The Art of Design and Development

Interfaces

What do they do?

- An interface between two entities
  - Generally defines a set of (optional) inputs
  - Potentially provides specified/required outputs
- Provide a functionality (~API’s)
  - Often hiding implementation details from the caller
  - May be by another company/team within the same company/BU
  - Multiple ways
    - Functions/
    - Command line
    - REST API’s
    - SNMP
    - ...

API’s or Interfaces

- What they should avoid
  - Play with global variables
    - Exception for stuff like counters (with mutex if reqd), but other than that can be risky
    - Proven failures in esp. concurrent environment
  - Modify caller’s data
    - unless it’s by design
  - Create a long chain of dependencies
    - Explicitly decide
  - Create a large number of unnecessary function calls as API’s
    - Esp. some of them providing multiple ways of doing the same thing
    - Often done for convenience, but best to avoid
  - Write to storage / transact on network (unknown to user -- needs documentation)
    - create temp/secret files OR write to database
    - transacting data on local network/internet
  - Hold up allocated memory

API’s or Interfaces

- What will help
  - Avoid large number of arguments
  - Doing the same thing the same way everywhere, avoid unpleasant surprises to the user
    - E.g I/O library order
    - Keep function signatures and return values intuitive
    - Borrow styles from standard libraries
  - Proper resource allocation and deallocation
    - Memory and storage
    - When using buffers see if a pool can be used/reused

Designing widely used API’s

- Keep it simple and easy to understand
- Give example code
  - All use cases with defensive error checking demonstrated
  - Highlight calls making (memory/storage) allocation and complementary deallocation calls to avoid holding up system resources
- Pass up/dump errors/conditions passed by third-party libraries “as is”
- Return values need to reflect more than just the result
- If your API’s are getting too complex, then it may be a code smell..
- Develop all test cases along with the API, and publish them
  - Stakeholders can review
- Multi-threading support
- Idempotent

Designing widely used API’s

- Pair functions, naming and documentation, usage
  - Get, set, Encode, decode, alloca, dealloc..
    - Testing
- Indicate deprecation?
- Calling another API required for more information
- Keep it simple and easy to understand
- Error handling
  - Quitting may not be the right solution, program may need to carry on
  - Allow higher level code to handle errors
  - Cautious on use of exceptions, use them only for unexpected events
  - Publish possible error codes, so that caller is not surprised
  - Errors by any other libraries called need to be considered - translate or pass ‘as is’

Designing widely used API’s

- Design program such that bad input is easy to detect and flag
  - Makes life easy from both usage and a security standpoint
- Clarity in error messages
  - E.g. : Error *‘Error:Parameter xlimit too small’* versus *‘Error:rxLib:setLimit, Parameter xlimit 15, valid range 25-600,’*
- Follow uniform convention for documentation, usage, arguments..
  - Usage of same terminology/words, sequence of arguments, units, formats, textual formatting as applicable
- Understand all available options
  - Big impact on overall design
  - Proper selection of language/tools and even OS
  - Certain constructs available may make the job much easier and faster
- Interfaces : GUI design, a course on its own!