Introduction to AppCasting

June 2018

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

1 Overview

AppCasting provides an *optional* and powerful new way of delivering application status output beyond the traditional means of writing to standard I/O in a terminal window. It is motivated by a few common recurring observations:

- The biggest headache of users new to MOOS (e.g., students in MIT 2.680) was the derailment of a mission due to an unnoticed configuration or runtime error.
- Debugging typically involves re-launching with app terminal windows open and analyzing expected vs. observed output.
- When deploying multiple simulated vehicles, each with multiple MOOS apps, the number of open terminal windows may be unmanageable.
- When deploying a vehicle in the field, one cannot ssh in and see any application terminal output at all.
- Since terminal output is rarely viewable for the above practical reasons, apps are rarely designed with much thought put into their status output.

AppCasting is designed to make it easier to see application terminal output. This includes app-specific status messages, configuration and runtime warnings, and notable events. It is designed to allow appcast viewing tools to render this information and alerts on a single screen across multiple vehicles, each with several running apps, whether in simulation or the field. Having this form of information easier at hand, application developers will find it more rewarding to add thoughtful status reports to their application. Most applications in the MOOS-IvP tree have been converted to support appcasting. It's worth repeating that this is an opt-in feature of MOOS. All existing MOOS apps work just fine without appcasting. Appcasting apps and non-appcasting apps work side-by-side seamlessly.

2 MOOS Applications and Terminal Output

A MOOS application typically interacts with the world primarily through its subscriptions and publications to the MOOSDB. It may also interact through a terminal interface by generating status or debugging output, or in some rare cases, accepting terminal input.

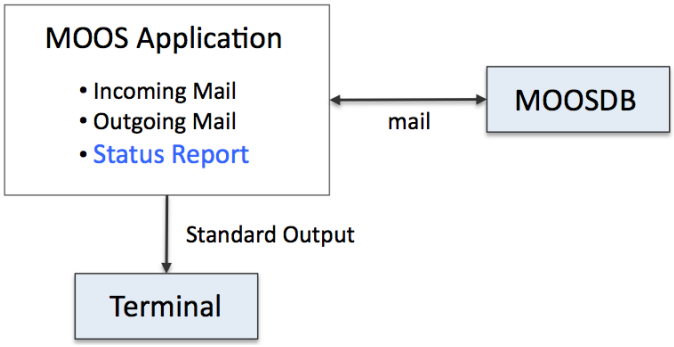

Figure 1: A typical MOOS application interacts with the world by publishing and receiving mail through the MOOSDB and by writing status messages to a terminal window, if one is open.

Even MOOS GUI applications have the option of opening a terminal interface in addition to the GUI. The terminal interface option is invoked by setting `NewConsole=true` for the given app in the `ANTLER` process configuration block in the mission file, or simply by just launching the app manually from the command line. With AppCasting, an application still generates terminal output, but does so by creating a report, in the form of an appcast data structure. The data structure may be converted to a list of strings, sent to the terminal, or a single long string, published to the MOOSDB.

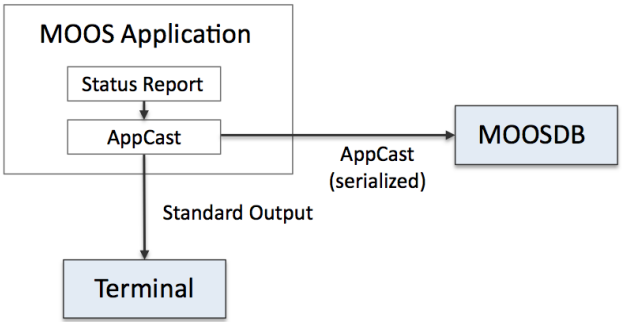

Figure 2: An appcasting MOOS app produces terminal by repeatedly generating and sending an *appcast* to the terminal. The appcast is a also serialized and published to the MOOSDB for other MOOS applications to consume.

By sending the appcast to the MOOSDB this makes a few things possible. First, even if the application was launched long ago without a terminal, it is now possible to launch a separate MOOS application (an appcast viewer tool discussed in Section 5) to start looking at the status output. Second, the same appcast data structure may now also be bridged to a separate off-board MOOS community, using something like `pShare`, so remote users may be alerted to or debug problems.

