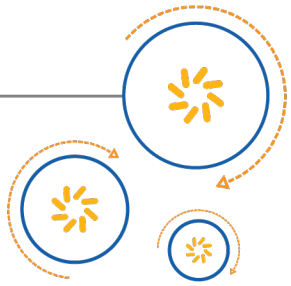
Qualcomm Technologies International, Ltd.



ADK Application Configuration Architecture

Reference

80-CU111-1 Rev. AE

November 10, 2017

Revision history

Revision	Date	Description
1	05 APRIL 2017	Initial release. Alternative document number CS-00400589-TO.
2	05 APRIL 2017	Minor editorial changes.
3	21 APRIL 2017	Minor editorial changes.
4	25 April 2017	Minor corrections.
AE	November 2017	Minor editorial changes. No changes to technical content. Migrated to CMS.

Contents

Tables

Figures

1 ADK Application Configuration Architecture

This document introduces the ADK Application Configuration Architecture (ACA). It provides an overview of the key concepts and explains:

- How the Application is structured to make use of configuration data encapsulated in modules
- How configuration data is generated, dimensioned and initialized in response to specific Application requirements
- How the configuration data is used by and shared among the Application Modules.
- How a Developer can modify configuration data using the ADK Configuration Tool.

NOTE The ACA described in this document only applies to Application operation and configuration data traditionally stored in User PS keys. Non-User PS keys are unaffected and outside the scope of the ACA.

1.1 Terminology

The following table defines the terminology used in the context of ADK Application Configuration Architecture.

Phrase	Meaning
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.

1.2 Summary: ADK Application Configuration Architecture

Unlike previous ADKs that employed a static, monolithic configuration architecture, the ADK Configuration Set structure is dynamic and re-defined at each build.

In the ADK Application Configuration Architecture:

- Fixed User PS key addresses are no longer reserved for specific Application features or functionality.
- There is no longer a single, monolithic Configuration Set definition for the Application. Therefore, there is now no explicit versioning of the Application Configuration Set.
- The default value for each Configuration Item is now stored in the Application, in const-space. Therefore, `.psr` files are no longer needed to initialize or used to configure User PS keys. When a Configuration Item is changed from its default value, this configuration setting is stored in a dynamic configuration store. The physical medium for this storage is abstracted from the Application, and need not necessarily be implemented by a User PS key.

The Application configuration architecture contains the following new components that play key roles in the system:

- Application Host Interface (AHI):
  - The AHI is implemented in two parts, as a DLL on the host and a library on the device. The AHI allows the host to:
    - Access and modify the Configuration Set on the device using the ADK Configuration Tool.
    - Control the execution state of the Application on the device.
- Config Store:
  - Config Store is a new library, which is used by Application Modules to access their configuration data. Config Store abstracts concern of the physical non-volatile storage from the Application Module.

2 Software architecture

The following sections describe the elements and concepts of the ADK Application configuration architecture:

- ADK Application.
- ADK libraries.
- ADK Configuration Tool.

2.1 ADK Application

In ADK, the Application has been refactored to achieve the following goals:

- To partition the Configuration Set so that visibility of Configuration Items is limited to only the parts of the Application that need to access this data.
- To formalize access to the Configuration Set through a new API.
- To abstract away the physical implementation of the non-volatile storage used.
- To reduce the run-time memory requirements for storing configuration data.

2.1.1 Modularization in the ADK Application

The primary refactoring applied in the ADK Application has been the partitioning of functionality into Application Modules and the encapsulation of the relevant parts of the Configuration Set into distinct Configuration Modules.

Configuration data is now owned by individual Application Modules and can only be directly accessed by that Application Module. However, configuration data can still be shared between Application Modules if needed. This is achieved by the Application Module controlling access to the configuration data it encapsulates by providing an appropriate API publicly to other Application Modules.

NOTE An Application Module may not use configuration data. However, if an Application Module does use configuration data, this data is referred to as the *Configuration Module*.

Therefore, the Configuration Set can be decomposed in levels of granularity into Configuration Modules, then once again into individual Configuration Items.

