD_ERROR_MULTIPLE_VERSION_CONSTRUCT   (0x80002000)
#define CSR_BT_VCARD_ERROR_INVALID_PARAMETER            (0x80004000)
```

A detailed description of each of the successful error codes:

- CSR_BT_VCARD_ERROR_SUCCESS: The operation was successful with no errors
- CSR_BT_VCARD_ERROR_UNKNOWN_PARAMETER_NAME: The operation was successful with entries having unknown parameter names which do not conform to the specification
- CSR_BT_VCARD_ERROR_UNKNOWN_ATTRIBUTE_NAME: The operation was successful with entries having an unknown attribute name which does not conform to the specification. Currently instant messaging attributes would cause this error to be returned
- CSR_BT_VCARD_ERROR_UNKNOWN_ENCODING: The operation was successful with entries being encoded with an algorithm which does not conform to the specification

- CSR_BT_VCARD_ERROR_INCOMPLETE: The operation is not completed. This is normally returned when the application used multiple buffers to decode the data. Each of the parsing operation would properly return this error to indicate that some of the vCard data are missing before the entire card is decoded. If this error is returned when calling the **CsrBtVcardParseComplete** the returned data structure only contains vCards that were decoded successfully and might indicate to the caller that not all the data are available.

A detailed description of each of the fatal error codes:

- CSR_BT_VCARD_ERROR_SYNTAX: The syntax of the vCard is incorrect. This is returned when the parser is unable to understand the construct of the data
- CSR_BT_VCARD_ERROR_INTERNAL_ERROR Internal error used for indicating an abnormal unrecoverable situation
- CSR_BT_ VCARD_ERROR_PARAMETER_NAME: A parameter name was not recognized as being part of the specification. This error can only be returned if the CSR_BT_ VCARD_FLAGS_ALLOW_UNSPECIFIED_PARAMETER_NAMES flag is not specified otherwise a CSR_BT_VCARD_ERROR_UNKNOWN_PARAMETER_NAME error code is returned instead
- CSR_BT_VCARD_ERROR_ATTRIBUTE_NAME: An attribute name was not recognized as being part of the specification. This error can only be returned if the CSR_BT_ VCARD_FLAGS_ALLOW_UNSPECIFIED_ATTRIBUTE_NAMES flag is not specified otherwise a CSR_BT_ VCARD_ERROR_UNKNOWN_ATTRIBUTE_NAME error code is returned instead
- CSR_BT_VCARD_ERROR_INVALID_ENCODING: An invalid encoding was used and is not part of the specification. This error can only be returned if the CSR_BT_VCARD_FLAGS_ALLOW_UNSPECIFIED_ENCODING flag is not specified otherwise a CSR_BT_ VCARD_ERROR_UNKNOWN_ENCODING error code is returned instead.
- CSR_BT_VCARD_ERROR_INVALID_VALUE: A invalid value exists
- CSR_BT _VCARD_ERROR_MISSING_MANDATORY_TAGS: The mandatory tags are missing meaning VERSION and N for vCard v2.1 and VERSION, N and FN for vCard v3.0. This error can only be returned if the CSR_BT_VCARD_FLAGS_NO_MANDATORY_TAGS flag is not specified otherwise the parser does check is these tags are present
- CSR_BT_VCARD_ERROR_WRONG_VERSION: The version number in the data is not 2.1 or 3.0
- CSR_BT_VCARD_ERROR_MULTIPLE_VERSION_CONSTRUCT: The vCard does not comply with either the vCard v2.1 or vCard 3.0 specifications but is a combination of the two. This can occur if the vCard is formatted according to the vCard v2.1 specification but version number indicates the version is actually 3.0. This error can only be returned if the CSR_BT_VCARD_FLAGS_NO_VERSION_MATCH flag is not specified
- CSR_BT_VCARD_ERROR_INVALID_PARAMETER: An invalid input parameter was specified when calling one of the vCard library functions

A helper function is provided to help detect if the operation is a success or not. It is implemented as a macro as follows:

```
#define CSR_BT_VCARD_SUCCEEDED(error) ((error & 0x80000000) != 0x80000000)
```

It is recommended to use a macro to detect if the parsing is a success.

3 vCard API Functions

This section gives an overview of the functions and parameters in the interface. Detailed information can be found in the corresponding **csr_bt_vcard.h** file.

Functions	Reference
CsrBtVcardGeneratorInitialize	See section 3.1
CsrBtVcardGeneratorWrite	See section 3.2
CsrBtVcardParseInitialize	See section 3.3
CsrBtVcardParseRead	See section 3.4
CsrBtVcardParseReadVersionOnly	See section 3.5
CsrBtVcardParseComplete	See section 3.6
CsrBtVcardParse	See section 3.7
CsrBtVcardFree	See section 3.8
CsrBtVcardIsDateValid	See section 3.9
CsrBtVcardIsTimeValid	See section 3.10

Table 1: List of vCard API functions

3.1 CsrBtVcardGeneratorInitialize

Function \ Parameters	control	flags
CsrBtVcardGeneratorInitialize	In	In

Table 2: CsrBtVcardGeneratorInitialize

Prototype

