

# Under the Hood of On-Demand AppCasting

June 2018

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

---

|          |  |          |
|----------|--|----------|
| <b>1</b> | <b>Overview</b>  | <b>1</b> |
| <b>2</b> | <b>Motivation</b>  | <b>1</b> |
| <b>3</b> | <b>AppCast Generation Criteria</b>                       | <b>2</b> |
| <b>4</b> | <b>Terminal Switching</b>                                | <b>2</b> |
| <b>5</b> | <b>AppCast Requests</b>                                  | <b>3</b> |
| <b>6</b> | <b>Limiting the AppCast Frequency</b>                    | <b>4</b> |
| <b>7</b> | <b>Generating and AppCast vs. Publishing and AppCast</b> | <b>5</b> |
| <b>8</b> | <b>Monitoring AppCast Traffic Volume</b>                 | <b>5</b> |

---

## 1 Overview

On-demand appcasting refers to the goal of minimizing generated appcasts, ideally only when there is a reasonable chance that an appcast will be tended to (looked at) by a user. Users may be reading an appcast either by looking at terminal output or through a uMAC utility.

## 2 Motivation

Consider the following scenario:

- 20 simulated vehicles,
- 10 MOOS applications on each vehicle,
- Each application running with an apptick of 4Hz,
- MOOS time warp running at 25x real time.

In the above scenario (a conceivable scenario in our lab), 20,000 appcasts would be generated *per second*. Consider a second scenario:

- One fielded underwater vehicle,
- 10 MOOS applications,
- Each application running with an apptick of 4Hz,
- Mission duration 6 months.

Although only 40 appcasts per second are generated, the vehicle is underwater and, limited to acoustic messages, likely no appcast will ever be viewed. With a six month mission, the CPU time and power budget needed to generate those 40 reports per second may come under scrutiny.

### 3 AppCast Generation Criteria

An appcast will be generated on any given iteration only if the contingencies and criteria depicted in Figure 1 are met. In short, an appcast is generated if either a terminal is open or a uMAC tool is requesting the appcast. Even when appcasts are being generated, they may be generated less frequently than each iteration, to be more in line with the frequency a user is able to process refreshed report information. These issues are discussed next.

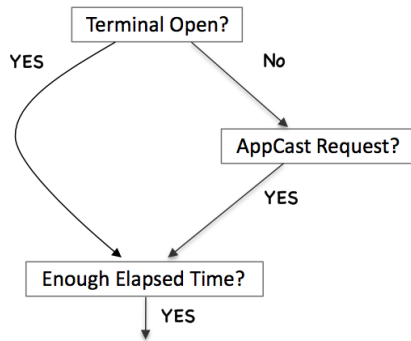


Figure 1: The appcast-generation decision is based on a few checks and contingencies. An appcast is generated only when tended to by a user, and only as often as a user may reasonably expect to process updated reports.

### 4 Terminal Switching

In the flow diagram of Figure 1, the first step in determining whether an appcast is to be generated involves the presence of a terminal. If an app is launched with a terminal window open, or launched within a terminal window, appcasts should be generated. In this regard, appcasting apps should behave like non-appcasting apps, although the former produce its output by different means. From the user's perspective, launching an application with or within a terminal window should result in the same familiar behavior regardless of the application.

Upon startup an AppCastingMOOSApp application will try to determine if information directed to `stdout` will make its way to an open terminal window. The following reasoning is applied:

- By default the application assumes information directed to `stdout` is indeed being rendered in a terminal window.
- During startup, when mission file parameters are examined, if the application detects the presence of a global parameter, `TERM_REPORTING`, and it is set to `"false"`, then the application assumes that a terminal window is not open to receive output written to `stdout`.
- During application startup, the following library utility function is invoked:  
`int isatty(int);`  
 This function is defined in `"unistd.h"` and is able to detect whether `stdout` is receiving output.