Configuration Set hierarchy

Figure 2-1 shows the composition of the Configuration Set and how it relates to the Application Modules.

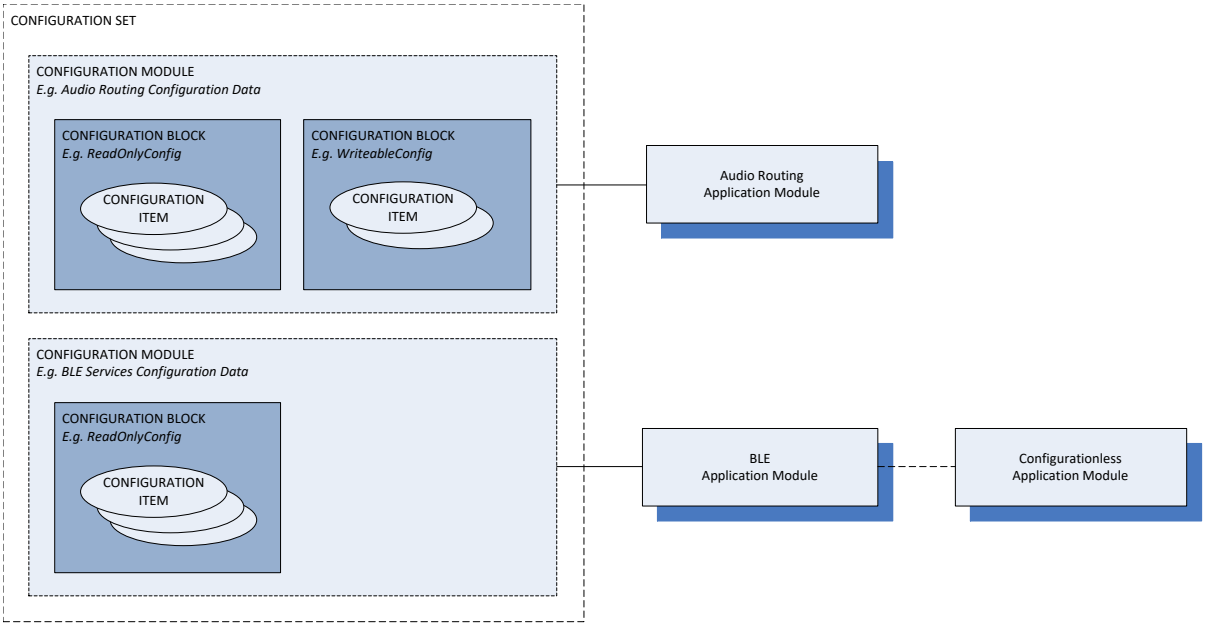


Figure 2-1 Configuration Set structure

The Configuration Set is decomposed as follows:

- The **Configuration Set** is composed of at least one or more **Configuration Modules**.
  - Configuration Modules belong to a single Application Module and are composed of one or more Configuration Blocks.
    - Configuration Blocks are composed of one or more Configuration Items.

NOTE Configuration Blocks can be used to group together Configuration Items that will be accessed in a particular way for reasons of resource efficiency. For example, the developer may wish to collect writeable Configuration Items together into a dedicated “writeable” Configuration Block to save Persistent Store and memory pool resources while using writeable data. This improves the efficiency of the Config Store and provides maintainability benefits whilst developing the Application software.

Partitioning of Global Data

In ADK, the approach of a single global structure for configuration and run-time state data within the Application has not been retained. Instead, global data is now global within the Application Module, to efficiently utilize the available memory pools.

NOTE This has increased the total number of variables stored in the global data (even though most of these variables only have Application Module visibility). Therefore, the `pmalloc` block required for storing global data has increased in this ADK.

Definition of Configuration Modules

Configuration Modules are defined by the Developer using Configuration Module Definition xml files. Each Application Module that uses configuration data (has a Configuration Module) must have a Configuration Module Definitions xml. During the build process these definitions are used to code generate the C language declarations, definitions and literals for the Configuration Module. This process is repeated for all the Configuration Modules included in the build to generate the Configuration Set for that specific build, please see Modularization in the ADK Application.

