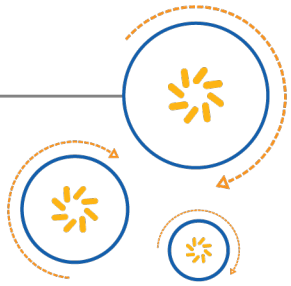
Qualcomm Technologies International, Ltd.



ADK Application Configuration System

User Guide

80-CT548-1 Rev. AE

October 26, 2017

Qualcomm Technologies International, Ltd. (formerly known as Cambridge Silicon Radio Limited) is a company registered in England and Wales with a registered office at: Churchill House, Cambridge Business Park, Cowley Road, Cambridge, CB4 0WZ, United Kingdom.
Registered Number: 3665875 | VAT number: GB787433096

Revision history

Revision	Date	Description
1	29 March 2017	Initial release. Alternative document number CS-00400610-UG.
2	30 March 2017	Minor editorial changes
3	21 April 2017	Minor editorial changes
4	25 April 2017	Minor corrections
AE	October 2017	Added to the Connect Management System. DRN updated to use Agile number. No change to the technical content.

Contents

Tables

Figures

1 ADK Application Configuration System

The ADK Application Configuration System is used with the ADK Configuration Tool, to differentiate a device by modifying its Configuration Set Item values during product development.

This document describes how to:

- Change configuration data
- Add or remove Configuration Items
- Add a new module to the Application

NOTE The terminology used in this document is described in the *ADK Applications Configuration Architecture Overview* document and it is assumed that the reader is familiar with this document.

1.1 Configuration process overview

Application Configuration includes three main processes

1. Initial build and Flashing
2. Modification of the Configuration Set
3. Accessing or Modifying the Configuration Set data

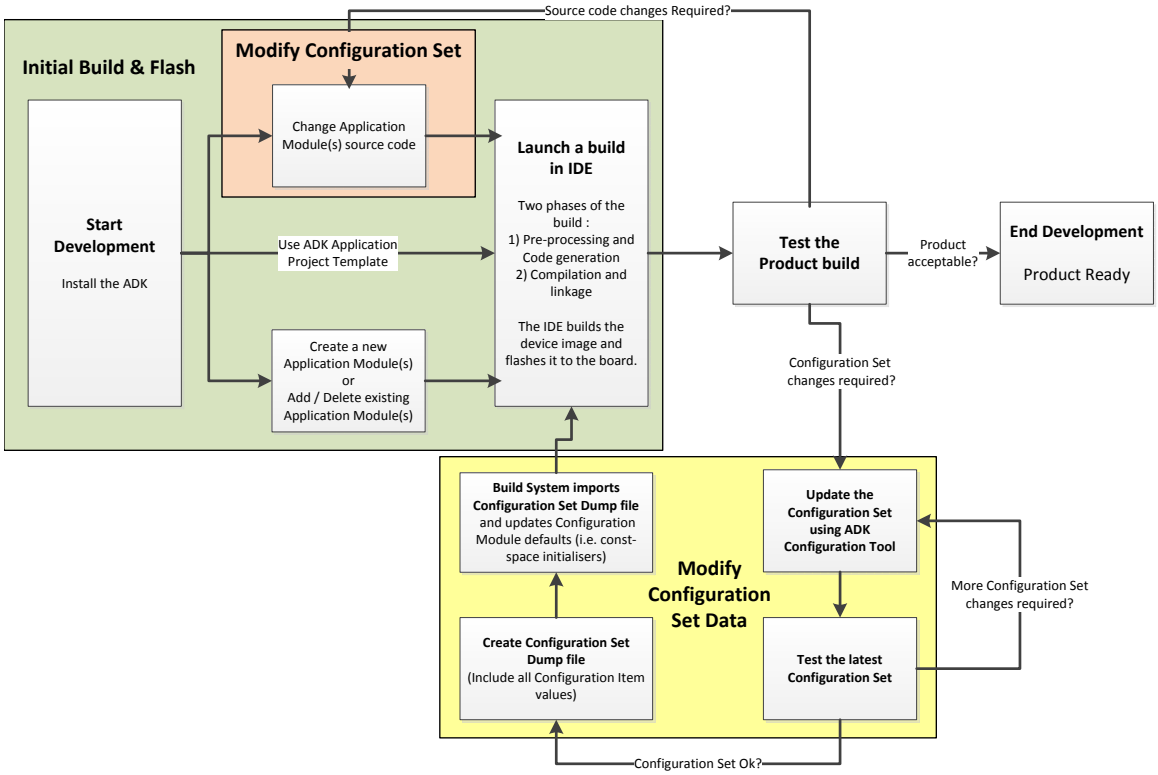


Figure 1-1 Application Configuration architecture: development process

2 Build an ADK application

The Configuration Build Tools are fully integrated into the ADK build system and a partial or full rebuild of the Application is only required if configuration data is modified, added or removed.