The above ensures that the application will err on the side of always producing appcasts/output to the terminal. To ensure otherwise, make sure to include `TERM_REPORTING="false"` in the MOOS configuration file. The last check is just a convenience or extra fail-safe; if an application is launched with `pAntler` using `NewConsole=false` and `pAntler` itself is also launched with `stdout` redirected to `/dev/null/`, then the application will automatically detect this. This style of launching is actually a fairly common scenario in launching MOOS communities on vehicles in our lab. In detecting this situation, the application will not generate an appcast unless a uMAC tool is explicitly requesting it. This is the next step in the flow diagram of Figure 1, and discussed next.

## 5 AppCast Requests

An *appcast request* is message sent to an application to begin or continue generating appcasts for some specified period of time. The request may originate in the local MOOSDB community, or in a remote MOOSDB community as the two cases suggest in Figure 2.

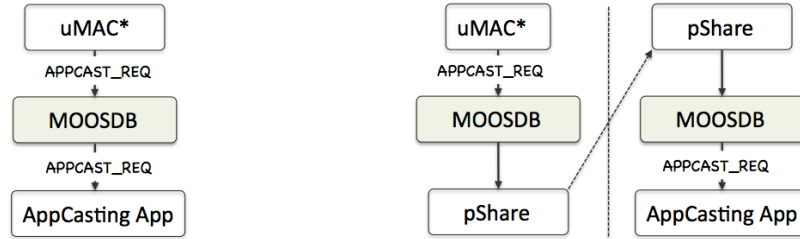


Figure 2: An appcast request requests that the receiving MOOS application begin or continue to generate appcasts for some specified period of time. The request may come from an app within the same MOOS community, or from an external community.

The content of an appcast request has the following fields:

- **node:** This must match the name of the MOOS community within which the application is running, otherwise the appcast request is ignored. If the `node` is "any", the match is always granted.
- **app:** This must match the name of the MOOS application receiving the appcast request, otherwise the request is ignored. If the `app` is "any", the match is always granted.
- **duration:** This number is given in seconds and represents the duration the appcast request would be honored after time of receipt. The maximum value is 30.
- **threshold:** The threshold name indicates under what conditions the receiving application should generate an appcast. The two possible values are "any" and "run\_warning". Their meaning is discussed below.
- **key:** The key helps the receiving application discern appcast requests from different applications.

An example `APPCAST_REQ` message:

```
APPCAST_REQ = "node=henry,app=uProcessWatch,duration=3.0,key=pMarineViewer:shore,thresh=any"
```

## Node and Application Name Matching

An appcast request will be ignored by a receiving application if the request does not *match*. The match must be made between both (a) the requested and actual node name, and (b) the requested and actual application name. A requested value "any" will match to anything. In most circumstances the requesting application, e.g., `uMAC`, `uMACView` or `pMarineViewer`, will publish requests naming nodes and applications explicitly. Upon startup however, requests may be sent broadly to all nodes and all applications just to learn about existing appcasting sources before beginning to make requests explicitly.

## Duration Time

An appcast request comes with a duration. If the requesting application disconnects, the duration caveat ensures the original request will timeout before too long, avoiding the perpetual generation of now unwanted appcasts. Typically if a `uMAC` tool is monitoring the appcasts of a particular application, it repeatedly sends an appcast request to that application, each time refreshing the timeout criteria.

## Request Threshold

The appcast request will specify one of two criteria to the application. First, if the criteria is "any", the application is to publish an appcast on each possible occasion. This may not be the same as each iteration, since a further minimum reporting interval may be applied, as described next in Section 6. The second threshold type is "run\_warning". This indicates to the receiving application that it should only generate an appcast when a new run warning has been added since the last appcast. Typically a `uMAC` tool will run in a mode sending appcast requests with the latter threshold to virtually all nodes and apps, but appcast requests using the former threshold to a single node and app chosen by the user.

## Request Key

AppCast requests specify a particular key, presumably a string that is unique to the originator. This allows the receiving application to do separate bookkeeping for each requesting application. As long as the appcast request threshold matches, hasn't timed out, and met the threshold criteria for *one* of its logged keys, then it indeed meets the criteria.

