Help Topic: The MOOSApp Superclass

Spring 2020

Michael Benjamin, mikerb@mit.edu
Department of Mechanical Engineering
MIT, Cambridge MA 02139

1 The MOOSApp Superclass - Structure and Methods

Nearly all MOOS applications in the MOOS-IvP tree inherit from the MOOSApp superclass. This class facilitates the construction of a new application by providing a template of operation for configuration, reading mail and executing the application's main iterate loop. Here we describe the basic structure of this class, how to use it to build your own MOOS App, and useful utility functions defined on the MOOSApp.

Creating Your Own Application Using MOOSApp

Creating your own MOOS application based on the MOOSApp superclass is done by doing something similar to the below example for the fictitous `pFooBar` application.

```
#include "MOOS/libMOOS/MOOSLib.h"
class FooBar : public CMOOSApp
{
 public:
   FooBar();
   ~FooBar();

 protected:
   bool OnNewMail(MOOSMSG_LIST &NewMail);
   bool Iterate();
   bool OnConnectToServer();
   bool OnStartUp();
};
```

Of course the above `FooBar.h` file could be made by just creating the file and typing it in, but we generally advocate using one of the scripts for generating a "template" MOOS application:

```
$ GenMOOSApp [app-name] [prefix]
```

For example:

```
$ GenMOOSApp FooBar p
```

There is also a version for generating an AppCasting MOOS App:

```
$ GenMOOSApp_AppCasting FooBar p
```

In the MOOS-IvP code base, most newly created MOOS apps in recent years are of the AppCasting type. You can read more on AppCasting here: http://oceanai.mit.edu/ivpman/appcasting. This link also contains information on how to convert a basic MOOS app later into an AppCasting MOOS App if you prefer to start with a simpler structure for now.

The Primary MOOS App Overloadable Functions

MOOS apps publish, subscribe for, and handle mail passed from one application to another through the MOOSDB. The key components are shown in Figure 1 below. All MOOS apps begin by being a subclass of the MOOSApp superclass defined in the Oxford MOOS library (included with the MOOS-IvP distribution). The primary work of the app developer is in writing the three functions shown in the figure.

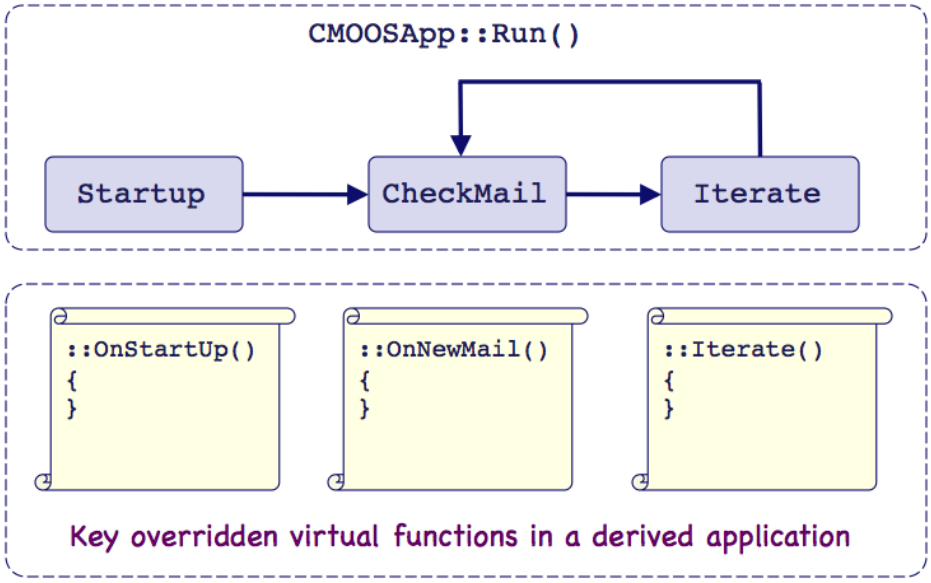

Figure 1: **The MOOSApp Key Functions:** All MOOS apps are a subclass of the MOOSApp superclass. Development mostly boils down to overriding the three functions below with the particulars of one's own liking.

MOOSApp Utility Functions

The Notify() function

This is the primary means for an app to publish mail to the MOOSDB. There are several versions of this function depending on whether a string, a double, or binary data is being posted, or if a "auxilliary" information is included in the posting. Normally a MOOS message will include automatically the time-stamp and source (the application making the post). But sometimes it's also useful to include auxilliary source information. For example, in pHelmIvP application, if a behavior is posting to the MOOSDB, the behavior name is contained in the auxilliary source.

```
bool    Notify(string varname, string value)
```

```
bool    Notify(string varname, double value)
```

```
bool    Notify(string varname, string value, string auxilliary_info)
```

```
bool    Notify(string varname, double value, string auxilliary_info)
```

```
bool    Notify(string varname, vector<unsigned char> binaray_data)
```

```
bool    Notify(string varname, vector<unsigned char> binaray_data, string auxilliary_info)
```

The `Register()` function

This is invoked within an app (typically upon startup) to register for mail, by naming a MOOS variable it is interested in. The `interval` indicates how often we want to receive mail. For example, if an application is posting mail at 20 Hz, but we are only interested in the latest value say once per second, then an interval of `1.0` willachieve this. By default the interval is zero meaning *all* mail is received for this variable, as fast as it may be published.

Registering for the same mail multiple times has no ill effect. However, if you wish to change the interval for an already-registered variable, you may need to unregister the variable first, and then re-register with the new frequency. An app may (unlikely) miss posted mail in the meanwhile. Very rarely, if ever, is this an issue.

```
bool    Register(string varname, double interval=0)
```

```
bool    Register(string varname_pattern, string appname_pattern, double interval=0)
```

The `UnRegister()` function

To unregister for mail, this function is used. If a user is no longer interested in certain mail, the same thing can also be achieved by simply ignoring mail of this type in the `OnNewMail()`. But if you're concerned about the size of your in-box, you can simply unregister for the mail using this function.

```
bool    UnRegister(string varname)
```

The `SetAppFreq()` function

```
void    SetAppFreq(double frequency, double max_frequency
```

The GetAppFreq() function

```
double  GetAppFreq()
```

The GetAppStartTime() function

```
double  GetAppStartTime()
```

The GetAppName() function

```
string  GetAppName()
```

The GetMissionFileName() function

```
string  GetMissionFileName()
```

The GetCPULoad() function

```
double  GetCPULoad()
```

The getIterateCount() function

```
int     getIterateCount()
```

The GetLastIterateTime() function

```
double  GetLastIterateTime()
```

The GetTimeSinceIterate() function

```
double  GetTimeSinceIterate()
```

The MOOSTime() function

```
double  MOOSTime()
```

The `GetMOOSTimeWarp()` function

```
double GetMOOSTimeWarp()
```