2.1 ADK hardware and software variants

By default, Configuration Item values exist in an XML file and are validated by the Build Tools; values can be customized and over-written as required, to suit a particular hardware or software variants.

For example, Configuration Item values can be modified to:

- Override the PIO mappings for the same Application running on different hardware platforms.
- Override the default volume levels for different Applications running on the same hardware platform.

NOTE CSRA68100 platform hardware and software variants default to “H13672v2” and “ADK6-Sink” respectively. CSR867x based platform software does not have a default value and a value is set in the project properties, before a new project is built for the first time.

2.2 Flash an ADK application

All other chip sub-systems must be flashed on to the CSRA68100 device, before the Application is flashed.

NOTE This guide only considers Applications and does not describe any other device sub-system. The processes described assume that all other sub-systems are installed correctly.

To update the Application and associated Configuration Set data, deploy the following projects:

Projects	Description
Sink	■ Contains the Sink Application with built-in default Configuration Set values
Customer_ro_filesystem	■ Contains the Configuration Set Definition as an `.xml` file, used by the Application Configuration Tool.
user_ps_filesystem	■ Deployed the first time an Application is deployed, to erase any modified Configuration Set values and reset all Configuration Set values back to their default settings

2.3 Modify ADK application functionality

Adding or removing functionality from the Application, for example enabling IR remote functionality, may increase or decrease the number of Configuration Blocks in the overall Configuration Set.

The Configuration Build Tools use a map of Configuration Block Id to PS Key to ensure that if the Configuration Set has changed, each Configuration Block will continue to map to the same PS Key. Because of this, it is not necessary to reset the Device Configuration to the defaults each time the Application functionality is changed.

2.3.1 Resolve ADK Configuration Set changes

This step is optional but recommended.

To ensure that the Tool reads fresh, valid Configuration Set data if a device Configuration Set has changed, either Save or discard the new values before deploying the new Application.

To save the current Configuration Set values and apply them to the new Application:

1. Use the ADK Configuration Tool to save the current Configuration Set values, see Save ADK configuration current values.
2. Update the Application default Configuration Set values using the Configuration Merge Tools, see Update ADK application default values.
3. Reset the device to the default Configuration Set values, see Reset device to default values
4. Build and deploy the new Application.

Or, to discard any existing Configuration Set data changes and use the values from the new Application:

1. Reset the device to the default Configuration Set values, see Reset device to default values.
2. Build and deploy the new Application.

3 Modify the ADK Application Configuration

It is possible to modify ADK Application Configuration Set values in six different ways, without changing the structure of the Configuration Set, by:

1. Changing Default Values
2. Changing Values at Runtime
3. Saving Current Values
4. Restoring Saved Values
5. Updating Default Values
6. Resetting the device to Default Values.

NOTE Changing the structure of the Configuration Set always requires the Application to be rebuilt, see Configuration process overview.

3.1 ADK Config Group Default Values

The XML `ConfigGroup` element defines both the data structure and a set of default values for each Configuration Item; the Application is rebuilt using these new values.

Change the Configuration Set default values to:

- Make semi-permanent changes when developing an Application.
  This avoids the need for the ADK Configuration Tool to re-enable values, each time a new device is flashed.
- Simplify the flashing process for multiple devices, set the final product configuration in the default values.
- Reduce run-time memory usage.
  - Configuration changes made by the ADK Configuration Tool require extra memory allocations to maintain a runtime copy of the data. This can be a limiting resource. The use of modified default values help to reduce both the number and size of memory allocations, reducing system demands.

NOTE The sink module configuration XML is located in **<ADK>\apps\sink \module_configurations**.

