HCA Technical Note: HCA Server Protocol

Document Version: February 12, 2016

HCA Version: 13.0

Changes:

09-Oct-2012: Changed the HCAApp.GetDisplays method to now take block # argument like GetDesign

09-Oct-2012: Fixed an error in the documentation for the return of GetDesign. It never did return the block number in the result message as was previously documented.

12-Nov-2012: Changed the Update message to include the icon state if a temporary icon image is in use

12-Nov-2012: Changed the GetDesign and GetDisplays method to return the icon state if a temporary icon image is in use.

12-Nov-2012: Added the HCAApp.GetFile method

05-Jan-2013: Added the HCAApp.TimeStamp and HCAApp.RefreshState methods

12-Jan-2013: Updated documentation to correct errors and expand explanations.

31-Jan-2013: Fixed the documentation for GetDisplayGraph and ThermostatState.

09-Mar-2013: Fixed typos. Added some additional info on the timestamp and refresh methods

10-Mar-2013: Added the Additional notes section

14-Mar-2013: Added a section on Web Sockets

HCA 12 – Protocol "C" changes

23-Dec-2013: Fixed several documentation errors in the HCAApp methods

02-Jan-2014: Added HCAApp.GetDisplayMRU

17-Jan-2014:

- Added to the data returned for an object by HCAApp.GetDesign.
    - Added wattage. This data is supplied for all objects. For non-device objects it is zero.

    - Added “no show”. For protocol A and B and object marked as “no show” is filtered out from GetDesign. For protocol C all objects are returned but some may be marked as “no show”

- Changed SetClientOptions to include new options
- Changed GetFile to take two optional arguments: File size and File checksum which will allow checking to see if the client copy of the file is the same as on the server. Only used when retrieving files and not icons.
- Add additional information to the data returned for a display by GetDisplays:
    - Added Background information (type and path
    - Added “no auto close” flag
    - Added the display label.
- Added wattage to device info in HCAApp.Update message
- Added HCAApp.GetInspectorReport
- Added HCAApp.FormatText
- Added HCAApp.GetDisplayTiles
- Added HCAApp.GetScheduleNames
- Added HCAApp.SetCurrentSchedule
- Added HCAApp.GetServerStatus
- Added server to client message HCAApp.UserDialog
- Added client to server message HCAApp.UserDialogReport
- Added server to client message HCAApp.TileUpdate
- Added server to client message HCAApp.PlaySound
- Added server to client message HCAApp.TextToSpeech
- Added server to client message HCAApp.ServerStatus

31-Jan-2014: Updated some descriptions

07-Feb-2015

- Changed GetInspectorReport to return only inspector messages
- Added GetAlertReport to return the alert report
- Added GetLogFilters
- Added GetLog
- Added LogAdd Update message
- Changed SetClientOptions to add an option for log update messages
- Minor correction to the section on GetScheduleNames

25-April-2015

- Improved the description in GetDisplayGraph
- Added HCAApp.GetExtServerStatus
- Added new data returned by ThermostatState
- Added HCAApp.ThermostatChange
- Added description to ThermostatState about what is returned for the setpoints and error results.
- Added HCAApp.GetThemeIcons
- Added HCAApp.GetThemeIcon
- Changed GetDisplays to return an additional item for V13
- Change GetServerStatus for V13 to return number of alerts as well as the count of yellow and red inspector messages.
- Changed TileUpdate message to pass additional data items
- Changed SetClientOptions to control the ExtServerStatus message
- Added the ExtServerStatus update message
- Added the TextDisplayChange update message

- Added the DisplayChange update message
- Change ServerStatus message to return number of alerts as well as the count of yellow and red inspector messages.

11-Feb-2016

- Updated GetDesign and GetDisplay to show returned Alexa info

Table of Contents

1 Overview

This technical note describes the method you can use to interface to HCA in the same manner as *HCA for Android*. This communication method allows for invoking the HCA Object interface but doing it remotely over a TCP/IP connection.

Very important note you must read: This is largely an internal interface to HCA. It is being documented for the convenience of members of the HCA User Community. It may change at any time in any way. Be prepared for that. The HCAObject group is probably safe to use as it is based upon the HCA Object model which is a published specification that we try and make only upward compatible changes to it. The HCAApp group may be more subject to change.

Second very important note: Much of this is based upon the HCA Object model that is documented in HCA User Guide *Object Model* appendix. You really need to understand that to use this. Some items, like how passwords work and error handling of object methods is all done in the same manner. So please review all that before you go through this.

All communication with the server is in text format over a standard IP connection talking to the same port number as the server is configured to listen on.

