



Training: Hardware Abstraction Layer (HAL)

Michael Schmidt

Technical Leader, Embedded Software

AGCO Advanced Technology Solutions

Objectives

- Explain the purpose of Hardware Abstraction Layers
- Provide helpful information for developing a HAL for an ISOBUS implement
- Show examples of HALs

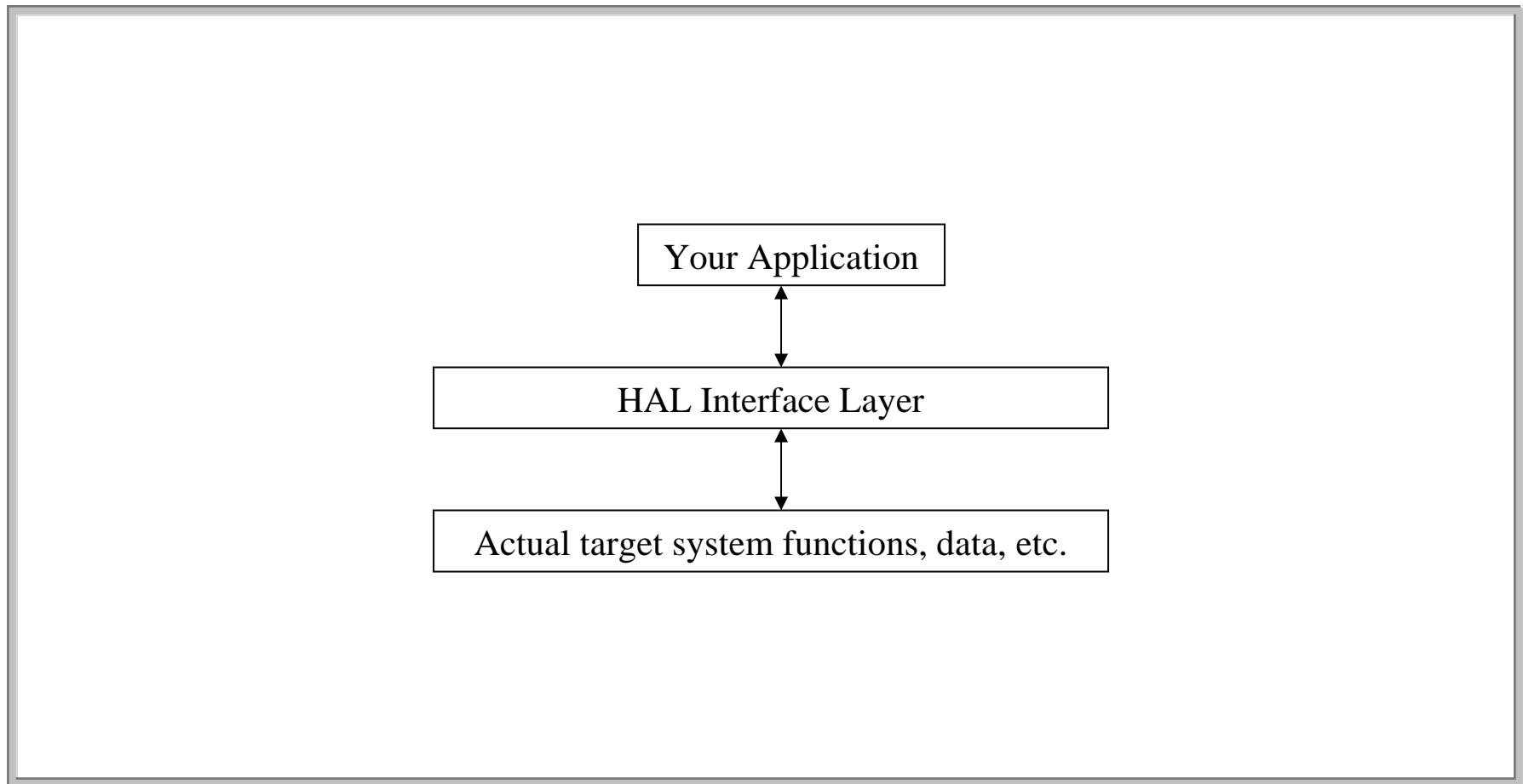
Topics

- What is a Hardware Abstraction Layer (HAL)?
- Software Layer Diagram
- What Pieces of the “Target System” can be Abstracted?
- Advantages of Using HALs
- Open Source HALs
- Minimum HAL Required to Develop an ISOBUS Implement
- Other HALs Available for Implements
- IsoAgLib HAL Structure and C++ namespaces
- IsoAgLib Application Programming Interface (API)
- IsoAgLib Software Layer Diagram
- Things to be careful of

What is a Hardware Abstraction Layer (HAL)?

- Generic Application Programming Interface (API) used to control a specific piece of the target system
- May include functions, variables, structures, classes, etc.

Software Layer Diagram



What Pieces of the “Target System” can be Abstracted?

- Physical hardware
 - CPU
 - Video card
 - I/O board
 - CAN Bus Card
 - Disk I/O
 - Communication
- ROM
- BIOS
- Operating System (O/S)
- etc.

Advantages of Using HALs

- Application source code can be developed independently of the actual target system that it will be run on
- Greatly speeds up development time
- Development and high-level logic testing can be done on a high-power PC with a multitude of powerful development, debugging, profiling, and testing tools
- Target system can be changed with either no impact, or only a minimal impact on the application software
- Centralize all details of how to talk to the target system into one small area of code
- Using inline functions and other compiler optimizations, the HAL can often be written in a way that it is just as efficient as talking directly to the target system

Open Source HALs

- ISOBUS Specific
 - IsoAgLib
- GUI
 - wxWindows
 - Qt
- Many others

Minimum HAL Required to Develop an ISOBUS Implement

- System functions
 - System Initialization / Shutdown
 - Watchdog
 - Time
- CAN Bus functions
 - CAN Bus Initialization / Shutdown
 - Hardware / Software filters
 - Read / Write

Other HALs Available for Implements

- EEPROM
- RS-232 Communication
- Data Streams (i.e. File I/O)
- Sensor Inputs
- Digital / Analog Outputs

IsoAgLib HAL Structure and C++ namespaces

■ “HAL” namespace

- Functions, variables, structures, etc. visible to applications that wish to safely use the HAL generically
- Direct function calls into similar “_ _HAL” functions with parameter swapping
- “Target Extension” functions that you create to handle the functions not provided by the target system