3.1.1 ADK Default Value parameters

Configuration Build tools validate the Default Values, which must conform to certain parameters.

Table 3-1 Default Type value

Type	Parameter
Integer Values	Must not be larger than the maximum number allowed, given the number of size bits they have.
	Start with '0x' - Value is a Hexadecimal Not start with '0x' - Value is an Integer
Enumerated values	Must be either the ASCII text representation of an enum value or the equivalent integer value.
Bitfield values	Combined using the '|' operator. For example: DefaultValue="Faststream | Qualcomm® aptX™".

3.1.2 ADK application Global Configuration XML files

It is important to know where the enum is defined for enumerated types.

System-wide definitions are defined in `global_config.xml` and the Configuration Build Tools checks:

- Application user events enumeration
- Application system events enumeration
- PIO pin enumeration

NOTE Any ADK Configuration Tool change always takes precedence over the default values. Resetting the Configuration Set on the device restores the default values. See Reset device to default values.

See the *ADK Build Scripts XML Definitions User Guide,* which describes the Configuration Set XML format.

3.2 Change ADK configuration values at runtime

The ADK Configuration Tool is used to change the Configuration Set values on a device that has already been flashed and is running.

Changing the Configuration Set at runtime can be useful for:

- Testing small changes, such as button mappings, tone prompts and so on.
- Interactively setting up new Configuration Item values, for example, setting the I2C initialization commands for a new external amplifier chip.
- Fine-tuning Configuration Item values, before setting a value as the default in the Configuration Set XML.

NOTE Any modified Configuration Item value is saved to the PS store so, if all user PS keys are deleted, the default Configuration Set values built into the Application are used. Refer to the *ADK Configuration Tool* User Guide for more information.

To change a Configuration Set Item value at runtime:

1. Select the device in the Device List drop-down menu.
2. Click **Go Configurable**.
   The device transitions from Normal AHI Mode and reboots in Configurable AHI mode and displays the Configurable Mode LED pattern. It is only possible to Read, modify or Write Configuration Set values when the Application is in Configurable AHI Mode.
3. Click **Read Device** to get the current Configuration Set from the device.
4. Use the Configuration Set Tree View to identify the Configuration Items to modify.
   The 'Filter' function is used to highlight specific Configuration Items.
5. Edit the Configuration Item as required.
6. Click the **Write Device** to write the changed Configuration Set back to the device.
   The device stops displaying the Configurable Mode LED pattern.
7. Either:
   a. Click the **Go Normal** to place the device into **Normal AHI mode**, or
   b. To put the device back into the "Normal" running mode, **Disconnect** the device.

3.3 Save ADK configuration current values

The ADK Configuration Tool is used to save the Configuration Set as an `.xml` Dump file that defines both the structure and default value of each Configuration Item in the Configuration Set.

Configuration Set Dump files are useful for:

- Saving the Configuration Set after the device has been interactively configured.
- Transferring a Configuration Set from one device to another, running the same Application image.
- Updating the Application default Configuration Set in the Module Configuration XML files.
  See the *ADK Configuration Tool User Guide* for more information.

3.4 Restore ADK application saved values

As restoring values to the device effectively over-writes the current values, to confirm that its binary layout and structure matches those on the device, the ADK Configuration Tool validates each Configuration Set Dump file before commencing the restore.

NOTE If the files are different, the restore fails and a warning is returned.

It is not possible to restore individual Item values; Restore is only able to Write All Item values.

See the *ADK Configuration Tool User Guide* for more information.

3.5 Update ADK application default values

The ADK Configuration Tool can be used to update the default values for the Application from a saved Configuration Set Dump `.xml` file.

See the *ADK Configuration Tool User Guide* for more information.

The updated default values will be applied to the Application the next time it is built and deployed onto the device.

3.6 Reset device to default values

Deleting all Config Store PS Keys removes any changes made to the device configuration at runtime and returns all Items to their default value.

To reset all Item values on a CSRA68100 device, deploy the `user_ps_filesystem` project, which resets all PS Keys between 5 and 99: `PSKEY_USR5-PSKEY_USR99`.

4 ADK Configuration Set Build Tools

The ADK Configuration Set Build Tools are fully integrated into the Application build system and run automatically when the project is built.

