C++ std::ios flags, Intro to Yocto

ESE2025

std::ios

- std::ios_base and its derived class, std::ios, are the base classes of I/O stream classes, providing features common to both input and output streams
- chief among the features are the ios state flags, which provide critical information regarding the condition of a stream
- for example: if a file cannot be opened, the state of the associated file stream will not register as “good” (see next slide)

std::ios (cont'd)

State flag functions:

good	Check whether state of stream is good (public member function)
eof	Check whether eofbit is set (public member function)
fail	Check whether either failbit or badbit is set (public member function)
bad	Check whether badbit is set (public member function)
operator!	Evaluate stream (not) (public member function)
operator bool	Evaluate stream (public member function)
rdstate	Get error state flags (public member function)
setstate	Set error state flag (public member function)
clear	Set error state flags (public member function)

source: http://www.cplusplus.com/reference/ios/ios/

std::ios example

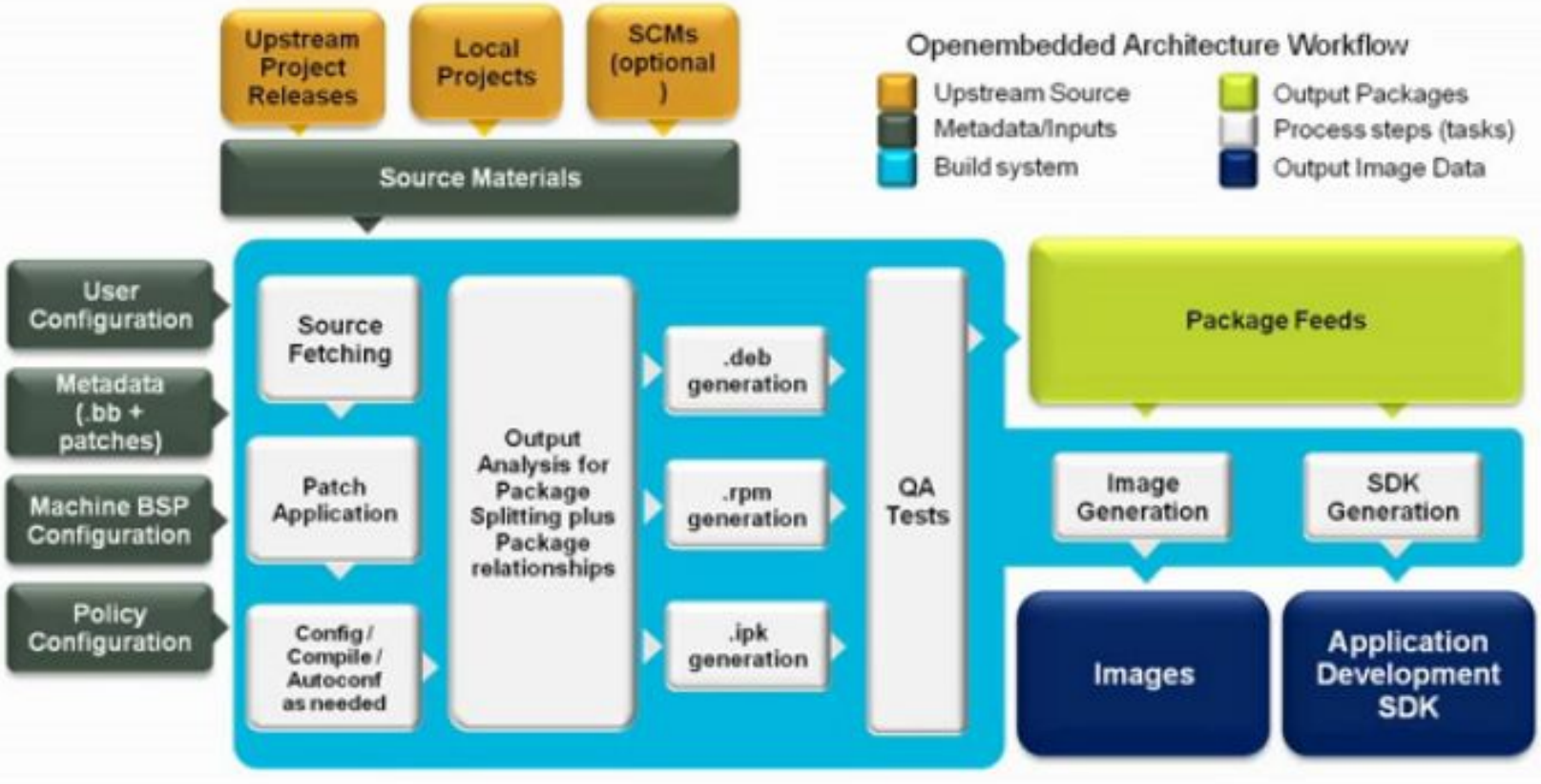
```
// error state flags
#include <iostream>     // std::cout, std::ios
#include <sstream>      // std::stringstream

void print_state (const std::ios& stream) {
  std::cout << " good()=" << stream.good();
  std::cout << " eof()=" << stream.eof();
  std::cout << " fail()=" << stream.fail();
  std::cout << " bad()=" << stream.bad();
}

int main () {
  std::stringstream stream;

  stream.clear (stream.goodbit);
  std::cout << "goodbit:"; print_state(stream); std::cout << '\n';

  stream.clear (stream.eofbit);
  std::cout << " eofbit:"; print_state(stream); std::cout << '\n';

  stream.clear (stream.failbit);
  std::cout << "failbit:"; print_state(stream); std::cout << '\n';

  stream.clear (stream.badbit);
  std::cout << " badbit:"; print_state(stream); std::cout << '\n';

  return 0;
}
```