Third, the appcast data structure may contain configuration and run-time *alerts* to bring issues to the attention of operators quickly. Fourth, the appcast structure may be logged like any other MOOS variable, making it possible to review terminal output during the post-mission analysis phase.

3 Viewing AppCasts and Navigating AppCast Collections

A primary motivation for appcasting is the ability to view appcasts over several applications with a single viewer:

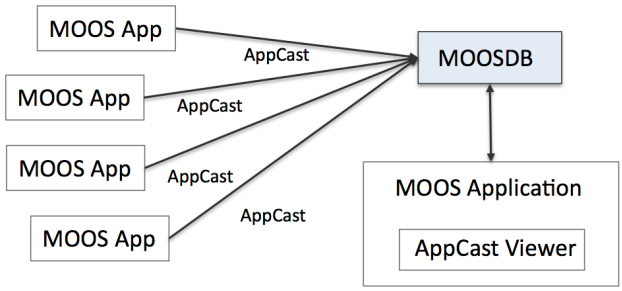

Figure 3: An AppCast Viewer is a separate MOOS application for gathering and navigating through appcasts from several other MOOS applications.

In addition to being able to see application output from a single window, it is possible to view application output for a particular app regardless of how that application was launched. In comparison, if the status output were only viewable from a terminal window, that app would have had to be launched with a terminal window from the outset. Furthermore, when the AppCast Viewer has a terminal interface, a user may log onto a remotely deployed vehicle and launch the viewer and see application output that would not be viewable otherwise since applications on fielded platforms are never run with terminal windows open.

An AppCast Viewer may also handle appcasts from several vehicles as conveyed in Figure 4. The viewer lets the user navigate between different vehicles and appcasts within a vehicle. The interface is discussed in a later section, but the idea is shown on the right in Figure 6.

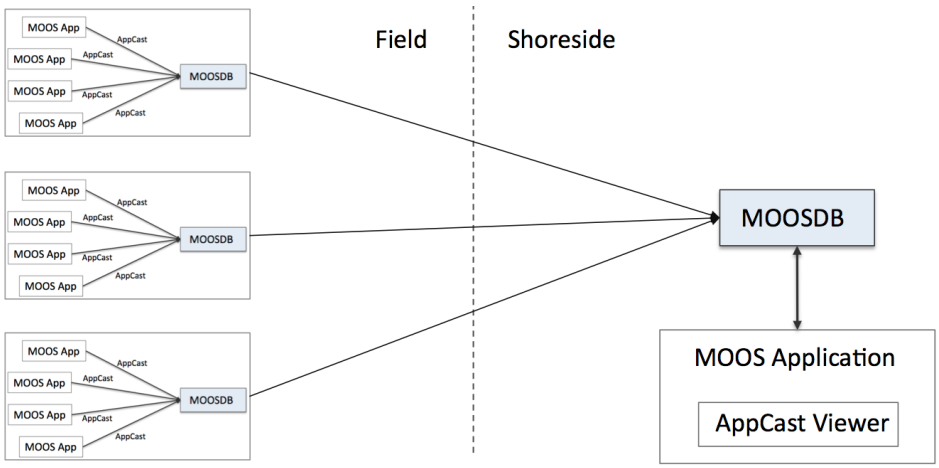

Figure 4: An AppCast Viewer may also be used for sorting and navigating through appcasts from several applications *over several vehicles* or nodes. The appcasts may be bridged from one community to a single shoreside community using a tool such as `pShare`.

The above arrangement assumes that appcasts are sent from each MOOS community to a single *shoreside* MOOS community, to which the appcast viewer is connected. This may be done with the `pShare` utility. The uField Toolbox also provides a set of MOOS utilities to facilitate this kind of arrangement.

4 The AppCast Data Structure

An example appcast is shown in Figure 5. It has four distinct parts:

- **config warnings:** Configuration warnings are typically generated during the application's OnStartUp() routine.
- **run warnings:** Run warnings may be generated any time during the execution of the application.
- **general messages:** A list of app developer-formatted strings having whatever the application developer thought would best constitute a succinct meaningful status report.
- **events:** A set of time-stamped events. Exactly what constitutes an event is determined by the application developer.

