Winnipeg Transit EBDD RFP Technical Appendix
Controller and Communications –R1

1 Controller Logic

Winnipeg Transit will build a back-end application to service information requests from the display controllers. This section outlines a potential controller/application server communications protocol. The exact format of the messages (e.g. encoding, compression, etc.) can be finalized at a later time. Indeed, modern application design dictates that the underlying message protocol is to be abstracted out of the application's core logic, which isolates behaviour logic from data access logic.

This proposal describes a "pull" model for a display controller consisting of several subsystems. These subsystems would ideally be implemented as independent services/processes spawned from a main boot/monitoring process.

NB: While the implementation details of the controller's operating system are largely irrelevant to Winnipeg Transit, it is expected that the controller will run an embedded version of a standard operating system such as Microsoft Windows CE or a Linux distribution. A custom-built, one-off operating system would equally acceptable. What is important is for the controller to be able to communicate with Winnipeg Transit's back-end application servers according to the protocols defined herein.

1.1 Subsystems

The controller's functions can be described in terms of semi-independent subsystems, each responsible for their own tasks yet all working together in symphony. This section also describes the order that the subsystems are to be started during initialization/reset or during recovery from a malfunction.

1.1.1 Networking

Immediately upon starting up, the boot process should start the networking subsystem. This fundamental subsystem is responsible for acquiring via DHCP the TCP/IP address and related information such as netmask, gateway, etc. Note that due to VPN network limitations, DNS information such as name servers and search domains can NOT be acquired via DHPC and must be manually configured (see below).

1.1.2 Configuration

The controllers must provide an area of non-volatile storage and/or memory that will contain sign-specific configuration values, including:

- The sign's stop number (or list of stop numbers, see Schedule Data Query)
- The application messaging URLs (see Messaging Protocol/Structure)
- Any other required URLs (TBD)
- DNS information including name servers and search domains (see Networking)
- Web management credentials (see Configuration Management)
- Any other information that must persist across occasional losses of power, restarts, etc (TBD)

The non-volatile storage could be any storage medium that is capable of reliable operation within the defined environmental parameters. Without limiting the options, this could include a flash storage card (CF, SD, etc), a solid state USB storage device, or an embedded memory chip/card.

The default configuration values should be blank/zero/null as appropriate so it is easily determined whether the controller has been configured or not; alternatively, a separate value could be used to indicate whether configuration has occurred.

If the configuration information has not been set, then further subsystem initialization should not continue. In this case, the message console should display “Awaiting Configuration” along with the sign’s IP address so that it can be recorded by the installer along with the stop number before blanking the display after a reasonable period of time (e.g. 30 seconds).

1.1.2.1Configuration Management

If the networking subsystem starts successfully, a simple configuration management web application should be started that listens for incoming http requests on port 80. This management application will allow remote administration of the controllers, for example, from the Transit base. The application should be secured to prevent unauthorized access, and allow the ability to change the credentials at any time.

The application should present the user with a simple HTML status page that displays basic system information such as system uptime, current time, temperature inside case/controller, IP/MAC address or ESN, and anything else that is available and might be relevant (e.g. the config parameters). There should also be a “restart” button which will cause the system to restart itself if required, as well as an “update schedule” button which will force the controller to immediately re-request schedule data from the application servers and update the display.

To enable convenient automated monitoring, this status page should be returned in XML format if the request’s “format” parameter is set to “xml”.

Another page must be provided that presents an interactive HTML form allowing the current configuration values to be updated. After input validation to ensure that the entered values make sense (validation rules TBD), the new values should be written to non-volatile storage, causing the new parameters to take effect.

Finally, a firmware upgrade page will allow the controller’s operating system and/or application to be upgraded remotely should enhancements or bug fixes become available.

If the inclusion of an embedded web server is not practical, the ability to telnet/SSH to the controller would be an acceptable alternative as long as all of the functions described in this section were still available.

1.1.3 System Clock

The *system clock* subsystem is responsible for keeping time internally and feeding time to the display. This subsystem periodically queries the application server to ask for the current time. The response to this request includes the current local time and the number of seconds before the time should be queried again; this allows the application server to dynamically throttle time updates as opposed to having some preset value hardwired into the signs.