```
void CsrBtVcardGeneratorInitialize(CsrBtVcardGeneratorControl * control,
CsrUint32 flags);
```

Description

This function is used for initializing the generator by setting default values in the control structure provided by the caller. It must be called prior to using the **CsrBtVcardGeneratorWrite** function.

Parameters

control	Reference to a **CsrBtVcardGeneratorControl** structure which should be allocated by the caller.
flags	Specify different flags to modify the outcome of the generator. See section 2.1.2.

Return Value

None

3.2 CsrBtVcardGeneratorWrite

Parameters / Function	CsrBtsVcardData	buffer	bufferSize	control
CsrBtVcardGeneratorWrite	In	In	In	In

Table 3: CsrBtVcardGeneratorWrite

Prototype

```
CsrUint32 CsrBtVcardGeneratorWrite(CsrBtVcardData * CsrBtVcardData,
CsrUint8 * buffer, CsrUint32 bufferSize, CsrBtVcardGeneratorControl *
control);
```

Description

This function is used for generating a vCard. There are two ways of using the function either the caller can use it to encode one vCard at the time and control the storing of the data or to encode multiple vCard linked together resulting in a continuous document holding all the vCard. The former method requires less memory both more buffer handling in the application and the latter requires more memory and less handling for the application.

To generate one or more vCards the function might have to be called multiple times if the buffer provided is not large enough to hold the entire context. The return value indicates how far the generator is with the generation.

Currently there is no way to determine the size of the final document prior to generating the context.

It is important that the data structure is deleted before the generation is completed.

Parameters

CsrBtVcardData	A pointer to a vCard data structure. See section 4
buffer	Reference to a buffer that should hold the generated data
bufferSize	Size of the buffer pointed by the buffer parameter
control	Reference to a **CsrBtVcardGeneratorControl** structure which should be initialized with the **CsrBtVcardGeneratorInitialize** function.

Return Value

The function returns the number of bytes used in the buffer provided by the caller. To determine the status of the decoding use the following rules:

- Value [0x0000 → (bufferSize – 1)]: The function has completed the generation of the context represented in the CsrBtVcardData variable
- Value [bufferSize]: The function filled the buffer and did not complete the generation of the vCard. By calling the function again the generator continues the generation
- Value [0xFFFF]: Indicates an error

3.3 CsrBtVcardParseInitialize

Function \ Parameters	control	flags
CsrBtVcardParseInitialize	In	In

Table 4: CsrBtVcardParseInitialize

Prototype

```
void CsrBtVcardParseInitialize(CsrBtVcardParserControl * control, CsrUint32
flags);
```

Description

This function is used for initializing the generator by setting default values in the control structure provided by the caller. It must be called prior to using the functions **CsrBtVcardParseRead**, **CsrBtVcardParseReadVersionOnly** and **CsrBtVcardParseComplete**.

Parameters

control	Reference to a **CsrBtVcardGeneratorControl** structure which should be allocated by the caller.
flags	Specify different flags to modify the outcome of the generator. See section 2.1.2.

Return Value

None

3.4 CsrBtVcardParseRead

Function \ Parameters	buffer	bufferSize	control
CsrBtVcardParseRead	In	In	In

Table 5: CsrBtVcardParseRead

Prototype

```
CsrUint32 CsrBtVcardParseRead(CsrUint8 * buffer, CsrUint32 bufferSize,
CsrBtVcardParserControl* control);
```

Description

This function is used for parsing a vCard document provided by the caller in single or multiple buffers and it stores it internally until all the data have been parsed. Since the function uses dynamic memory allocation internally to store the data, memory consumption can be a problem when using this function. Once the caller provided all the available data to the parser the function **CsrBtVcardParseComplete** should be called to retrieve the parsed data.

Parameters

buffer	Reference to a buffer containing the vCard document to be parsed.
bufferSize	Size of the buffer pointed by the buffer parameter.
control	Reference to a **CsrBtVcardParserControl** structure which should be initialized with the **CsrBtVcardParseInitialize** function.

Return Value

See section 2.1.3.

Note: The caller can choose to ignore the returned value and only check the error code returned from the **CsrBtVcardParseComplete** function.

3.5 CsrBtVcardParseReadVersionOnly

Function \ Parameters	buffer	bufferSize	control
CsrBtVcardParseReadVersionOnly	In	In	In

Table 6: CsrBtVcardParseReadVersionOnly

Prototype

```
CsrUint32 CsrBtVcardParseReadVersionOnly(CsrUint8 * buffer, CsrUint32
bufferSize, CsrBtVcardParserControl* control);
```

Description

This function is used for extracting the version number of a vCard document provided by the caller in single or multiple buffers. The function is similar to the **CsrBtVcardParseRead** function except that data is not stored and most of the verification functionality is bypassed. The advantage of using this function compared to the **CsrBtVcardParseRead** function is a significant increase in parsing speed and yet providing a quick way of reading the version number which is returned by calling the function **CsrBtVcardParseComplete**.

Parameters

buffer	Reference to a buffer containing the vCard document to be parsed.
bufferSize	Size of the buffer pointed by the buffer parameter.
control	Reference to a **CsrBtVcardParserControl** structure which should be initialized with the **CsrBtVcardParseInitialize** function.

Return Value

See section 2.1.3.

Note: The caller can choose to ignore the returned value and only check the error code returned from the **CsrBtVcardParseComplete** function.

3.6 CsrBtVcardParseComplete

Function \ Parameters	CsrBtVcardDataEntries	vcardData	version	control
CsrBtVcardParseComplete	Out	Out	Out	In

Table 7: CsrBtVcardParseComplete

Prototype