Figure 5: An appcast consists of four main parts: (1) configuration warnings, (2) run-time warnings, (3) general status report messages, and (4) run-time events. The bar at the top of the figure is rendered in red due to the presence of a run-time warning. The `"1/1"` on this line indicates one configuration warning and one run-time warning. The `"(150)"` on this line indicates that it is iteration #150 for this application. This image is a screen shot take from the `uMACView` utility described later.

For any given appcast, all fields are optional. Indeed, often an appcast will be devoid of any configuration or run warnings. It is also not uncommon for an application not to have a notion of an event.

For reasons explained later, an appcast instance is typically created once upon application startup. The block of general messages is cleared and overwritten each time an appcast is generated. Run warnings and events are added any time during the application operation, but are limited in amount (first-in-first-out/FIFO). This is done to ensure against unbounded growth of the appcast message, and relieve the app developer from addressing the logic of bounded message growth. Configuration warnings are unbounded however since they are only generated at startup time and are typically bounded from above by the number of application configuration lines.

5 A Preview of AppCast Viewing Utilities

An appcast viewer is primarily a utility for viewing appcasts, rendering an appcast to look something like that shown in Figure 5. It also does a couple other important things. First, it provides a mechanism to allow the user to navigate between incoming appcasts from multiple vehicles, each with multiple applications. Second, it implements, under the hood, a protocol between the appcast viewer utility and the applications, to ensure on-demand appcasting.

The `uMAC` utility shown on the left in Figure 6 is run from a terminal window. The `uMACView` utility shown on the right is a GUI with a bit more capable interface, allowing the user to see all vehicles, all apps for a chosen vehicle, and the appcast for a single chosen application. The advantage of the `uMAC` utility however is that it may be launched remotely after logging in to a vehicle that may otherwise be unresponsive and in need of some debugging.