The Configuration Build Tools:

- Generate the binary configuration data and C code from the input module configuration and global XML Files.
- Validate the input XML, by checking for duplicate configuration elements and bounds checking on default values.
- Map configuration groups to PS keys to.
  - Define the Configuration Set structure
  - Generate the Configuration Definition XML file, which is used by the ADK Configuration tool. See the *ADK Configuration Tool User Guide*.

4.1 ADK Build integration

The Configuration Set tools Build rules are contained in the `config_build.mak` file, which is located in the Application root directory.

Based on the project build settings, `Config_build.mak` performs a number of roles, which include:

- Collecting the set of input Module Configuration XML files.
- Calling the Configuration Set build tools with the appropriate parameters.
- Copying the Built tool-generated Configuration Definition XML file into the `customer_ro_filesystem` Read-only directory.

NOTE The contents of the `customer_ro_filesystem` directory is flashed onto the Device, by deploying the `customer_ro_filesystem` project.

Older CSR867x platforms copied the Configuration Definition file into the internal file system directory, which was automatically included when the project was flashed onto the device, in xIDE.

4.1.1 ADK Configuration Set PS Keys

The Configuration Build tools reserve PSKEY_USR5 to PSKEY_USR99 for use by the Configuration Store library, while the Application uses PSKEY_USR0 to PSKEY_USR4.

The Configuration Store library monitors the state of the individual Configuration Blocks.

NOTE To avoid the risk of the Configuration Store becoming undefined, Application code must never directly access any PS Key assigned to the Configuration Store.

Change Configuration store PS Key range

The range of user PS Keys reserved for use by the Configuration Store is changed by editing the call to the Configuration Build tools, in `config_build.mak`; any existing configuration on the Device becomes invalid.

Configuration Set Item default values must be reset before flashing a new Application onto the device.

NOTE If the input XML contains more Configuration Groups than reserved user PS Keys, then the Configuration Build tools are unable to map each Group directly to a unique PS Key and an error is returned.

To change the range of user PS Keys reserved for the Configuration Store:

1. Open the `config_build.mak` file located in the **<ADK>\apps\sink** directory, for editing.
2. Change the range of user PS Keys option `-psk 5-49,150-199`.
   PS Key options are located in the call to `$(CONFIG_BUILD_EXE_CMD)`, in the `config_build` target.

   NOTE The `-psk` option defines the PS Key index in terms of the value passed into the PS store firmware API. For legacy reasons, PSKEY_USR0 to PSKEY_USR49 are 0-49 and PSKEY_USR50 to PSKEY_USR99 are 150-199.
3. Rebuild the Application.
4. Reset the Configuration Set Item values on the device, by following the steps in Reset device to default values.
5. Flash the new Application back onto the device.

4.2 ADK Build artifacts

Configuration build errors

The Configuration Build Output Log, `config_build.log` provides detailed processing error log information and can be useful for de-bugging errors, when generating an output file.

Identify Configuration Block PS Key allocation

The Configuration Build PS Key Map, `pskey_map.xml`, is used to identify which PS Keys are in use by the Configuration Block.

To identify which PS Keys are allocated for use by the Configuration Block:

1. Identify the Configuration Group ID of interest. For example, **A2DP Config** for the main A2DP Configuration Block.
2. Search `pskey_map.xml` for the ID.
   The allocated PS Key (6), appears in the same ConfigGroup element, for example:

```
<ConfigGroup
    Active="True"
```

```
PsKeyNumber="6"
Id="A2DP Config" />
```

Find the Configuration Block Binary Defaults

The Configuration Set defaults for the Application are stored in the `config_defnition.c` file, where the defaults for a single Configuration Block are held within a `CONFIG_DATA_CONFIG_BLOCK_DATA_TYPE` block.

To find the Binary Defaults for a Configuration Block:

1. Find the Configuration Block PS Key.
2. Search for the `CONFIG_DATA_PS_CONFIG_BLOCK_DATA_TYPE` that contains the same PS Key. The default data is found in the previous `CONFIG_DATA_CONFIG_BLOCK_DATA_TYPE` section. Using the example of “A2DP Config”, which uses PS Key 6:

```
(CONFIG_DATA_CONFIG_BLOCK_TYPE << 12) | 5,
    (CONFIG_DATA_CONFIG_BLOCK_DATA_TYPE << 12) | 2,
    0x003c, 0x32c0,
    (CONFIG_DATA_PS_CONFIG_BLOCK_DATA_TYPE << 12) | 1,
    6,
```

The Binary defaults are: `0x003c, 0x32c0`.