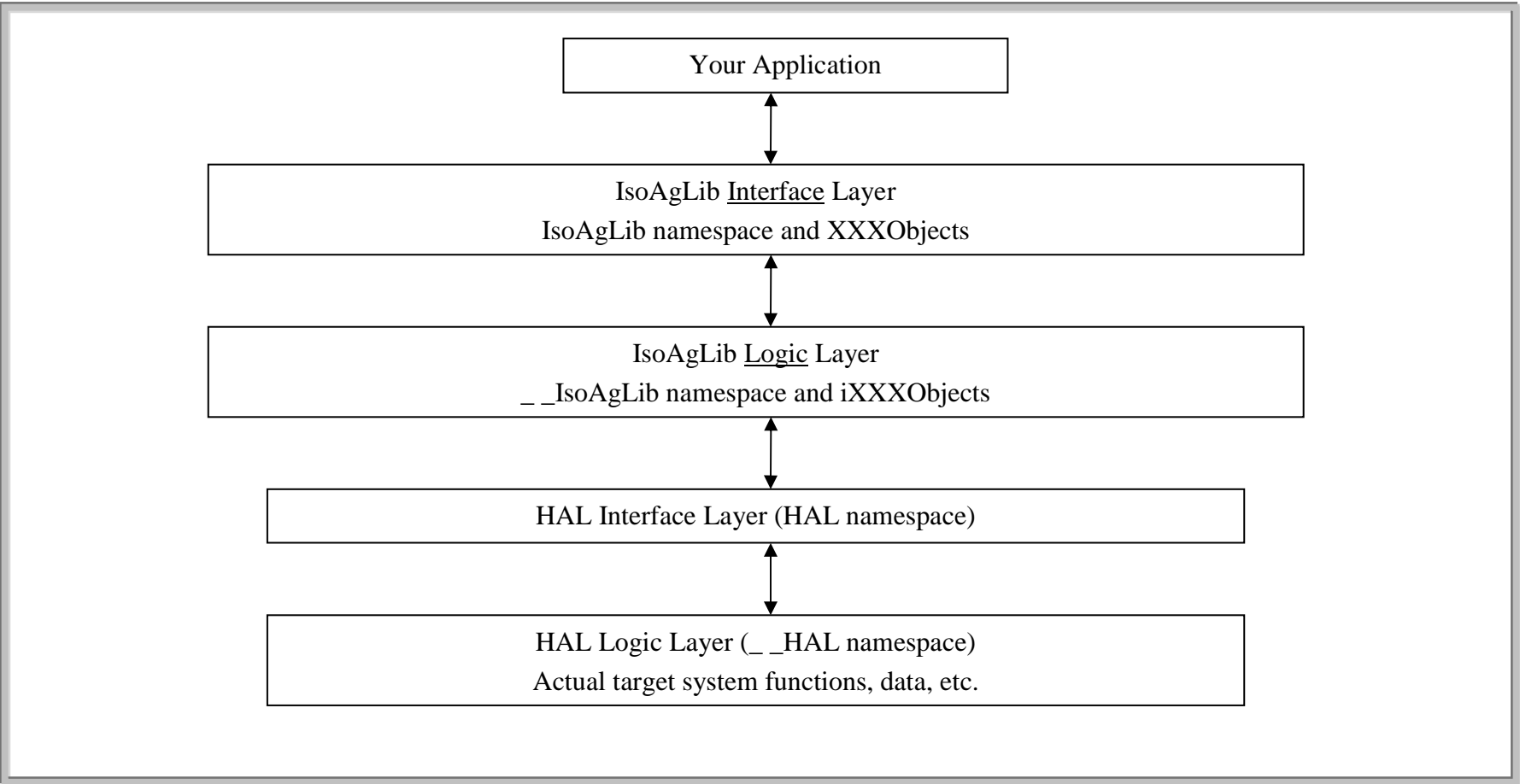
■ “_ _HAL” namespace

- Functions, variables, structures, etc. provided by the manufacturer of the target system
- Should not be used outside of the _ _HAL or HAL namespaces to guarantee compatibility.

IsoAgLib Application Programming Interface (API)

- IsoAgLib namespace - “iXxxObject” objects (the “i” means “interface”)
 - Functions, variables, structures, etc. visible to applications that wish to use IsoAgLib
- __ IsoAgLib namespace - “XxxObject” objects
 - Functions, variables, structures, etc. used internally within IsoAgLib
 - Should not be used outside of IsoAgLib

IsoAgLib Software Layer Diagram



Things to be careful of

- Always use target-independent variables at the application level
 - For example, define and use "uint32_t" instead of "unsigned long int"
 - Compiler Target and the number of bits of "int" Types

Compiler Bits	8	16	32	64
short int	N/A	8	16	32
int	8	16	32	64
long int	16	32	64	128

Things to be careful of

■ Big-Endian vs. Little-Endian

- Use #define's at the application level when byte ordering could be a problem
- IsoAgLib uses:
 - OPTIMIZE_NUMBER_CONVERSIONS_FOR_LITTLE_ENDIAN
 - OPTIMIZE_NUMBER_CONVERSIONS_FOR_BIG_ENDIAN

■ Compiler Adaptions

- Length shortening of class names
- Classes provided / not provided by compiler
- C namespace adaption