```
CsrUint32 CsrBtVcardParseComplete(CsrUint32 * CsrBtVcardDataEntries,
CsrBtVcardData ** vcardData, CsrBtVcardVersion* version,
CsrBtVcardParserControl* control);
```

Description

This function completes the parsing operation and returns the parsed data to the caller. This function must be called if one of the functions **CsrBtVcardParseRead** or **CsrBtVcardParseReadVersionOnly** has been used even if they returned a fatal error otherwise a memory leak can occur.

Parameters

CsrBtVcardDataEntries	Reference to a variable that will hold the number of entries parsed. This parameter can be NULL.
vcardData	Reference to a variable that will hold the parsed data. This data has to be freed using the **CsrBtVcardFree** function otherwise memory leak can occur. See section 4 for more details. Data will only be available if the **CsrBtVcardParseRead** function is used for parsing the data. This parameter can be NULL.
version	Reference to a variable that will hold the version identifier for the provided document. This parameter can be NULL.
control	Reference to a **CsrBtVcardParserControl** structure which should be initialized with the **CsrBtVcardParseInitialize** function.

Return Value

See section 2.1.3.

3.7 CsrBtVcardParse

Parameters / Function	buffer	bufferSize	CsrBtVcardDataEntries	CsrBtVcardData	flags
CsrBtVcardParse	In	In	Out	Out	In

Table 8: CsrBtVcardParse

Prototype

```
CsrUint32 CsrBtVcardParse(CsrUint8 * buffer, CsrUint32 bufferSize,
CsrUint32 * CsrBtVcardDataEntries, CsrBtVcardData ** CsrBtVcardData,
CsrUint32 flags);
```

Description

This function is used for parsing a vCard document that can be represented in a single continuous buffer. This function is implemented using the other parsing functions internally and is provided as a simpler way of parsing a vCard document. The data structure returned by this function in the **CsrBtVcardData** variable hold the parsed data and should be freed using the **CsrBtVcardFree** function

Parameters

buffer	Reference to a buffer containing the vCard document to be parsed.
bufferSize	Size of the buffer pointed by the buffer parameter.
CsrBtVcardDataEntries	Returns the number of vCard parsed in the document. This parameter can be NULL.
CsrBtVcardData	Return a complex structure holding the parsed information. See section 4.
flags	Specify different flags modify the outcome of the parser. See section 2.1.2.

Return Value

See section 2.1.3.

3.8 CsrBtVcardFree

Function \ Parameters	CsrBtVcardData
CsrBtVcardFree	In

Table 9: CsrBtVcardFree

Prototype

```
void CsrBtVcardFree(CsrBtVcardData * CsrBtVcardData);
```

Description

This function frees the memory allocated for the vCard data structure by the parser. Using this function with custom created vCard data structure might cause problems and should be used carefully.

Parameters

CsrBtVcardData	A reference to a **CsrBtVcardData** structure returned by the **CsrBtVcardParse** function or **CsrBtVcardParseComplete** function.

Return Value

None.

3.9 CsrBtVcardIsDateValid

Function \ Parameters	iso8601Date
CsrBtVcardIsDateValid	In

Table 10: CsrBtVcardIsDateValid

Prototype

```
CsrBool CsrBtVcardIsDateValid(CsrBtIso8601Date * iso8601Date);
```

Description

This function is used determine if a date entry has a valid value. The function fails if input data is NULL or the year entry is 0.

Parameters

iso8601Date Reference to a **CsrBtIso8601Date** structure to be validated.

Return Value

True if date is valid

3.10 CsrBtVcardIsTimeValid

Function \ Parameters	iso8601Time
CsrBtVcardIsTimeValid	In

Table 11: CsrBtVcardIsTimeValid

Prototype

```
CsrBool CsrBtVcardIsTimeValid(CsrBtIso8601Time * iso8601Time);
```

Description

This function is used determine if a time entry has a valid value. The function fails if input data is NULL or the flags entry is 0.

Parameters

`iso8601Time` Reference to a **CsrBtIso8601Time** structure to be validated.

Return Value

True if time is valid

4 Data Structures

A number of structures are defined to hold the different types of information found in a vCard. The structure **CsrBtVcardData** holds one vCard and has a reference to additional vCards. In this structure all the possible attributes can be found. Each attribute is defined as the structure **CsrBtVcardDataType** which describes how the data is represented. In the next sub-sections each structure is described in details.

4.1 CsrBtVcardData

CsrBtVcardData holds references to all the possible attributes in the vCard. If the attribute exists in the document it is assigned a valid reference to a linked list of **CsrBtVcardDataType** otherwise it is NULL. Prior to reading an attribute verify that it is not a NULL pointer.

The structure has a next pointer, when might hold a reference to another **CsrBtVcardData** structure. It is up to the application to enumerate the linked list of **CsrBtVcardData** structure.

Definition:

```
typedef struct CsrBtVcardDataRef
{
    struct CsrBtVcardDataRef * next;
    CsrBtVcardDataType *       formattedName;
    CsrBtVcardDataType *       vcardName;
    CsrBtVcardDataType *       name;
    CsrBtVcardDataType *       nickNames;
    CsrBtVcardDataType *       photo;
    CsrBtVcardDataType *       birthDate;
    CsrBtVcardDataType *       source;
    CsrBtVcardDataType *       address;
    CsrBtVcardDataType *       label;
    CsrBtVcardDataType *       tel;
    CsrBtVcardDataType *       email;
    CsrBtVcardDataType *       mailer;
    CsrBtVcardDataType *       timezone;
    CsrBtVcardDataType *       globalPosition;
    CsrBtVcardDataType *       jobTitle;
    CsrBtVcardDataType *       jobRole;
    CsrBtVcardDataType *       jobLogo;
    CsrBtVcardDataType *       jobAgent;
    CsrBtVcardDataType *       organization;
    CsrBtVcardDataType *       catagories;
    CsrBtVcardDataType *       note;
    CsrBtVcardDataType *       productIdentifier;
    CsrBtVcardDataType *       revision;
    CsrBtVcardDataType *       sortString;
    CsrBtVcardDataType *       sound;
    CsrBtVcardDataType *       uid;
    CsrBtVcardDataType *       url;
    CsrBtVcardDataType *       classification;
    CsrBtVcardDataType *       key;
    CsrBtVcardDataType *       xIrmcCallDatetime;
    CsrBtVcardDataType *       xIrmcLuid;

} CsrBtVcardData;
```

Description:

Detailed description about each attribute can be found in RFC2426.

4.1.1 CsrBtVcardDataType

CsrBtVcardDataType defines how an attribute is represented. The dataType value determines which kind of payload the attribute has. The payload union has a list of the possible data types. See the following sub-section for detail about the structures referenced.

Definition:

```
typedef struct CsrBtVcardDataTypeRef
{
    struct CsrBtVcardDataTypeRef *       next;
    CsrUint8                             dataType;
    union
    {
        CsrCharString *                  string;
        CsrBtVcardDataUriString *        uriString;
        CsrBtVcardDataBinary *           binary;
        CsrBtVcardDataAddress *          address;
        CsrBtVcardDataName *             name;
        CsrBtVcardDataTelephone *        telephone;
        CsrBtIso8601Date *               iso8601Date;
        CsrBtIso8601DateTime *           iso8601DateTime;
        CsrBtVcardDataLabel *            label;
        CsrBtVcardDataTypedString *      typedString;
        CsrBtVcardDataIso8601Date *      iso8601DateTyped;
        CsrBtVcardDataIso8601DateTime *  iso8601DateTimeTyped;
        CsrBtVcardDataOrganization *     organization;
        void *                           data;
    } payload;
} CsrBtVcardDataType;
```

Description:

The following table is a list of possible **dataType** values and details about which variable should be used in the payload union:

DataType	Description
CSR_BT_VCARD_DATA_TYPE_NONE	The payload does not hold a value
CSR_BT_VCARD_DATA_TYPE_STRING_UTF8	The string field holds a zero terminated UTF-8 encoded string
CSR_BT_VCARD_DATA_TYPE_STRING_UCS2	The string field holds a zero terminated UCS-2 encoded string
CSR_BT_VCARD_DATA_TYPE_URI_UTF8	The uriString field holds a zero terminated UTF-8 encoded string
CSR_BT_VCARD_DATA_TYPE_URI_UCS2	The uriString field holds a zero terminated UCS-2 encoded string
CSR_BT_VCARD_DATA_TYPE_BINARY	The binary field holds a reference to a `CsrBtVcardDataBinary` structure
CSR_BT_VCARD_DATA_TYPE_ADDRESS_UTF8	The address field holds a reference to a `CsrBtVcardDataAddress` structure.

DataType	Description
	Each string in the structure is zero terminated UTF-8 encoded strings
CSR_BT_VCARD_DATA_TYPE_ADDRESS_UCS2	The address field holds a reference to a `CsrBtVcardDataAddress` structure. Each string in the structure is zero terminated UCS-2 encoded strings
CSR_BT_VCARD_DATA_TYPE_NAME_UTF8	The name field holds a reference to a `CsrBtVcardDataName` structure. Each string in the structure is zero terminated UTF-8 encoded strings
CSR_BT_VCARD_DATA_TYPE_NAME_UCS2	The name field holds a reference to a `CsrBtVcardDataName` structure. Each string in the structure is zero terminated UCS-2 encoded strings
CSR_BT_VCARD_DATA_TYPE_TELEPHONE_UTF8	The telephone field holds a reference to a `CsrBtVcardDataTelephone` structure. Each string in the structure is zero terminated UTF-8 encoded strings
CSR_BT_VCARD_DATA_TYPE_TELEPHONE_UCS2	The telephone field holds a reference to a `CsrBtVcardDataTelephone` structure. Each string in the structure is zero terminated UCS-2 encoded strings
CSR_BT_VCARD_DATA_TYPE_ISO8601_DATE	The date field holds a reference to a `CsrBtIso8601Date` structure.
CSR_BT_VCARD_DATA_TYPE_ ISO8601_DATETIME	The dateTime field holds a reference to a `CsrBtIso8601DateTime` structure.
CSR_BT_VCARD_DATA_TYPE_LABEL_UTF8	The label field holds a reference to a `CsrBtVcardDataLabel` structure. Each string in the structure is zero terminated UTF-8 encoded strings
CSR_BT_VCARD_DATA_TYPE_LABEL_UCS2	The label field holds a reference to a `CsrBtVcardDataLabel` structure. Each string in the structure is zero terminated UCS-2 encoded strings
CSR_BT_VCARD_DATA_TYPE_TYPED_STRING_UTF8	The typed string field holds a reference to a `CsrBtVcardDataTypedString` structure. Each string in the structure is zero terminated UTF-8 encoded strings
CSR_BT_VCARD_DATA_TYPE_TYPED_STRING_UCS2	The label field holds a reference to a `CsrBtVcardDataTypedString` structure. Each string in the structure is zero terminated UCS-2 encoded strings
CSR_BT_VCARD_DATA_TYPE_ISO8601_DATE_TYPED	The iso8601DateTyped field holds a reference to a `CsrBtVcardDataIso8601Date` structure.

DataType	Description
CSR_BT_VCARD_DATA_TYPE_ISO8601_DATETIME_TYPED	The `iso8601DateTimeTyped` field holds a reference to a `CsrBtVcardDataIso8601DateTime` structure.
`CSR_BT_VCARD_DATA_TYPE_ORGANIZATION_UTF8`	The organization field holds a reference to a `CsrBtVcardDataOrganization` structure. Each string in the structure is zero terminated UTF-8 encoded strings
`CSR_BT_VCARD_DATA_TYPE_ORGANIZATION_UCS2`	The organization field holds a reference to a `CsrBtVcardDataOrganization` structure. Each string in the structure is zero terminated UCS-2 encoded strings

4.1.2 CsrBtVcardDataUriString

CsrBtVcardDataUriString defines details about URL/URI data.

Definition:

```
typedef struct
{
  CsrCharString * type;
  CsrCharString * string;

} CsrBtVcardDataUriString;
```

Description:

type — A string defining which type of data the structure references to. Some attributes uses predefined IANA types:

Photo:

http://www.iana.org/assignments/media-types/image/

Sound:

http://www.iana.org/assignments/media-types/audio/

Key:

“X509” / “PGP” / IANA-TOKEN

string — This string specifies the telephone number. See the specification for more details.

It is UTF-8 encoded if CSR_BT_VCARD_DATA_TYPE_TELEPHONE_UTF8 is specified or UCS-2 encoded if CSR_BT_VCARD_DATA_TYPE_TELEPHONE_UCS2 is specified.

4.1.3 CsrBtVcardDataBinary

CsrBtVcardDataBinary defines details about binary data.

Definition:

```
typedef struct
{
  CsrCharString * binaryType;
  CsrUint32   binarySize;
  CsrUint8 *  binaryData;

} CsrBtVcardDataBinary;
```

Description:

binaryType	A string defining which type of data the structure references to. Some attributes uses predefined IANA types: Photo: http://www.iana.org/assignments/media-types/image/ Sound: http://www.iana.org/assignments/media-types/audio/ Key: “X509” / “PGP” / IANA-TOKEN
binarySize	Hold the size of the data referenced by binaryData
binaryData	Reference to the raw data

4.1.4 CsrBtVcardDataAddress

CsrBtVcardDataAddress defines details about address data.

Definition:

```
Typedef struct
{
  CsrBtVcardParamType  addressType;
  CsrCharString *          poBox;
  CsrCharString *          extendedAddress;
  CsrCharString *          street;
  CsrCharString *          locality;
  CsrCharString *          region;
  CsrCharString *          postalCode;
  CsrCharString *          countryName;

} CsrBtVcardDataAddress;
```

Description:

addressType	Specify the address type. It is specified as a bit pattern and multiple bit can be set. The following types are available

```
#define CSR_BT_VCARD_PARAM_TYPE_HOME    (0x00000001)
#define CSR_BT_VCARD_PARAM_TYPE_WORK    (0x00000002)
#define CSR_BT_VCARD_PARAM_TYPE_PREF    (0x00000004)
#define CSR_BT_VCARD_PARAM_TYPE_DOM     (0x00040000)
#define CSR_BT_VCARD_PARAM_TYPE_INTL    (0x00080000)
#define CSR_BT_VCARD_PARAM_TYPE_POSTAL  (0x00100000)
#define CSR_BT_VCARD_PARAM_TYPE_PARCEL  (0x00200000)
```

Note that this type can also be an IANA type or an “X-“ type both of which is not supported.

poBox, extendedAddress, street, locality, region, postalCode, countryName	These strings specify the different element of the address. See the specification for more details. These strings are UTF-8 encoded if CSR_BT_VCARD_DATA_TYPE_ADDRESS_UTF8 is specified or UCS-2 encoded if CSR_BT_VCARD_DATA_TYPE_ADDRESS_UCS2 is specified.

4.1.5 CsrBtVcardDataName

CsrBtVcardDataName defines details about name data.

Definition:

```
typedef struct
{
  CsrCharString * familyName;
  CsrCharString * givenName;
  CsrCharString * additionalNames;
  CsrCharString * honorificPrefixes;
  CsrCharString * honorificSuffixes;

} CsrBtVcardDataName;
```

Description:

familyName, givenName, additionalNames, honorificPrefixes, honorificSuffixes	These strings specify the different element of the name. See the specification for more details. These strings are UTF-8 encoded if CSR_BT_VCARD_DATA_TYPE_NAME_UTF8 is specified or UCS-2 encoded if CSR_BT_VCARD_DATA_TYPE_NAME_UCS2 is specified.

4.1.6 CsrBtVcardDataTelephone

CsrBtVcardDataTelephone defines details about telephone data.

Definition:

```
typedef struct
{
  VcardParamType  type;
  CsrCharString *      number;

} CsrBtVcardDataTelephone;
```

Description:

type — Specify the telephone type. It is specified as a bit pattern and multiple bit can be set. The following types are available