NOTE The binary data is stored as an array of unsigned 16-bit hextets, since this is the format used by the PS Key store.

5 ADK application Configuration Set data

Only access Configuration data using the Configuration Store library API, located in the `config_store.h` file.

5.1 ADK Config Store library

Configuration Blocks can be opened as either Read-only or writable, by using separate API functions; Read-only uses fewer blocks.

When using the Configuration Store API, it is important to:

- Always define the Configuration Block size as the number of unsigned 16-bit hextets it contains.
- Ensure that the Configuration Store library data in the block is consistent, between the runtime and stored versions.
  It is only possible for a Configuration Block to be opened by a single client, at any one time.

See the *ADK Applications Configuration Architecture Overview* for more information about the Configuration Store

5.2 ADK application Configuration Block data

The Configuration Build Tools generate C header files that define data types, which are used to access the Binary data returned by the Configuration store.

See the *ADK Applications Configuration Architecture Overview document*, for more information.

For example: For the `sink_a2dp` module:

- If sink_a2dp_module_def.xml is the input
- Then `sink_a2dp_config_def.h` is generated as a header file.

The following code sample shows an extract from the `sink_a2dp_config_def.h` file, which includes the `#define` for a Configuration Block ID and related C structure definition:

```
#define A2DP_CONFIG_BLK_ID 33
typedef struct {
    unsigned short A2dpLinkLossReconnectionTime_s;
    unsigned short padding:6;
    unsigned short EnableA2dpMediaOpenOnConnection:1;
    unsigned short EnableA2dpStreaming:1;
    unsigned short a2dpclass:2;
```

```
    unsigned short A2dpOptionalCodecsEnabled:6;
} a2dp_config_def_t;
```

NOTE Before it is used, any data from the `A2DP_CONFIG_BLK_ID` block is cast from a `(void *)` pointer to a `(a2dp_config_def_t *)` pointer.

5.2.1 ADK read-only access

Read-only access uses fewer memory allocations than a writable access and is used to optimize device performance.

To open a Configuration Block as Read-only:

1. Call:
   `ConfigStoreGetReadOnlyConfig(config_blk_id_t id, uint16 *config_data_size, const void **data)`

   Where:

`config_blk_id_t`	The block ID for the Block to open, from the generated header. For example, `A2DP_CONFIG_BLK_ID` opens the a2dp configuration data.
`config_data_size`	The size of the Configuration Block. Size is the number of unsigned 16-bit hextets in the Configuration Block.
`data`	A pointer to the Configuration Block data. The Configuration Store library controls all memory allocations. If the Configuration Block only contains default values, then no extra buffer is allocated for Configuration Block data. `data` points to a location in the `const` memory space.

2. Using the structure defined in the generated header; Read one or more values from the Configuration Block
   When access is no longer required:
3. Release the Block by calling:
   `ConfigStoreReleaseConfig(config_blk_id_t id)`

NOTE To avoid the small extra overhead required to repeatedly open and close frequently used Configuration Blocks, such as button mappings, the Block can be opened once during Application start-up and kept open for the duration of runtime.

5.2.2 ADK writeable access

NOTE Writable access incurs more overheads than Read-only; only open Blocks as Writable if there is an actual need to modify the content.

To open a fixed-length Configuration Block as Writable (For variable-length Configuration Blocks see ADK application Variable length arrays):

1. Call:
   `ConfigStoreGetWriteableConfig(config_blk_id_t id, uint16 *config_data_size, void **data)`

Where:	
config_blk_id_t	The #define value for the Block to be opened, obtained from the generated header file. For example, A2DP_CONFIG_BLK_ID for the a2dp configuration data.
config_data_size	Either: □ Set with the buffer size that the Configuration Store should allocate for the configuration data, or □ 0 – To allocate a buffer of the exact size of the stored configuration data.
data	A pointer to the configuration data The Config Store library allocates a memory buffer to hold the writable copy of the configuration data.

2. Modify the configuration data as required.
3. Call ConfigStoreWriteConfig (config_blk_id_t id) to write the contents of the modified buffer back into the Config Store.
4. To release the Configuration Block, call: ConfigStoreReleaseConfig(config_blk_id_t id).