Yocto Project

- backed by a number of large tech corporations and individuals, in order to expedite getting embedded Linux systems working!
- hardware companies like IBM and TI have a vested interest in getting IoT hardware systems working (so they can sell their chips!)
- getting a Linux OS working for a newly designed embedded system is not a simple matter (there is no off-the-shelf distribution like Beagleboard Debian or Raspbian!)
- although there are several embedded OS “generic” operating systems available, notably: Android, Angstrom, OpenWrt;

Yocto Project Cont’d

- Yocto is not the only embedded Linux Build System:
  - Baserock
  - Buildroot
  - OpenEmbedded
- What is hard about a custom Linux distribution?
  - Bootloader
  - Kernel
  - Device Drivers
  - Life Cycle Management
  - Application Software Management


Openembedded Architecture Workflow
Upstream Source
Metadata/Inputs
Build system
Output Packages
Process steps (tasks)
Output Image Data
Upstream Project Releases
Local Projects
SCMs (optional)
Source Materials
User Configuration
Metadata (.bb + patches)
Machine BSP Configuration
Policy Configuration
Source Fetching
Patch Application
Config / Compile / Autoconf as needed
Output Analysis for Package Splitting plus Package relationships
.deb generation
.rpm generation
.ipk generation
QA Tests
Package Feeds
Image Generation
SDK Generation
Images
Application Development SDK

Yocto: fetch, extract, patch, configure, build...

- essentially, you use various configuration files and special script files called “recipes” to direct the build system in how to construct your Linux image
- the image is then “flashed” on your embedded system
  - in this unit, we’ll be emulating the embedded system, so you won’t have to worry about flashing a physical device!
- the Yocto’s BitBake program then runs, based on the configuration files and recipes goes through a repetitive process involving the same steps:
    - fetch (from many different sources on the Internet)
    - extract (un-zipping, un-archiving)
    - patch (updating the source code)
    - configure (configuring the build)
    - build (compiling, linking)
    - install (placing the program in the right directories, etc.)
    - package (creating the archive or image)