The clock subsystem can be started if the networking subsystem starts successfully and the configuration values have been set.

It is important to note that we would prefer a time service of our own creation (as opposed to a NTP client) so we can ensure that the displays are perfectly synchronized with our schedule data.

1.1.4 Schedule Controller

The final subsystem is the *schedule controller*, which is responsible for requesting schedule data and updating the display. This subsystem periodically queries the application server to request schedule data for the stop. The stop number is passed to the server so it returns only relevant data. The response includes all the information on the upcoming arrivals, including the route numbers and names, via/destination info, and the expected arrival times. The application server can determine how much information should be returned for a stop (i.e. how far into the future to look, the number of passing times for a route, etc). Like the system clock query response, the schedule query response also includes the number of seconds before the schedule data should be queried again, which provides a great deal of flexibility, including the ability to dynamically throttle updates depending on such variables as the time of day, whether there are any schedule exceptions in effect, and even the stop number.

The schedule controller subsystem should be started after the system clock subsystem has started.

Note that the schedule controller could be broken down into two subsystems, one which deals entirely with retrieving schedule data, and another which deals entirely with updating the display. Under this architecture, the schedule controller would signal the display controller to refresh the display whenever new data were retrieved, as would the system clock when the minute should be incremented.

1.1.5 Other

Other subsystems can be considered to handle things like abstracting the communication process, monitoring and reporting status/error conditions, etc.

1.2 Logging and Analysis

The application server can record every request received from the signs, which allows for detailed analysis and diagnostics. For example, the system could alert maintenance staff if a stop were to miss sending queries as scheduled.

1.3 Controller Diagnostics

A physical switch (toggle or momentary) on the controller is required for diagnostic purposes. When pressed, the display would show the controller's IP address and stop number. This will be invaluable for troubleshooting issues should the need arise.

1.4 Messaging Protocol/Structure

Simple REST-style XML web services are an excellent candidate for the underlying encoding for a variety of reasons:

- They are ubiquitous and hence available to a large number of application environments
- They rely on simple, proven technology e.g. the HTTP protocol and web servers
- They are human-readable, making it easy to diagnose problems
- They offer a low-overhead web services solution when compared to traditional SOAP web

services

1.4.1 Time Query

The time request would look something like this, using stop 10064 as an example:

```
GET http://ebdd.winnipegtransit.org/time/10064
```

The response would be well-formed XML like the following:

```
<time local-time="2008-08-12T12:34:56" next-query="900" />
```

The controller would parse the date and time info in the `local-time` attribute and use that to reset the current time. The value in `next-query` would be used to determine the number of seconds from now when the time should next be queried.

1.4.2 Schedule Data Query

The schedule data query for stop 10064 would look something like this:

```
GET http://ebdd.winnipegtransit.org/schedule/10064
```

The response might be:

```
<schedule next-query="60">
  <route name="16 Selkirk">
    <time departure="2008-09-12T12:47:00" destination="Burrows" via="">
    <time departure="2008-09-12T13:05:00" destination="McPhillips" via="">
    <time departure="2008-09-12T14:05:00" destination="Burrows" via="">
  </route>
</schedule>
```

The meaning of this response should be readily understandable. Depending on the stop and time of day, there could be multiple `<route>` elements returned. The application server would determine how many `<time>` elements should be returned. As with the time query, the `next-query` value would be used to determine when the schedule data should next be polled.

In cases where a sign is servicing multiple stops, the query might look something like this:

```
GET http://ebdd.winnipegtransit.org/schedule/10064,10065
```

In every case, the list of stops (in this case, “10064,10065”) is simply the verbatim configuration value for the stop number. The application server will recognize multiple stop queries and will incorporate their individual results into a single response, transparent to the controller.

The controller should not make any assumptions as to the ordering/sorting of the elements and attributes nor to the presence of elements and attributes. For example, times within a route may not be in chronological order, routes may not be sorted by route name, and the `via` attribute may be omitted entirely if it is blank to minimize the amount of data transmitted.

Additional scheduled information could be returned as required.

1.4.3 Other Messages

Other messages type may be required. Examples could include “heartbeat” messages, error reporting messages, etc.