5.2.3 ADK application Variable length arrays

At runtime, it is necessary to calculate the length of a variable-length Configuration Block before it is Read or modified.

Number of items

The number of items in a variable-length array is not stored in the Configuration Block itself, instead the number of items are calculated by dividing the array size by the size of one element.

For example, to calculate the size of the Configuration Block in words:

1. To convert the value from unsigned 16-bit hextets, multiply the size returned by the Config Store API by sizeof(uint16).
2. Divide the size in words by the sizeof() of the first element in the generated C structure.

Variable length array code example

The following code samples demonstrate calculations for the LED pattern to sink event mapping array SINK_LED_EVENT_PATTERN_CONFIG_BLK_ID 757.

```
#define SINK_LEDEVENT_PATTERN_CONFIG_BLK_ID 757
typedef struct {
    LEDPattern_t pEventPatterns[1];
} sink_led_event_pattern_config_def_t;
```

The following code extract demonstrates calculation of the number of elements in the SINK_LED_EVENT_PATTERN_CONFIG_BLK_ID Configuration Block:

```
uint16 i;
uint16 size;
uint16 num_patterns;
```

```
sink_led_event_pattern_config_def_t *event_pattern;
size = configManagerGetReadOnlyConfig(SINK_LED_EVENT_PATTERN_CONFIG_BLK_ID,
(const void **)&event_pattern);
if, size)
{
    num_patterns = ((size * sizeof(uint16)) / sizeof(event_pattern-
>pEventPatterns[0]));
    …
    configManagerReleaseConfig(SINK_LED_EVENT_PATTERN_CONFIG_BLK_ID);
}
```

Add elements at run-time

QTIL does not recommend adding elements to the Configuration Store at run-time, since the Store is optimized for read-only accesses; always try to finalize the configuration data before releasing the product to customers.

To add an element:

1. If the number of elements in the array is not already known: Calculate the number of elements.
2. Close the block after performing the calculation.
3. Specify a Block size that is large enough to hold the existing number of elements plus an extra one, for example:

   ```
   size = ((num_patterns + 1) * sizeof(event_pattern->pEventPatterns[0]));
   ```

4. Re-open the Block.
5. Set the value of the final element.
6. Call: `ConfigStoreWriteConfig (config_blk_id_t id)`, to write the contents of the modified buffer back into the Configuration Store.
7. Call: `ConfigStoreReleaseConfig(config_blk_id_t id)`, to release the Configuration Block.

Removing elements

It is only possible to remove individual elements from the Configuration Block if the Configuration block is not in use (Opened), by any other Application Module.

To remove an element from the Configuration Block:

1. Open the Configuration Block as Read-only.
2. Calculate the number of elements.
3. Allocate a temporary buffer to store the current configuration data.
4. Copy the current configuration data into the temporary buffer.
5. Close the Read-only Configuration Block.
6. Set the Block size to be one element smaller than current size, for example:

   ```
   size = ((num_elements – 1) * sizeof(element));
   ```

7. Re-open the Configuration Block as writable.

8. Except for the element that is being removed, copy the other elements from the temporary buffer into the writeable Configuration Block.
9. Call: `ConfigStoreWriteConfig (config_blk_id_t id)` to write the contents of the modified buffer back into the Config Store.
10. Call: `ConfigStoreReleaseConfig(config_blk_id_t id)` to release the Configuration Block.
11. Free the temporary buffer.

5.2.4 ADK application Text strings

Text strings are stored in the Configuration Store as arrays of 8-bit ASCII values, packed into the underlying array of 16-bit hextets.

Where:

- Each hextet contains two 8-bit ASCII characters
- A Nul (0x00) character is added to the end of any string containing an odd-number of characters.

NOTE Strings are not always Nul terminated, the length of the string must be calculated using the size of the Configuration Block containing the string.

The `byte_utils` library provides utility functions for packing strings-to and unpacking strings-from a Configuration Block. Utility functions are able to handle C-style strings from both CSR867x and CSRA68100 platforms.

Read string Example

The code sample shows how a string is Read from the Configuration Store.

