Jem/Jive 3.0
Release Notes

Dynaflow Research Group

20-12-2019

1 Overview

Version 3.0 of Jem/Jive is a major release with some new features, bug fixes, lots of improvements, and, last but not least, an update to the C++11 standard. This version should be largely source compatible with version 2.2, although you might need to make some small changes to your Jem/Jive programs.

The following list provides an overview of the new features and improvements. More detailed information can be found in the following sections.

- Extension of the `NonlinModule` with support for solving non-linear minimisation problems with box constraints. These kind of problems occur, for instance, when using a phase field to describe the propagation of a crack in a solid material.
- Extension of the format of properties files; you can now use the += operator to extend a previously defined property.
- Improvement of the `Ref` class so that it behaves more like a regular pointer. You no longer need to use the `NIL` constant, but can use `nullptr` instead.
- Support for move semantics where that makes sense.
- Support for initializer lists for arrays and other sequence types.
- Support for the GLFW OpenGL toolkit in addition to the GLUT toolkit.
- Complete overhaul of the implementation of the `gl` package. It can now make use of modern shaders to achieve better and faster graphics.
- Improved parallel performance of the linear solvers and preconditioners, and that of the coarse grid preconditioner in particular. Improved handling of simulations in which degrees of freedom are added/removed dynamically only in parts of a model.
- New class `jem::Flags` for handling bitwise flags in a better way.
- Change of the `jem::idx_t` type from a typedef into a class type. This should be mostly transparent to Jem/Jive applications if you used the `idx_t` type properly. You may have to change a few `int` variable declarations to `idx_t` declarations.
- New header files with forward declarations. All Jem/Jive packages now provide a header file `<forward.h>` that contains forward declarations for the classes provided by that package.
- New Jem package `crypto` for performing encryption/decryption operations. This requires the OpenSSL library.
- New Jem package `net` for performing network operations, such as sending and receiving data through sockets.

- New Jem package `ssl` for performing network operations over secure sockets. This requires the OpenSSL library.

2 Update to the C++11 standard

Starting with version 3.0, a C++ compiler with support for the C++11 standard is required to compile and use Jem/Jive. Jem/Jive now support move semantics (where that makes sense), initializer lists, the `nullptr_t` type, literal operators (for the `idx_t` and `Time` classes, among others), and the `noexcept` and `override` function specifiers, among others.

A consequence of this change is that the `Ref` class now more closely resembles a standard pointer. Here are some examples:

```
Ref<Object>  obj = nullptr;  // No need to use NIL.

if ( obj == nullptr ) ...

if ( obj ) ...

if ( ! obj ) ...
```

This example shows how to use the literal operators provided by Jem:

```
using namespace jem::literals;

Time   t = 24_hour;
idx_t  i = 34_idx;

t = t + 60_min;
i = i + 128_idx;
```

The following example shows how you can use initializer lists:

```
Tuple<String,3>  s = { "one", "two", "three" };
Array<int>       a = { 1, 2, 3, 4, 5 };
```

The next example demonstrates how you can make use of move semantics:

```
Flex<MyClass>  list;  // Assume MyClass supports move semantics.

while ( condition )
{
  MyClass  tmp;

  // Initialize tmp ...

  list.pushBack ( std::move( tmp ) );
}
```

3 Extension of the `Properties` file format

The `Properties` file format now supports the use of the += operator to extend previously defines properties This is best explained through an example:

```
settings =
{
  list = [ "one", "two", "three" ];
```

```
    word = "Hello";
  };

  settings +=
  {
    more  = 10;          // Add a new member to "settings".
    list += [ "four" ];  // Extend the "list" array.
    word += " World!";   // Extend the "word" string.
  };
```

The += operator is particularly useful if you want to modify an existing properties file; you only need to add a section that modifies previously defined properties.