```
#define CSR_BT_VCARD_PARAM_TYPE_HOME    (0x00000001)
#define CSR_BT_VCARD_PARAM_TYPE_WORK    (0x00000002)
#define CSR_BT_VCARD_PARAM_TYPE_PREF    (0x00000004)
#define CSR_BT_VCARD_PARAM_TYPE_VOICE   (0x00000008)
#define CSR_BT_VCARD_PARAM_TYPE_FAX     (0x00000010)
#define CSR_BT_VCARD_PARAM_TYPE_MSG     (0x00000020)
#define CSR_BT_VCARD_PARAM_TYPE_CELL    (0x00000040)
#define CSR_BT_VCARD_PARAM_TYPE_PAGER   (0x00000080)
#define CSR_BT_VCARD_PARAM_TYPE_BBS     (0x00000100)
#define CSR_BT_VCARD_PARAM_TYPE_MODEM   (0x00000200)
#define CSR_BT_VCARD_PARAM_TYPE_CAR     (0x00000400)
#define CSR_BT_VCARD_PARAM_TYPE_ISDN    (0x00000800)
#define CSR_BT_VCARD_PARAM_TYPE_VIDEO   (0x00001000)
#define CSR_BT_VCARD_PARAM_TYPE_PCS     (0x00002000)
```

Note that this type can also be an IANA type or an “X-“ type both of which is not supported

number — This string specifies the telephone number. See the specification for more details.

It is UTF-8 encoded if CSR_BT_VCARD_DATA_TYPE_TELEPHONE_UTF8 is specified or UCS-2 encoded if CSR_BT_VCARD_DATA_TYPE_TELEPHONE_UCS2 is specified.

4.1.7 CsrBtDate, CsrBtDateTime, CsrBtVcardDataIso8601Date and CsrBtVcardDataIso8601DateTime

CsrBtDate, CsrBtDateTime, CsrBtVcardDataIso8601Date and CsrBtVcardDataIso8601DateTime define details about date and time.

Definition:

```
typedef struct
{
  CsrInt32  year;
  CsrUint8  month;
  CsrUint8  day;
} CsrBtIso8601Date;

typedef struct
{
  CsrUint8  hour;
  CsrUint8  minute;
  CsrUint8  second;
  CsrUint8  fraction;
  CsrInt8   timezoneHour;
  CsrUint8  timezoneMinute;
  CsrUint8  flags;
} CsrBtIso8601Time;

typedef struct
{
  CsrBtIso8601Date date;
  CsrBtIso8601Time time;
} CsrBtIso8601DateTime;


typedef struct
{
  CsrBtVcardParamType type;
  CsrBtIso8601Date    iso8601Date;
} CsrBtVcardDataIso8601Date;

typedef struct
{
  CsrBtVcardParamType  type;
  CsrBtIso8601DateTime iso8601DateTime;
} CsrBtVcardDataIso8601DateTime;
```

Description:

year, month, day	These values specify the date.
hour,minute,second	These values specify the time.
fraction	This value indicates a time fraction. This value can be either a hour fraction, minute fraction or second fraction depending on the flags parameter.
timezoneHour, timezoneMinute	These values specify the time difference from UTC time.

flags

This value specifies which of the other values that are available, time zone information and what they specify. This is value is a bitmask and can hold multiple bits:

```
#define CSR_BT_TIME_FLAGS_HOUR_VALID         (0x01)
#define CSR_BT_TIME_FLAGS_MINUTE_VALID       (0x02)
#define CSR_BT_TIME_FLAGS_SECOND_VALID       (0x04)
#define CSR_BT_TIME_FLAGS_HOUR_FRACTION      (0x08)
#define CSR_BT_TIME_FLAGS_MINUTE_FRACTION    (0x10)
#define CSR_BT_TIME_FLAGS_SECOND_FRACTION    (0x20)
#define CSR_BT_TIME_FLAGS_UTC_TIME           (0x40)
#define CSR_BT_TIME_FLAGS_TIMEZONE_VALID     (0x80)
```

CSR_BT_TIME_FLAGS_HOUR_VALID, CSR_BT_TIME_FLAGS_MINUTE_VALID and CSR_BT_TIME_FLAGS_SECOND_VALID can help determine if hour, minute and second holds a valid value.

CSR_BT_TIME_FLAGS_HOUR_FRACTION, CSR_BT_TIME_FLAGS_MINUTE_FRACTION and CSR_BT_TIME_FLAGS_SECOND_FRACTION determine what the fraction field represents.

CSR_BT_TIME_FLAGS_UTC_TIME defines the time as being presented in UTC time and not local time.

CSR_BT_TIME_FLAGS_TIMEZONE_VALID determine if timezoneHour and timezoneMinute hold valid values.

type

Specify the date/time type. It is specified as a bit pattern and multiple bits can be set. The following types are available:

```
#define CSR_BT_VCARD_PARAM_TYPE_MISSED   (0x01000000)
#define CSR_BT_VCARD_PARAM_TYPE_RECEIVED (0x02000000)
#define CSR_BT_VCARD_PARAM_TYPE_DIALED   (0x04000000)
```

Note that this type can also be an IANA type or an “X-“ type both of which is not supported.

4.1.8 CsrBtVcardDataTypedString (CsrBtVcardDataLabel)

CsrBtVcardDataTypedString defines details about typed strings. CsrBtVcardDataLabel is included for backwards compatibility only and new implementation should use CsrBtVcardDataTypedString instead.

Definition:

```
typedef struct
{
  CsrBtVcardParamType  type;
  CsrCharString *          string;