To Read a string from the Configuration Store:

```
uint16 len_words = 0;
uint16 len_bytes = 0;
sink_hfp_data_writeable_config_def_t *write_data = NULL;
len_words =
configManagerGetReadOnlyConfig(SINK_HFP_DATA_WRITEABLE_CONFIG_BLK_ID, (const
void **)&write_data);
if (len_words)
{
    /* Find length of phone number string and unpack. */
    len_bytes = ByteUtilsGetPackedStringLen((const uint16*)write_data-
>phone_number, len_words);
    ByteUtilsMemCpyUnpackString((uint8 *)phone_number, (const
uint16*)write_data->phone_number, len_bytes);
    configManagerReleaseConfig(SINK_HFP_DATA_WRITEABLE_CONFIG_BLK_ID);
}
```

Write string Example

The code sample shows how a string is written to the Configuration Store.

To write a string to the Configuration Store:

```
sink_hfp_data_writeable_config_def_t *write_data = NULL;
if (configManagerGetWriteableConfig(SINK_HFP_DATA_WRITEABLE_CONFIG_BLK_ID,
(void **)&write_data, 0))
{
    memset(write_data->phone_number, 0, sizeof(write_data->phone_number));
    ByteUtilsMemCpyPackString((uint16 *) write_data->phone_number,
phone_number, size_phone_number);

configManagerUpdateWriteableConfig( SINK_HFP_DATA_WRITEABLE_CONFIG_BLK_ID);
    configManagerReleaseConfig(SINK_HFP_DATA_WRITEABLE_CONFIG_BLK_ID);
}
```

A ADK application Configuration transports

During product development, the ADK Configuration Tool is able to communicate with the device through the Application Host Interface (AHI) protocol. This is not supported for End-user customer use cases and must be disabled before shipping finished products.

The Application supports the AHI protocol over the following transports:

- USB HID Generic:
  - A USB HID Generic interface that uses the standard device USB port, used for; charging, audio or other USB functionality.
- USB SPI:
  - An SPI-based transport, routed via the CNS10020 USB to SPI converter, which uses the USB driver to pass SPI commands to the device.

A.1 Select AHI Transport for ADK applications

The Application can only support one AHI transport at any one time and defaults to the USB HID Generic transport; the USB SPI transport is set at Application build-time, using a C macro definition.

Macro setting is platform-dependent:

- CSRA68100 devices only support the USB HID Generic AHI transport
- CSR867X devices set the AHI transport in the Project Properties.
  - In xIDE set **Project Properties > Build System > AHI Transport** to one of the following:
    - **USB** - For USB HID Generic transport.
    - **SPI** - For USB SPI transport.
    - **Disabled** - All AHI transports are switched off.

NOTE If all AHI transports are disabled, the ADK Configuration Tool is not able to connect to the device.

A.2 Disable AHI Transport in production

The ADK Configuration Tool is only intended for use during product development and must be disabled in all production devices.

The AHI transport is disabled using either of the following methods:

1. Irrevocably, at Application build time,, see Appendix Select AHI Transport for ADK applications)
2. On the Production Line: By setting the first word of `PSKEY_USR1 = 0000`.
   This is a one-way command that causes the device to reset with the USB HID Generic AHI transport disabled and unavailable.

NOTE For CSR867X platforms, either disable the AHI Transport:

- Using the AHI protocol itself, or
- Manually, using the PS Tool or pscli.

Document references

Document	Reference
ADK Build Scripts XML Definitions User Guide	80-CT541-1/CS-00346862-UG
ADK Configuration Tool User Guide	80-CT554-1 /CS-00309942-UG
ADK Applications Configuration Architecture Overview	80-CU111-1/CS-00400589-TO

Terms and definitions

Term	Definition
ACA	Applications Configuration Architecture
Application Module	A single part of the Application targeted at a particular purpose or feature. Application Modules implement an explicit public interface and encapsulate state data. They may, or may not, use configuration data. For example; The ADK Sink Application, Application Modules are often comprised of a single compilation unit, that is, a single .c implementation file.
Configuration Item	Any single item of non-volatile configurable data used by an Application Module.
Configuration Block	A block of Configuration Items belonging to a specific Configuration Module. Configuration Items are normally grouped into Configuration Blocks to represent logical or functional relationships between the Configuration Items.
Configuration Module	The set of Configuration Blocks owned and encapsulated by a single Application Module.
Configuration Set	The set of all the Configuration Modules used by the Application Modules included within a specific Application build.