NOTE Reference information for the elements and attributes used in the Configuration Modules Definitions xml files is provided in *ADK Applications Configuration XML Definitions Reference Guide*.

Configuration Set default values

The Configuration Module Definitions xml files contain all the data required for the build system to auto-generate a C language data structure specifying the Configuration Set. This data structure is declared as a read-only structure that resides in the Applications const-space. It has a literal initializer that is also constructed from the Configuration Module Definitions specifications at build time, based on the default values and dimensions for all the Configuration Items in the Configuration Set.

At run-time the data structure containing the Configuration Set defaults is passed by the Application initialization code to the Config Store to initialize the device configuration. This is shown by the curved arrow in Figure 2-2.

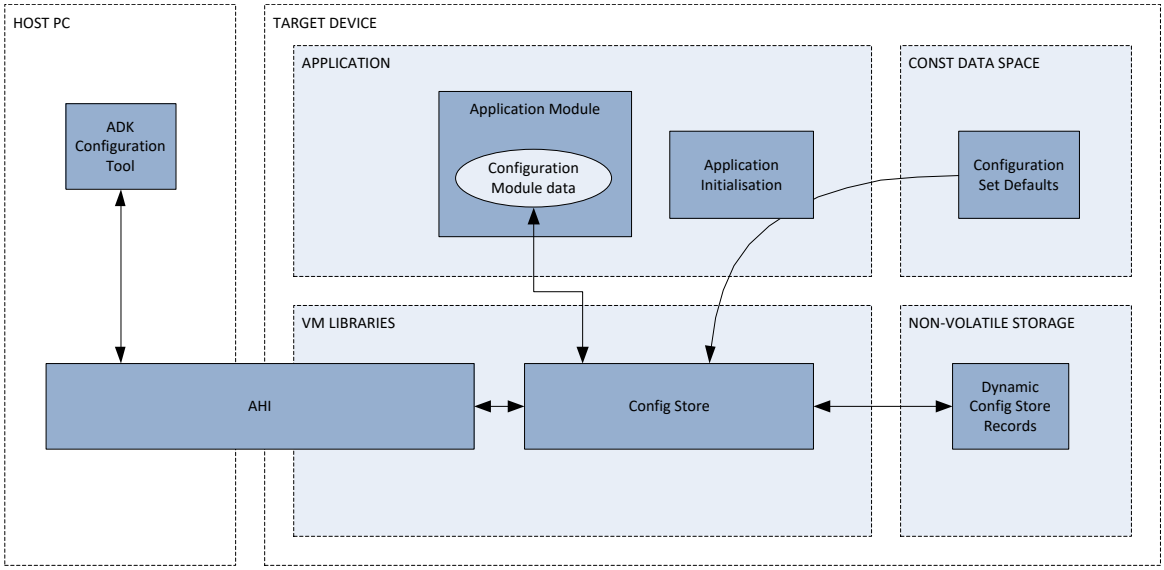


Figure 2-2 Primary elements of the Application Configuration architecture

2.2 ADK libraries

ADK includes two new libraries that are used to support the Application Configuration Architecture: the Config Store and Audio Host Interface (AHI).

2.2.1 ADK Config Store

The Config Store is a new library used to abstract the physical non-volatile storage from the Application Modules, which now uses a standardized API to access Configuration Module data.

The Config Store helps to reduce dynamic memory resources requirements for the Application by allowing read-only Configuration Modules to be accessed through a pointer to const data.