} CsrBtVcardDataLabel;
```

Description:

type	Specify the label type. It is specified as a bit pattern and multiple bit can be set. The following types are available:
	
	`#define CSR_BT_VCARD_PARAM_TYPE_HOME (0x00000001)`
	`#define CSR_BT_VCARD_PARAM_TYPE_WORK (0x00000002)`
	`#define CSR_BT_VCARD_PARAM_TYPE_PREF (0x00000004)`
	`#define CSR_BT_VCARD_PARAM_TYPE_DOM (0x00040000)`
	`#define CSR_BT_VCARD_PARAM_TYPE_INTL (0x00080000)`
	`#define CSR_BT_VCARD_PARAM_TYPE_POSTAL (0x00100000)`
	`#define CSR_BT_VCARD_PARAM_TYPE_PARCEL (0x00200000)`
	
	Note that this type can also be an IANA type or an “X-“ type both of which is not supported.

string	This string specifies a complete address label. See the specification for more details. It is UTF-8 encoded if CSR_BT_VCARD_DATA_TYPE_LABEL_UTF8 is specified or UCS-2 encoded if CSR_BT_VCARD_DATA_TYPE_LABEL_UCS2 is specified.

4.1.9 CsrBtVcardDataOrganization

CsrBtVcardDataOrganization defines details about organization data.

Definition:

```
typedef struct
{
    CsrCharString * organizationalName;
    CsrCharString * organizationalUnit1;
    CsrCharString * organizationalUnit2;

} CsrBtVcardDataOrganization;
```

Description:

organizationalName, organizationalUnit1, organizationalUnit2	These strings specify the different element of the name. See the specification for more details. These strings are UTF-8 encoded if CSR_BT_VCARD_DATA_TYPE_ORGANIZATION_UTF8 is specified or UCS-2 encoded if CSR_BT_VCARD_DATA_TYPE_ORGANIZATION_UCS2 is specified.

Appendix A Decoding Examples

This section illustrates two examples of using the parser. The first example is the normal way of parsing data and the second way is the fast way. Both examples use some common code illustrated last.

A.1 Normal Parsing

This example provide as simple way of parsing a large document:

```
CsrBtVcardParserControl control;
CsrBtVcardData*         vcardData;
CsrUint32               vcardDataEntries;
CsrUint32               count;
CsrUint8                buffer[256];
CsrUint32               size;
FILE*                  f;

f = fopen("vcard.vcf", "r");

CsrBtVcardParseInitialize(&control,CSR_BT_VCARD_FLAGS_UCS2_STRINGS |
                                   CSR_BT_VCARD_FLAGS_ALLOW_UNSPECIFIED_ATTRIBUTE_NAMES |
                                   CSR_BT_VCARD_FLAGS_ALLOW_UNSPECIFIED_PARAMETER_NAMES |
                                   CSR_BT_VCARD_FLAGS_ALLOW_UNSPECIFIED_ENCODING |
                                   CSR_BT_VCARD_FLAGS_NO_MANDATORY_TAGS |
                                   CSR_BT_VCARD_FLAGS_NO_VERSION_MATCH);

while ((size = fread(buffer, 1, sizeof(buffer), f)) > 0)
{
  CsrBtVcardParseRead(buffer, size, &control);
}

errorCode = CsrBtVcardParseComplete(&vcardDataEntries, &vcardData, NULL, &control);

if (CSR_BT_VCARD_SUCCEEDED(errorCode))
{
  CsrBtVcardData* currentVcardData = vcardData;
  CsrBtVcardData* nextVcardData;

  while(currentVcardData)
  {
    nextVcardData = currentVcardData->next;

    DisplayVcard(currentVcardData);

    currentVcardData = nextVcardData;
  }

  CsrBtVcardFree(vcardData);
}
```

First control structure for the parser is initialized with flags that allow loose parsing. The data is then read from a file and parsed in small blocks and afterwards the **CsrBtVcardParseComplete** function is called to retrieve that data. It is then checked that the parsing was successful before the vCard data structure is enumerated and display using the **DisplayVcard** function. In the end the vCard data structure is freed with the **CsrBtVcardFree** function.

A.2 Fast Parsing

This example provides a simple way of parsing a large document:

```
errorCode = CsrBtVcardParse(buffer, size, &vcardDataEntries, &vcardData,
                            CSR_BT_VCARD_FLAGS_UCS2_STRINGS);
if (CSR_BT_VCARD_SUCCEEDED(errorCode))
{
  CsrBtVcardData* currentVcardData = vcardData;
  CsrBtVcardData* nextVcardData;

  while(currentVcardData)
  {
```

```
    nextVcardData = currentVcardData->next;

    DisplayVcard(currentVcardData);

    currentVcardData = nextVcardData;
  }

  CsrBtVcardFree(vcardData);
}
```

First the data is parsed all at once with the **CsrBtVcardParse** function. It is checked that the parsing was successful before the vCard data structure is enumerated and displayed using the **DisplayVcard** function. In the end the vCard data structure is freed with the **CsrBtVcardFree** function

A.3 Common Display Functionality

In the function **DisplayVcard** each attribute in a single vCard are now displayed by calling the function **DisplayVcardType**:

```
void DisplayVcard(CsrBtVcardData* vcardData)
{
  DisplayVcardDataType(vcardData->formattedName);
  DisplayVcardDataType(vcardData->vcardName);
  DisplayVcardDataType(vcardData->name);
  DisplayVcardDataType(vcardData->nickNames);
  DisplayVcardDataType(vcardData->photo);
  DisplayVcardDataType(vcardData->birthDate);
  DisplayVcardDataType(vcardData->source);
  DisplayVcardDataType(vcardData->address);
  DisplayVcardDataType(vcardData->label);
  DisplayVcardDataType(vcardData->tel);
  DisplayVcardDataType(vcardData->email);
  DisplayVcardDataType(vcardData->mailer);
  DisplayVcardDataType(vcardData->timezone);
  DisplayVcardDataType(vcardData->globalPosition);
  DisplayVcardDataType(vcardData->jobTitle);
  DisplayVcardDataType(vcardData->jobRole);
  DisplayVcardDataType(vcardData->jobLogo);
  DisplayVcardDataType(vcardData->jobAgent);
  DisplayVcardDataType(vcardData->organization);
  DisplayVcardDataType(vcardData->catagories);
  DisplayVcardDataType(vcardData->note);
  DisplayVcardDataType(vcardData->productIdentifier);
  DisplayVcardDataType(vcardData->revision);
  DisplayVcardDataType(vcardData->sortString);
  DisplayVcardDataType(vcardData->sound);
  DisplayVcardDataType(vcardData->uid);
  DisplayVcardDataType(vcardData->url);
  DisplayVcardDataType(vcardData->classification);
  DisplayVcardDataType(vcardData->key);
  DisplayVcardDataType(vcardData->xIrmcCallDatetime);
}
```

The **DisplayVcardDataType** function enumerates the attribute list. It assures that the reference is actually present in the vCard. The payload is then displayed depending on its **dataType**. In this example only CSR_BT_VCARD_DATA_TYPE_STRING_UTF8 and CSR_BT_VCARD_DATA_TYPE_STRING_UCS2 are decoded.

```
void DisplayVcardDataType(CsrBtVcardDataType* vcardDataType)
{
  CsrBtVcardDataType* nextVcardDataType;

  while (vcardDataType)
  {
    nextVcardDataType = vcardDataType->next;

    switch (vcardDataType->dataType)
    {
      case CSR_BT_VCARD_DATA_TYPE_STRING_UTF8:
      {
        printf ("%s%s\n", name, vcardDataType->payload.string);
        break;
      }
      case CSR_BT_VCARD_DATA_TYPE_STRING_UCS2:
      {
        printf ("%s", name);
        wprintf (L"%s\n", vcardDataType->payload.string);
        break;
      }
      ...
    }
    vcardDataType = nextVcardDataType;
  }
}
```

Appendix B Appendix B – Generation Example

This section illustrates an example of using the generator:

```
CsrBtVcardGeneratorControl control;
CsrBtVcardData*            vcardData;
CsrUint32                  bufferOffset;
CsrUint8                   buffer[256];
CsrUint32                  bufferSize;
CsrUint32                  result;
FILE*                     f;

f = fopen("vcard.vcf", "w");

CsrBtVcardGeneratorInitialize(&control, CSR_BT_VCARD_FLAGS_UCS2_STRINGS |
                                        CSR_BT_VCARD_FLAGS_VERSION_3_0);

bufferOffset = 0;
bufferSize   = sizeof(buffer);

if (GenerateVcard(&vcardData) == TRUE)
{
  do
  {
    while (bufferOffset < bufferSize)
    {
      result = CsrBtVcardGeneratorWrite(vcardData, &buffer[bufferOffset],
                                        bufferSize - bufferOffset, &control);
      if (result == 0xFFFF)
      {
        break;
      }
      else if (result + bufferOffset != bufferSize)
      {
        if (GenerateVcard(&vcardData) == FALSE)
        {
          break;
        }
      }
    }

    fwrite(buffer, 1, bufferOffset, f);

  } while (bufferOffset < bufferSize);
}
```

First control structure for the generator is initialized with flags that generate vCard v3.0 output based of a vCard data structure using UTF-16 / UCS-2 strings. The data structure is generated using the **GenerateVcard** function and is then the vCard document is generated with the **CsrBtVcardGeneratorWrite** function. Once the buffer is filled it is written to a file. This example does not specify how the vCard data structure is generated or freed.

5 Document References

Document	Reference
Specification of the Bluetooth System Version 1.1, 1.2 and 2.0	[BT]

Terms and Definitions

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
CSR	Cambridge Silicon Radio
UniFi™	Group term for CSR's range of chips designed to meet IEEE 802.11 standards

Document History

Revision	Date	History
1	26 SEP 11	Ready for release 18.2.0

TradeMarks, Patents and Licences

Unless otherwise stated, words and logos marked with ™ or ® are trademarks registered or owned by CSR plc or its affiliates. Bluetooth® and the Bluetooth logos are trademarks owned by Bluetooth SIG, Inc. and licensed to CSR. Other products, services and names used in this document may have been trademarked by their respective owners.

The publication of this information does not imply that any licence is granted under any patent or other rights owned by CSR plc.

CSR reserves the right to make technical changes to its products as part of its development programme.

While every care has been taken to ensure the accuracy of the contents of this document, CSR cannot accept responsibility for any errors.

Life Support Policy and Use in Safety-critical Compliance

CSR's products are not authorised for use in life-support or safety-critical applications. Use in such applications is done at the sole discretion of the customer. CSR will not warrant the use of its devices in such applications.

Performance and Conformance

Refer to www.csrsupport.com for compliance and conformance to standards information.