# 6 Limiting the AppCast Frequency

A third criteria is applied to the decision of generating an appcast on any given iteration. This criteria is depicted at the bottom of Figure 1. Even if a terminal window is open, or a valid appcast request has been received recently enough, the generation of an appcast may still be skipped if the previously generated appcast was generated too recently. AppCast content is meant to be human-readable, and humans can only read so fast. There is no sense in updating a terminal report say 50 times per second. Although most MOOS apps are typically not configured with an apptick of 50Hz, an apptick of 5 is fairly common, as well as a `MOOSTimeWarp` of say 10 or greater. By default, appcasting applications are limited in real-time frequency to once every 0.6 seconds.

This number just reflects a number that, from experience, feels right for an update frequency. This can be overridden for any application with the following parameter in its configuration block:

```
term_report_interval = N
```

where  $N$  ranges from zero (appcast generation only limited by the iteration frequency) to at most 30 seconds.

## 7 Generating and AppCast vs. Publishing and AppCast

The flow chart shown in Figure 1 addresses the issue of whether or not an appcast is *generated*. Whether or not the appcast is *published* is a separate issue. Simply put, if a terminal window is open for the application, and it has not received any appcast requests, the appcast is generated (and rendered to the terminal `stdout`) but not published.

Below is informal logic shorthand for policy and conditions described over the last few pages. First, on the policy of whether an appcast is generated:

```
generate_appcast = (terminal || appcast_requested) && !recent_appcast
```

Next, on the question of whether an appcast is published:

```
publish_appcast = generate_appcast && appcast_requested
```

Both of the above depend on the term `appcast_requested` from the following expression:

```
appcast_requested = unexpired_request && ((request_threshold == "any") || new_run_warning)
```

The terms in this last expression will hopefully be apparent from the preceding pages.

## 8 Monitoring AppCast Traffic Volume

The uMAC tools provide a means for verifying that on-demand appcasting is behaving. These same tools are also useful for catching other anomalies. The information in Figure 3 below focuses on the information at the top of the uMACView tool depicted in Figure 3.

The screenshot shows a window titled "uMACView" with a menu bar containing "File" and "AppCasting". Below the menu bar is a table with two main sections: "Node" and "App". Each section has columns for "AC", "CW", and "RW".

| Node      | AC | CW | RW | App                | AC | CW | RW |
|-----------|----|----|----|--------------------|----|----|----|
| shoreside | 79 | 0  | 0  | uFldMessageHandler | 4  | 0  | 0  |
| ernie     | 16 | 0  | 0  | uSimMarine         | 2  | 0  | 0  |
| david     | 44 | 0  | 0  | pHelmIvP           | 2  | 0  | 0  |
| archie    | 16 | 0  | 0  | pNodeReporter      | 2  | 0  | 0  |
| charlie   | 16 | 0  | 0  | uProcessWatch      | 28 | 0  | 0  |
| betty     | 16 | 0  | 0  | pBasicContactMgr   | 2  | 0  | 0  |
| prey      | 12 | 0  | 0  | uFldNodeBroker     | 2  | 0  | 0  |
|           |    |    |    | pHostInfo          | 2  | 0  | 0  |

Figure 3: The uMAC tools include tallies of the number of appcasts received for each node (vehicle) and each application. The above is from the top of the `uMACView` and appcast tallies are shown under the "AC" column.

From the above figure, one might ask why so many appcasts have been received when the user is only focusing on one vehicle and one application. There are few answers to this question and each, listed below, are related to the need for a uMAC utility to *find* existing sources of appcasts. After all, how can it send an appcast request to a particular vehicle and particular application if it doesn't know that it exists?

- When an application starts up, it generates appcasts for the first few iterations. Even if no one is tending to this information, a few iterations worth of un-tended appcasts is not harmful. In practice this helps uMAC tools discover appcasting apps.
- When a uMAC tool starts up, it posts an appcast request to all known nodes and all known applications. The uMAC tool doesn't need to know or keep track of vehicles, but just simply posts `APPCAST_REQ="node=all,app=all, ..."` to its local MOOSDB. Other applications have the responsibility of bridging this variable to other vehicles as they are discovered.
- Each time a uMAC tool receives a new appcast from a vehicle/node it has never heard from before, it responds by posting an appcast request back to that vehicle for all applications, e.g., `APPCAST_REQ="node=henry,app=all, ..."`. This request will time-out shortly, but is usually sufficient to learn about all applications on that node. Subsequent requests are then more selective.