
# StatusCode Extension

Frank Winklmeier
University of Oregon

*Gaudi Workshop 2017*
*26 Sept 2017*


# StatusCode: Status Quo

## StatusCode can store an arbitrary number as its code

```
enum { FAILURE=0, SUCCESS=1, RECOVERABLE=2 };
StatusCode(unsigned long code);
```

- But any code >2 results in `isFailure()==True`
- Hence at the moment one cannot return additional information in case of SUCCESS

## Use cases

- Flag “small problems” during event processing that should not result in an abort
  - Event reconstruction was aborted prematurely due to timeout
  - Event has some missing data fragments
- In the ATLAS HLT this is currently done via a dedicated HLT::ErrorCode and custom base classes
  - But we want to get rid of all this special code in athenaMT

## Advantage of storing this in StatusCode

- Algorithm StatusCode is already stored in AlgExecStateSvc
  - Can be queried at the end of processing and appropriate actions taken
- If not stored in StatusCode would have to re-invent a very similar service

# Prototype I

## Split `code` into framework and user-specific bits

- see MR Gaudi!380
- isSuccess(), isFailure(), etc. only act on the framework-specific bits

## Advantage

- Minimal invasive change
- Does not increase sizeof(StatusCode)
- Same performance as today

## Problem

- Could break existing code
  - e.g. StatusCode(513) currently is considered FAILURE
  - By only considering e.g. the lowest 8 bits [513 = 0b1000000001] this becomes SUCCESS
- Would be better if user could decide on the mapping of SUCCESS/FAILURE for individual codes

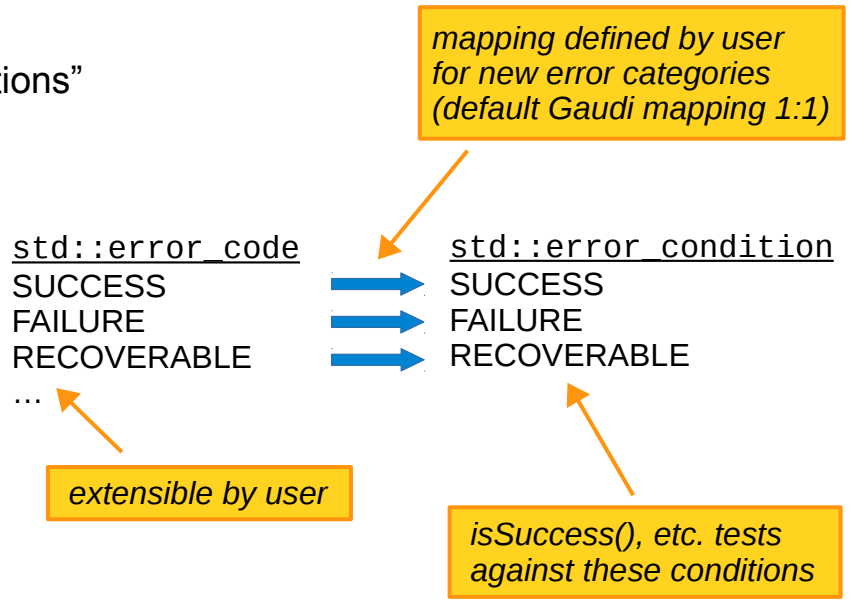
## Fix for above problems

- Could add a completely separate “user-code” that is never considered in isXYZ() methods

# Prototype II

## Leverage std::error_code / std::error_condition

- Allows type-safe custom mapping of “error codes” to “error conditions”
  - Thanks to Gerhard for the pointer
  - https://akrzemi1.wordpress.com/2017/08/12/your-own-error-condition/
- Each error_code belongs to a “category” (domain)
  - Mapping of codes to conditions is done for each category
  - New error codes and categories can be added by clients




## Implementation

- https://gitlab.cern.ch/fwinkl/sandbox/tree/master/StatusCode
- change unsigned long code → std::error_code in StatusCode
- Mostly backwards-compatible with existing code
  - Mapping of integer values has to be changed to follow std convention: 0 = SUCCESS
  - But removed operator long() → will break code that relies on this implicit cast → should be trivial to fix

## Cost

- sizeof(StatusCode) from 32 to 40 bytes (on x86_64)
- For most cases, one extra function call for each StatusCode comparison

# Example

- Relevant code from https://gitlab.cern.ch/fwinkl/sandbox/tree/master/StatusCode

```
enum class MyErr {
  SUCCESS = 0,
  FAILURE = 1,
  RECOVERABLE = 2,
  NO_LVL1_ITEMS = 10,
  MISSING_FEATURE = 11
};

struct MyErrCategory : std::error_category
{
  virtual bool equivalent(int code, const std::error_condition& cond) const {
    // here define mapping of error_code to error_condition
  }
};

// […] some template magic to register above enum with category (see MyErr.h)

{  // no change of current behavior
   StatusCode sc(StatusCode::SUCCESS);
   assert(sc.isSuccess())
}
{  // custom error codes in default Gaudi category behave as today
   StatusCode sc(42);
   assert(sc.isFailure());
}
{  // custom error codes in user category behave as defined by user
   StatusCode sc(MyErr::MISSING_FEATURE);
   assert(sc.isSuccess());
}
```