Abstraction of physical non-volatile storage

The Config Store library is structured and designed to allow details of the physical implementation of the non-volatile storage to be contained in a separate library from the higher-level, implementation independent Config Store API

Figure 2-3 illustrates the Config Store architecture:

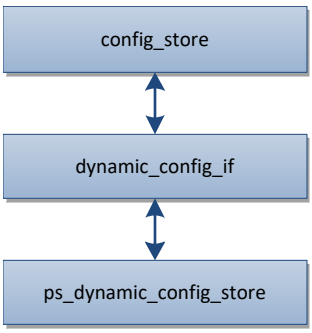


Figure 2-3 Libraries implementing Config Store functionality

Where:

- The high-level API is implemented in the `config_store` library.
- When accessing the functionality implemented by a particular non-volatile storage implementation, the Config Store uses an internal API declared by `dynamic_config_if`.
- `ps_dynamic_config_store` represents an instance of a particular implementation (the Persistent Store of a Qualcomm® device in the example shown in Figure 2-3).

NOTE It is possible to write alternative dynamic Config Store implementations for device or system-specific non-volatile storage mediums.

Initialization

The Config Store library is initialized when the Application itself initializes (See Modularization in the ADK Application).

Reading and writing configuration data

Application Modules can read any Configuration Module data they own from the Config Store. If the Configuration Module contains writeable data, the Application Module can modify this data and write it back to the Config Store.

The Application Module requests read and write accesses to its Configuration Module data by providing the Config Store with its Configuration Block identifier.

Dynamic Config Store records

When a Configuration Module is modified from its const-space default values, either by the Application Module or the ADK Configuration Tool through the AHI, the Config Store creates a dynamic record for the configuration data in non-volatile storage. This dynamic store record will be used to override the const-space default values, whenever it exists.

The Config Store expects the User Persistent Store to be clear of data when an Application is first built and flashed to the device. If this is not the case, stale User Persistent Store data could be misinterpreted by the Config Store as dynamic Config Store records. The Developer should ensure the device User Persistent Store on the device is erased before commencing development.

NOTE Dynamic storage of Configuration Modules is only intended for the Developer's convenience during the development of the Application. When the Application development is completed, and periodically throughout the development process, the Developer should import all the configuration changes that they have made (that is, all those residing in the dynamic Config Store records) back into the Configuration Set default values. This can be accomplished by using the *Export Configuration Set to Configuration Module definition files* feature of the ADK Configuration Tool, which is explained in the *ADK Configuration Tool User Guide*.

2.2.2 ADK Application Host Interface (AHI)

The AHI provides the host PC access to the ADK Application, and indirectly to the Config Store, using either SPI or USB HID physical transports.

NOTE For any particular build of the Application, the AHI transport is fixed as *either* SPI *or* USB HID. To change the AHI physical transport, rebuild the Application.

AHI Application modes

The AHI implements three modes that the Application can operate under; Normal, Configurable and Test modes:

- Normal and Configurable modes are needed to guarantee mutual exclusion between the host PC and the Application Modules when interacting with the Config Store.
- In Test mode, the Application operates in a very similar manner to Normal mode, with the addition of User and System Event injection and sniffing capabilities.

NOTE Application Modules are automatically disconnected from the Config Store whenever the AHI re-starts the Application in Configurable mode. In Normal and Test modes, the AHI is excluded from the Config Store and the Application executes and operates normally.

2.3 ADK Configuration Tool

The ADK Configuration Tool uses the AHI for reading from and writing to the connected target devices Configuration Set, the Tools GUI operation and functionality is consistent with previous ADKs (refer to the *ADK Configuration Tool User Guide*).

The ADK Configuration Tool provides the following features:

- AHI Application Mode selection (Normal / Configurable), in the GUI using a toggle button.
- Read or Write Configuration Set Dump .xml files (Replacing PSR).
- Imports and Exports Configuration Set data directly to the host PC file system (*offline* mode).
- Use the same Tool binary to work with any ACA compliant device, for example:
  - CSRA68100 based devices
  - Qualcomm® BlueCore™ ROM devices (QCC300x)
  - Legacy BlueCore devices (CSR867x)
- The Configuration Set .xml definitions are loaded at run-time from the device.

3 System overview

To understand the ADK Application Configuration Architecture, it is essential to understand the processing performed at build-time and the system behavior at run-time.

Based on the logical processing order of building and running an ADK Application, the following sections first consider the build-time processing to produce the Application Configuration Set. The sections then explain how this Configuration Set is used by the Application Modules and configured by the Developer using the ADK Configuration Tool.

