### Charm++ - Feature #1444

**General Implementation for Serializing Enums**

02/22/2017 03:02 PM - Nils Deppe

<table>
<thead>
<tr>
<th>Status:</th>
<th>New</th>
<th>Start date:</th>
<th>02/22/2017</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
<td>Due date:</td>
<td></td>
</tr>
<tr>
<td>Assignee:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Category:</td>
<td></td>
<td>% Done:</td>
<td>0%</td>
</tr>
<tr>
<td>Target version:</td>
<td>6.9.0</td>
<td>Estimated time:</td>
<td>0.00 hour</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Spent time:</td>
<td>0.00 hour</td>
</tr>
</tbody>
</table>

**Description**

Currently it's not possible to serialize enums in general. However, by using C++11's `std::is_enum` this can be done by as follows:

```cpp
template <typename T,
          typename std::enable_if<std::is_enum<T>::value>::type* = nullptr>
inline void operator|(PUP::er& p, T& s) {
  pup_bytes(&p, static_cast<void*>(s), sizeof(T));
}
```

This also requires modifying `pup.h` to

```cpp
#if __cplusplus >= 201103L
namespace charm_pup_details {

  template <typename... Ts>
  struct make_void {
    typedef void type;
  };

  template <typename... Ts>
  using void_t = typename make_void<Ts...>::type;

  template <typename T,
            typename U = void>
  struct is_pupable : std::false_type{};

  template <typename T>
  struct is_pupable<T, void_t<decltype(std::declval<T>().pup(std::declval<PUP::er &>()))>>
    : std::true_type {};

} // namespace charm_pup_details
```

```cpp
#else
#endif
```

```cpp
inline void operator|(PUP::er &p, T &t) {
  p.syncComment(PUP::sync_begin_object);
  t.pup(p);
  p.syncComment(PUP::sync_end_object);
}
```

**History**

**#1 - 02/22/2017 05:11 PM - Sam White**

Thanks for pointing this out. After the release of Charm++ 6.8.0 (in the coming weeks), we will begin requiring full C++11 support. Before then, I don't think we want to add this because then depending on the compiler used to build Charm++ the application would need to specify their own PUP routine or not.
Yes, that makes sense. I'm really glad to hear full C++11 support will be required! We are currently using C++11 and some C++14 and have had to modify a couple other things in Charm++ to get it to work for us. I'm going to clean those changes up and submit feature requests with complete code there as well. Thanks for the quick response!

Thanks for your patch, Nils. If you'd like, you can directly submit it for code review and merge at https://charm.cs.illinois.edu/gerrit/. As Sam said, it won't be included in the 6.8.0 release, but it's always good to get an early start on reviewing and testing patches.

If you have any questions about the patch submission process, feel free to contact me at rabuch2@illinois.edu.