ALLSEEN ALLIANCE

AllJoyn Coding Standards Guidelines

March 18, 2014

Styling

1. **Spacing (Tabs vs Space)**
    a. Do not use tab characters for indentation. Use only spaces for indents

       Rationale: Different editors use different tab width space and this could make the code unreadable.
    b. Indent levels should be 4 spaces
    c. Don't aggressively split up statements to stay under 80 characters per line
    d. Avoid lines over 120 characters
    e. Long list of function parameters (above 120 columns) should be vertical and indented to line up with the first parameter
    f. Class public, protected and private access labels are indented by 2 spaces (C++ only)
2. **Whitespace**
    a. Make sure there are no trailing whitespaces on any line. Doxygen tool used in AllJoyn is supposed to take care of this. Please make sure you run the tool with WS=fix before submitting the code.
3. **Braces**
    a. The AllJoyn core code follows K&R style
    b. Opening and closing brace for functions should be on its own separate line
    c. All if..else statements must have braces even if it has just one statement
    d. Include braces to create a code block for a case cause only when needed to manage the scope of local variables within the clause

```
void my_function(void)
{ /* Opening function brace is on its own line */
    switch (var) {
    case C1:
        /* Case clauses normally do not have braces surrounding the entire clause */
        if (cond1) {
            /* Note that blocks with one line still have braces. */
            DoSomething();
        } else {
            DoSomethingElse();
        }
        break;

    case C2: {
        /*
         * Include braces to create a code block for a case clause only
         * when needed to manage the scope of local variables within
         * the clause.
         */
        bool cond2;
        do {
            cond2 = DoSomethingRepetitively();
        } while (cond2);
        break;
    }
    default:
        DoNothing();
    }
}
```

Naming

4. **Function naming convention**
   a. Function name should use camel case with the first word Upper case
   b. Avoid long function names
5. **Variable naming convention**
   a. Variable name should use came case with the first word lowercase
6. **Use of company specific names in Allseen code**
   a. Do not use your company name or any abbreviation in the code contributed to the Allseen repository
7. **Other naming convention**
   a. Try to have short unique prefixes for related code
      For e.g. All the code related to IP Name Service has 'IPName' as the prefix
   b. File names should have convey the class or object it implements
8. **Constants**
   a. All constants should be uppercase
   b. Give meaningful names to constants rather than using literals

```
/* Good example */
const int MAX_PACKET_LEN = 0xFFFF;
const int ER_BUFFER_OVERFLOW = 234;
if (len > MAX_PACKET_LEN) {
    return ER_OVERRUN;
}

/* Bad example */
if (len > 0xFFFF) {
    return -1;
}
```

   c. const should be used on all pointers (or C++ reference) parameters which are not modified by the function or any function called by the function

```
#define PREPROCESSOR_CONSTANT 0x1234
enum {
    ENUM_CONSTANT,
 …
};
const int CONSTANT=0x1234;
```

9. **Comments**
   a. Use Doxygen style markup for public API documentation
   b. Block comments that are for public APIs in the header files should start with '/**' followed with '*' for every new line with the comment and end with '*/' on a new line
   c. Single line comments should start with a '/*' and end with a '*/' on the same line

```
/*
 * Multi-line
 * Comment
 */
```

Code structure and internal

10. Header files

a. All header files must include incorporate multiple inclusion protection

b. #includes double quotes should be limited to header files in the same directory as the file with the #include directive

c. Any include file that is in either standard C include paths or in a project include directory (passed with a “-I”) should use angle brackets

d. Header files for public APIs go under inc/<name of file>

e. Header files for internal use go under in the same directory as the C/C++ file that implements the defined functionality

f. All header files that use types defined in other header files must include those header files

```
#include "same_directory.h" /* Header file is in the same directory as this one */
#include <header.h>         /* For system headers or headers found using the include
path */
```

11. Standard functions to avoid

a. Do not use standard functions that can easily lead to buffer overruns and cause security problems. A known list of functions include

Functions to avoid	Recommended alternative
strcpy()	strlcpy()
strncpy()	strlcpy()
sprintf()	snprintf()
memcpy()	memscpy()
strcat()	strlcat()

12. Status checking

a. Explicitly check status codes rather than using macros

13. Pragma

a. Avoid use of pragma unless it is completely unavoidable for platform specific code

b. If at all it is required then it must be conditionally compiled in only for the compiler that it applies to

14. Data scope and access

a. Declare private data and functions as private or static

b. Avoid global variables

c. Use accessor functions for read-only values and if possible return a const pointer for the data they are accessing
d. Modules requiring multiple global values should group them in a class or data structure

15. Public API

a. If they are not thread safe they should clearly mention that in the header file comments
b. No STL apart from the ones that are a part of the core cod should be used in public APIs

16. Locking

a. Use fine-grained locking rather than global locks

17. Location of code

a. AllJoyn core code related to router should go under alljoyn_core/router
b. Code which is supposed to have utility purpose and possibly have platform dependent implementation should go under common. If you have a confusion ask yourself "Can this code be used outside of AllJoyn independently?" If the answer is yes, it is very likely a candidate for common
c. Platform specific build rules should go under build_core/conf/<platform name>
d. Unit tests for the core code goes under alljoyn_core/unit_tests

18. Error reporting/Exceptions

a. Error reporting should be done using QStatus
b. Do not use exceptions

Copyright and license

19. Copyright notice

a. All source and header files should have a copyright license header mentioning ISC open source license for Allseen Alliance at the top

Bindings

20. Java bindings

a. We follow the Android and Sun Java coding conventions found here: http://source.android.com/source/code-style.html
b. All bindings code for Java goes under alljoyn_java/src
c. All native JNI code should be placed under alljoyn_java/jni
d. Comments should follow the Javadocs convention

21. Objectivc-C bindings

a. All bindings code and files have a prefix "AJN" followed by the name of the file that you have in your native code
b. We follow the coding conventions mentioned by Apple https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CodingGuidelines/CodingGuidelines.html
c. Comments should follow the appledoc convention

22. **Unity**
    a. Unity bindings uses tab spacing
    b. Must use IODisposable pattern to clean up native resources
    c. All public headers should have standard C# data types
    d. For windows, all C# public APIs should use the macro AJ_API when exposing the header files

23. **Javascript**
    a. Javascript bindings uses widlproc for documentation

24. **C bindings (for Thin library)**
    a. Name of source files should start with a prefix “aj_” followed by the actual name which is lower case
    b. Name of functions should start with a prefix “AJ_” followed by camel case function name with the first alphabet in uppercase
    c. Target specific code for thin library should go under root/target folder
    d. All pointers must be initialized to NULL and not 0