3.1 Build-time

When the ADK application is built, the Configuration Set for that combination of Application Modules and functionality is also specified and generated dynamically. This means that the build-time processing required is more complex than was the case in previous ADKs. However, this process is completely automated within the build system, and requires no intervention from the Developer.

There are two distinct phases to the processing which occurs at build time of the Application

1. Pre-processing and code generation
2. Compilation and linkage

These phases combine and process information about the Application Modules and Configuration Set to produce data structures within the Application binary, which can be accessed through standardized APIs at run-time either by the Application itself (on the device) or the ADK Configuration Tool (running on a host). For information about how data structures are used, see Build-time artifacts.

3.1.1 Build-time artifacts

At the ADK Application build-time, the Configuration Set contains three types of artifacts:

- Source Code
- Intermediate Files
- Build Outputs

Table 3-1 Build-time artifacts

Classification	Description
Source Code	Items normally stored in a Revision Control System and set, modified or altered during the software development process. Source Code forms the raw inputs to the Application Build process and is combined in accordance with the product requirements and targeted hardware. Source Code forms part of the development source codebase.
Intermediate Files	Files generated during the build process that are derivative of Source Code but not stored in Revision Control. Intermediate Files have a life-span based on the validity of their dependencies, in the normal manner of compilation object data.
Build Outputs	Files output from the Build process that are packaged into an Image, to be flashed to the target device.

3.1.2 Build-time process flow

The ADK Application build-time process is the same whether the process is initiated from the IDE or by invoking the makefile directly.

Figure 3-1 illustrates how the input Source Code, Intermediate Files and Build Output files are used, when building an Application image, for Flashing to a device.

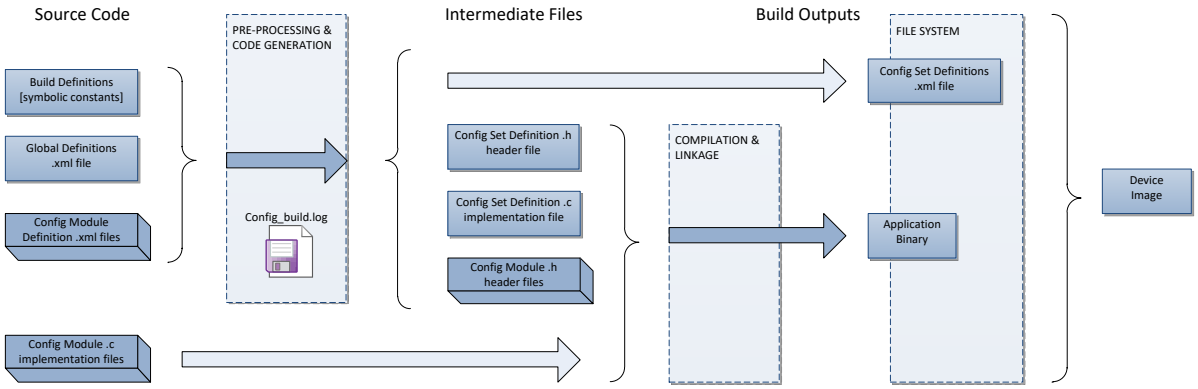


Figure 3-1 Build-time process flow

The left column of blue boxes represents Source Code artifacts that are used primarily by the *Pre-processing and code generation* phase to generate Intermediate Files. These files are then combined with more Source Code, to form inputs to the *Compilation and linkage* phase of the build.

When working with a CSRA68100 device, after the *Compilation and linkage* phase of the build is complete:

- The Application binary is flashed to the Apps Processor (`apps1`).
- The Config Set Definitions .xml file is inserted into the `customer_ro_filesystem`, which is then flashed separately to the device.

NOTE In the CSRA68100 development context, there is no common file system that holds the two Build outputs.

3.1.3 Pre-processing and Code Generation inputs

The following source code items are processed as inputs to the pre-processing and code generation phase of the ADK Application build process.