2 The Protocol

2.1 Establishing a connection

After a TCP session has been established to the server, the fist message sent to the server is always **exactly 16 bytes in length** and consists of this:

Byte 1	Always 'H'
Byte 2	Always 'C'
Byte 3	Always 'A'
Byte 4	Always 0
Byte 5	Always 0
Byte 6	Always 0
Byte 7	Protocol identifier. Set to 'A' or 'B' or 'C'
Byte 8-10	Major version number of client
Byte 11-13	Minor version number of client
Byte 14-16	Build number of client

The HCA Server responds back with **exactly** 16 bytes containing:

Byte 1	Always 'H'
Byte 2	Always 'C'
Byte 3	Always 'A'
Byte 4	Return code

Byte 5	Client number
Byte 6	Needs password. If this is non-zero a password must be set using the HCAObject method for doing that before you do much else. You also must check for error returns from this method because the password may not match what is set in the file.
Byte 7	Protocol number. Echo of what sent in your message
Byte 8-10	Major version number of server
Byte 11-13	Minor version number of server
Byte 14-16	Build number of server

The server validates the message and if there is a problem the return code – byte 4 of the returned message - is non-zero. This could be because the message isn't correct, the protocol number isn't valid, or the **server rejects the version number of the client**.

Note: It is expected that clients test the return code and if a non-zero number report that to the user**. It is also expected that the client will validate the version number of the server against what it expects**.

Return Code #	Description
0	Success
1	First three characters for the connect message are not HCA
2	Not valid protocol. Currently only 1, 2, 'A', 'B', 'C' are valid. 2 and 'A' are the same.
3	Server doesn't support client version
4	Client has disconnected before message could be processed

Figure 1: Connection Return Codes

Once this first message has been sent and the reply received then requests can be sent to the server. In most cases – except for the update and notify messages described later – all interactions between the client and the server are synchronous. The client sends a request to the server and the server responds. There are no client messages sent to the server that do not generate a reply from the server.

2.2 Message Format

The message format between the client and the server is all ASCII based. A format is used to pass one or more parameters to the server. The message format is:

<preamble><4 spaces><data>

The preamble consists of one or more four character groups. Each group provides the position of the last character of the data for that parameter. For example:

```
00370055007600770078007 9    HCAObjectDevice.RockerPressBedroom - Bath Lights000
```

This message contains 6 parameters. This is determined as the length of the string before the 4 spaces. In this example, it is 24 characters, and 24 / 4 = 6. Character 79 is the last character –the first character is character number 1.

Parameter #	Parameter Location Value
1	0037
2	0055
3	0076
4	0077
5	0078
6	0079

Figure 2: Example Message Parameter decoding

Note: This is how you can tell if you have a complete message when you are receiving messages from the server. You don't have a complete message until you have received enough characters to have received the 4 spaces. Then you can look at the preamble and know how many characters are in the whole message.

The 1st parameter goes from the 1st character after the 4 spaces – which in this example message is located at position 29 – up until position 37. The '37' comes from the 1st 4 character group in the preamble. The next parameter starts at the next character after the last character of the previous parameter. In this case, that is character 38 and it ends at character 55 – which comes from the preamble. And so on. Decoding the whole string is this:

Parameter Location	Parameter Value
29 to 37	HCAObject

38 to 55	Device.RockerPress
56 to 76	Bedroom – Bath Light
77 to 77	0
78 to 78	0
79 to 79	0

Figure 3: Example Parameter Values

It is possible to have a zero length string for a parameter in which case the string ends where it began. In this example preamble show below, the last three parameters are all empty.

```
00370055007600770077007700
```

Note: There is no type information in the string. You have to know, based upon the documentation, what type of data each parameter is and convert it from text as necessary.

There are two programs that are installed with HCA: **ENSDisplay.exe** and **ENCompose.exe**. There are not accessible from the Start menu but you can browse to the installation and they are in the "Program" folder. You can use these programs to better learn how the encoded strings work.

Using ENSDisplay.exe you can break an encoded string into its component parts. If the parts are themselves encoded strings, you can cut them out of the display, and paste them into the input and decode them.

You can use the Remote Access Viewer – described later – to listen in on the communication between the server and the Android application, save the messages sent and responses to a text file and then cut and paste into ENSDisplay to see how the various messages and the responses are encoded.

The ENCompose program lets you enter a series of strings that are then combined into one encoded string.