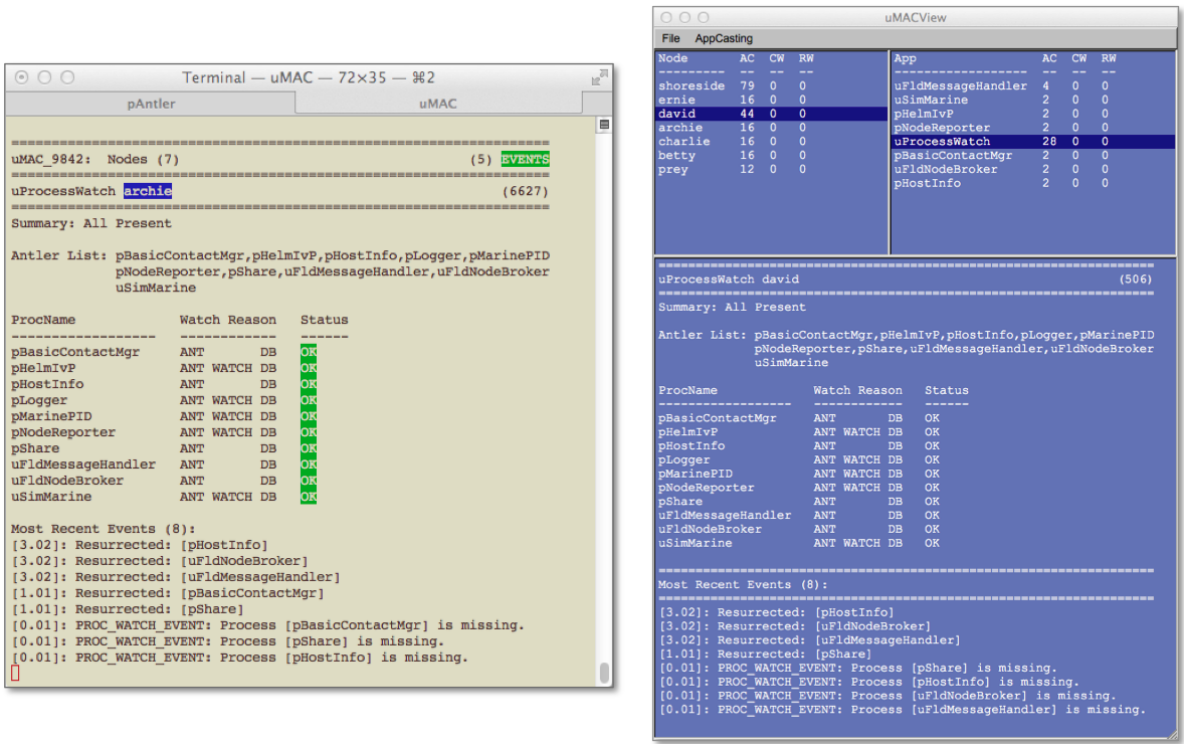

Figure 6: Two appcast viewer utilities: On the left is the terminal-based `uMAC` utility. On the right is the GUI-based `uMACView` utility. Both provide access to the same information. While the latter has a bit nicer user interface, the former may be run remotely while ssh'ing into a fielded vehicle.

While the `uMAC` and `uMACView` utilities are completely stand-alone and do not assume the use of any other tool, a third option exists for users of the `pMarineViewer` tool. This tool has been augmented to support an integrated `uMACView` style interface into the same single window as shown in Figure 7. The appcasting interface may be toggled on and off by simply hitting the `'a'` key. The user may also specify the mode upon startup.

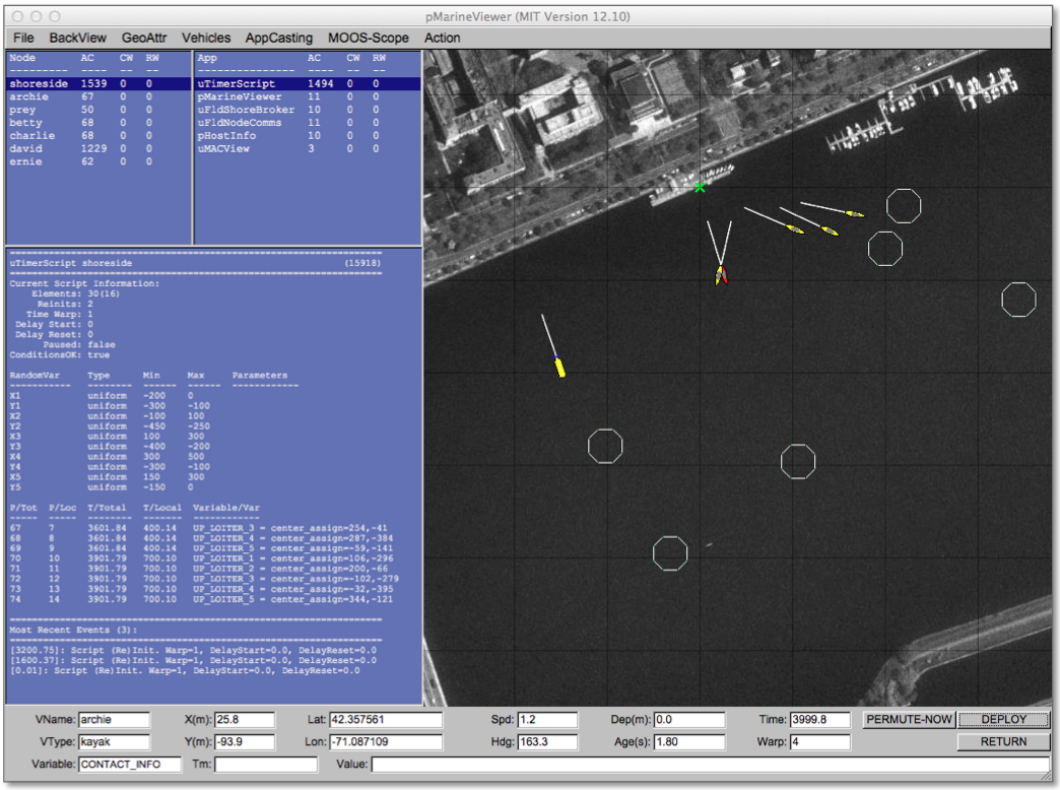

Figure 7: The pMarineViewer application has been augmented to support appcast viewing in a separate window pane. The interface is nearly identical to that of the uMACView application. The integration is for better convenience to existing pMarineViewer users. The appcasting information may be toggled on and off with the 'a' key.