Build definitions

These are symbolic constants configured by the developer according to the device hardware targeted and the application functionality included. The three build definitions are

```
HW_VARIANT
SW_VARIANT
PLATFORM
```

The build system needs the Software and Hardware variants to be specified to ensure that the correct Configuration Item default values are selected for initializing the Configuration Set data structure.

The platform is set according to the HW variant and determines how `struct` packing should be performed in the Configuration Set data structures.

Configuration Module Definitions .xml files

Every Application Module that uses configuration data must specify the form of this data. This is specified to the build system in the form of .xml definitions on a per module basis. The Module definitions describe the name of the Configuration Item as referenced from the Application Module source code, as well as the type and default value for each Configuration Item.

The module definitions .xml file also contains a long-form Configuration Item name and some ‘help text’ description, both of which are displayed to the Developer by the ADK Configuration Tool.

NOTE If an Application Module is not included in the build, then its Configuration Module will not be included in the Configuration Set created for the build. This means that none of the Configuration Items present in that Configuration Module will be present in the type definition that represents the Configuration Set, furthermore they will not be visible in the ADK Configuration Tool when it is used to view or change the Configuration Set. They are completely excluded from the binary created by the build and will therefore not use any const or Persistent Store resources.

The filename convention of the Configuration Module definition files is `*_config_def.xml`, where `*` denotes the name of the Application Module.

Global definitions .xml file

This file defines definition data which is reused across multiple modules. It also defines the hierarchical structure determining how the Configuration Set is displayed to the Developer in the ADK Configuration Tool.

The filename is `global_config.xml`.

3.1.4 Pre-processing and Code Generation phase outputs

The Pre-processing and Code Generation Build phase produces:

- Configuration Set definition implementation and header files
- Application Module header files
- Configuration build log file
- All required Configuration Set definitions .xml files

Configuration Set Definition implementation and header files

These are intermediate files which define the type for the data structure representing the Configuration Set and its associated metadata. This data structure dictates the default Configuration Item values, the mapping between each configuration block and the user PS key it is assigned to and also some metadata about the Configuration Set (for example md5 hash signature).

NOTE The filenames are `config_definition.c`, and `config_definition.h` for the implementation and header files.

Application Module header files

The Module headers are code-generated intermediate files that define the types used by the Application Modules to read the binary configuration data into the application.

NOTE The filename convention of the Module header files is `*_config_def.h`, where `*` denotes the name of the Application Module and each Module Definition input file generates a single `*_config_def.h`.

Configuration Build Log file

The Configuration Build Log file (`config_build.log`), is used to record all Pre-processing and Code Generation phase processing data, generated during the build.

Configuration Set Definitions .xml file

This file is a build output. It is derived from the specified Module .xml file, Global .xml file and build definitions and defines the specific Configuration Set generated by the build system.

The Config Set Definitions .xml file is compressed using gzip and placed in the read-only file system that will be created on the target device. At run-time, this gzip archive is accessed and retrieved by the ADK Configuration Tool to allow the Developer to browse and modify the Configuration Set.

NOTE The filenames are `config_definition.xml` for the definitions file, and `config_definition.gz` for the compressed archive.

3.1.5 Compilation and linkage phase

The following intermediate files from pre-processing and code generation are used as inputs to the compilation and linkage stage of the build:

- Configuration Set Definition implementation and header files
- Application Module header files

These are compiled with the final part of source code used in the Configuration Set build system

Application Module .c implementation files

The Application Module implementation files use the configuration data owned by the Module. They use the Config Store API to access this data, and expose it to other Modules on the system as required using their own module level APIs.

The build output for the compilation and linkage phase is an Application binary.

3.2 Run-time

After the ADK Application and its Configuration Set have been built and flashed to the device, the run-time behavior of the system can be considered.

At any point during run-time, the Application operates in one of the following AHI modes:

1. Normal
2. Configurable
3. Test

To following sections explain the differences between Normal/Test and Configurable modes:

- Normal mode
- Configurable mode

3.2.1 Normal mode

In normal mode the ADK Application runs as normal. Application Modules execute, respond to events and access/process data according to their design. Any Application Modules that utilize Configuration Items are free to read or write their configuration data to the Config Store at any time.

The AHI operation is restricted to establishing connections with devices and controlling AHI mode transitions. The AHI cannot access the Configuration Set values in this mode of operation.

NOTE AHI mode transitions are supervised by the Application. The AHI library implements the mechanism to control restarting the Application in the requested mode, but decision making about when device resets are required are deferred to the Application.

Figure 3-2 illustrates run-time operation of the system in Normal AHI mode:

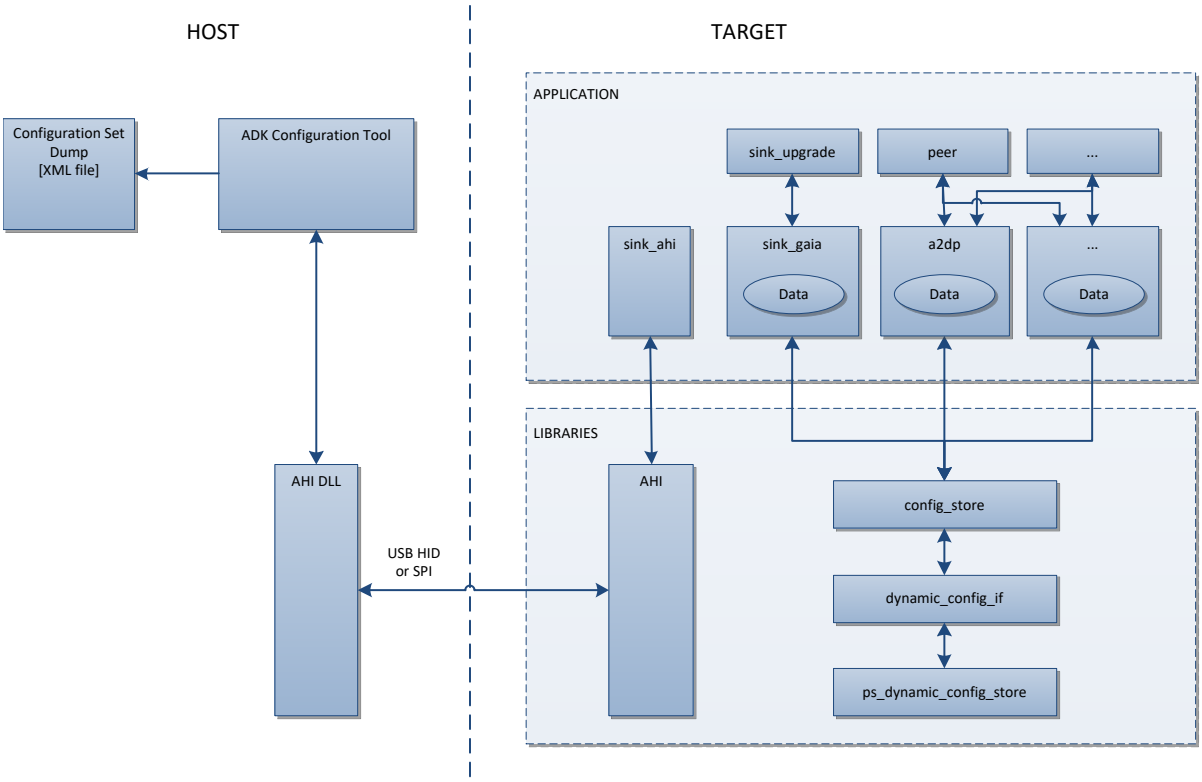


Figure 3-2 Run-time Normal mode operation

NOTE In Test mode, the Application operates as in Normal mode, with the only addition being that User and System Events can be injected and sniffed from the host PC, using the AHI system.

3.2.2 Configurable mode

In contrast, in configurable mode the ADK Application runs in a restricted mode so that no Application Modules can access the Config Store. This guarantees exclusion to enable the AHI to read and write the Configuration Set. This is shown in Figure 3-3, which illustrates run-time operation in Configurable AHI Mode:

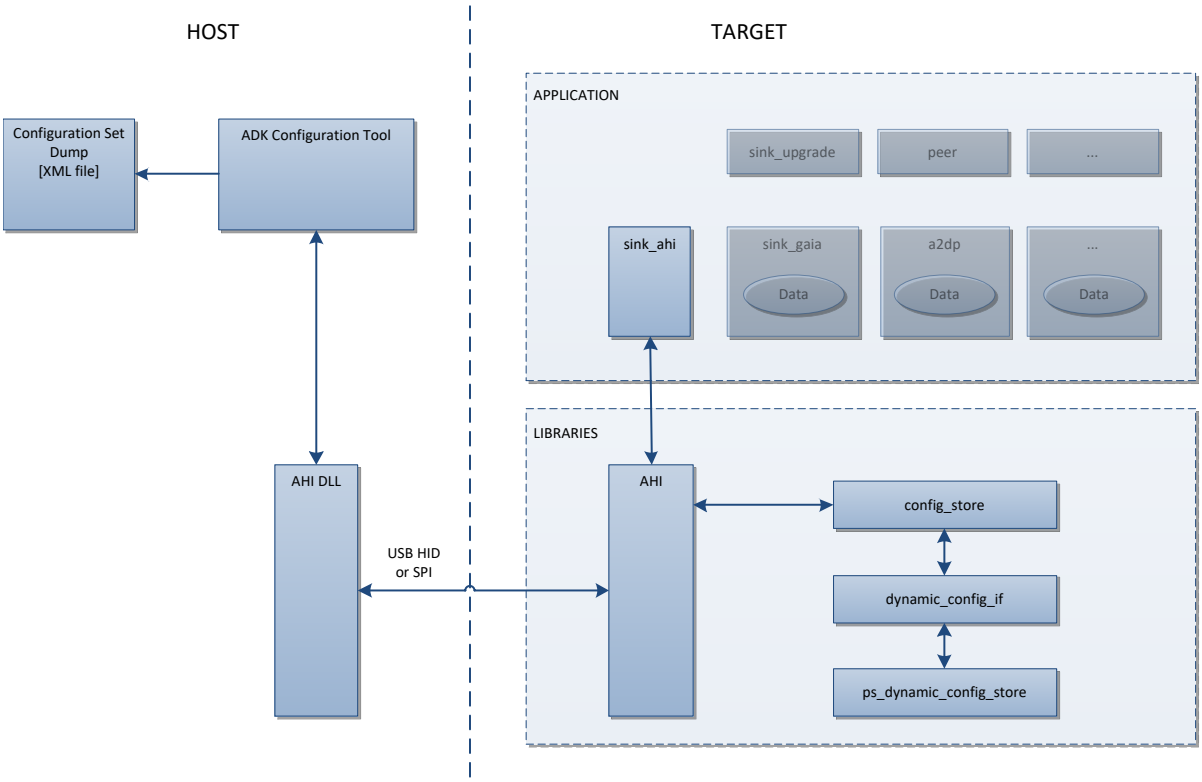


Figure 3-3 Run-time Configurable AHI mode

NOTE The Application Modules are shown grayed out to indicate they are not running. In Configurable mode the message scheduler only operates in a restricted capacity and module execution is postponed until the AHI is returned to normal mode.

Configurable mode allows the ADK Configuration Tool to modify the Configuration Set of the Application. Any changes made by the Developer using the Configuration Tool will be applied to the Config Store, but will not be used by the Application until the AHI is returned to Normal mode. At this point the device will restart and the Application modules will initialize themselves from the Config Store using the modified Configuration Item values.

Terms and definitions

Term	Definition
ACA	Application Configuration Architecture
AHI	Audio Host Interface
API	Application Programming Interface

Document references

ADK Application Configuration System User Guide	80-CT548-1/CS-00400610-UG
ADK Configuration Tool User Guide	80-CF419-1/CS-00406798-UG
ADK Applications Configuration XML Definitions Reference Guide	80-CT541-1/CS-00346862-UG