Below is a decode algorithm written in C with MFC string handling.

```
static BOOL DecodeParameterList (CString s, Parameter* *pList, int *pctParams)
{
   int iSep = s.Find("    ");

   if (iSep == -1)
      return (FALSE);

   // Extract the part before the 4 blanks
   CString pspec(s.Left(iSep));
   CString sEnd;
   CString text;

   // Make sure it contains a multiple of 4 characters
   if ((pspec.GetLength() % 4) != 0)
      return (FALSE);

   // Determine count of parameters.  Each one is defined by 4 characters
   int ctParams = pspec.GetLength() / 4;

   // Start of the parameter.  Moved along as each is extracted
   int iStart = iSep + 4;
   int iEnd;

   Parameter *pParams = new Parameter[ctParams];
   Parameter *p = pParams;

   for (int i = 0; i < ctParams; i++)
   {
      sEnd = pspec.Mid(i * 4, 4);      // Extract out the spec for this parameter

      if (!ConvertToInt(sEnd, &iEnd))  // Make into number form.  Must convert ok
      {
         delete[] pParams;
         return (FALSE);
      }

      if (iEnd > s.GetLength())        // This number tell us the end character.  So
                                       // makke sure we have sufficient in the string
      {
         delete[] pParams;
         return (FALSE);
      }

      // Extract the text for this parameter.  Could be a null string
      if ((iEnd - iStart) == 0)
         text = "";
      else
         text = s.Mid(iStart, iEnd - iStart);

      p->text = text;

      p++;
      iStart = iEnd;
   }

   *pList = pParams;
   *pctParams = ctParams;

   return (TRUE);
}
```

Figure 4: Sample C code for decoding of messages

This parameter encoding method is sufficient for all the messages passed in the HCAObject group. Messages in the HCAApp group expand upon this slightly. By reading the preamble, the string for a given parameter is extracted. That's what the above algorithm does. Normally this extracted parameter is a number or a text string but it could also be another set of parameters encoded in the same manner. This allows one parameter to be an "array" of data.

To work with this encoding method, take the string received from the server and extract out each parameter using the above algorithm. If you know that the parameter is an array, then again use the above algorithm to break that string down into its parts.

2.3 Errors

There are two types of errors that can be returned from a message you send. One type of error is an error in the message itself. For example, the format of the message is bad or the requested object/method doesn't exist. Another error is a problem with the data supplied to the command being executed.

For example, you could pass an improperly formatted message and an error would be returned. Or you could pass a properly formatted message, name a valid object and method – like Device.RockerPress - but the command itself doesn't work. An example of this would be that you supplied the name of a device and that device didn't exist in your design. On one hand the message "worked" in that it was properly encoded and got all the way to the object method but then the method failed.

In the first case, the message you receive back what is known as a "short error". This error is formatted as a string with one parameter. For example:

```
0012    -100
```

The errors that are returned in this manner are:

-100	Problem with decoding the string. Doesn't have 4 spaces in it. The preamble isn't composed of all numbers, etc.
-102	Less than 2 arguments in the string
-103	Invalid group. The 1st argument isn't HCAObject or HCAApp
-104	HCAObject group error - the object.method doesn't exist or the string doesn't contain a '.'. For HCAApp group, the command named doesn't exist.

-105	The number of arguments to the object method doesn't match the method, or the type of the arguments is incorrect. For example, data that must be a number can't be converted into to a number.

Really big note: It is very important to always check the error return values! You should check that the message you sent did receive a return code that isn't an error. That is, look for a short error return or the return value from the method being something other than 0. Always do this! On the Server side all data sent by you is checked for the correct data type and range of values. If a method accepts a parameter that can be 0 or 1 and you supply a number outside that range, the server detects it and returns an error. During development, these errors, if tested and displayed in your application can greatly shorten development time. The HCA Appendix on the object model discusses the error returns from the object methods.

3 HCAObject Messages

From this point on in this discussion, the data will not be presented in the encoded format showing the preamble and the packed arguments. This is just for ease of reading. All strings passed to the server in this protocol have to be packed into the format described above.

The first parameter in the messages is called the group. The group for most messages you may want to use is "HCAObject". For HCAObject messages, the next parameter is the name of an object and method as one string with the object and method separated by a dot. These are the exact same object and methods as exposed by the HCA object model and documented in the HCA User Guide Appendix.

Note: You must spell the object name and method exactly as it appears in the User Guide Appendix. Upper and lower case is important.

The remaining parameters are the arguments to the object method. For example:

```
HCAObject HCADevice.RockerPress Bedroom – Bath Lights 0 0 0
```

The RockerPress method takes four parameters: The name of the object and three additional parameters which are all numbers. That is what is supplied in this message.

Each command sent to the server generates a reply. This could be a short error if the command can't be processed because of the format, or if the message can be processed, then the server returns:

```
<returncode> <group> <command> <result>
```

The group and command are echoed from your message. The return code is from execution of the message. For example, the RockerPress command returns a number which is the return code from the

method. For RockerPress if the return code is zero the function worked. If non-zero it can be one of many possible errors. These are documented in the HCA Object Model User Guide Appendix.

An example:

```
0 HCAObject HCADevice.RockerPress 0
```

This is the return message reporting that the command worked. Note that the first parameter in the message is "0" which means the object method worked. The return value of the method is also given in the last part of the message.

An object method like HCADevice.Name takes a single argument – the number of the device in the enumeration – and returns the name of the device. The return message from this looks like:

```
0 HCAObject HCADevice.Name Bath - Lights
```

You know that the method worked because the first parameter in the message is "0". The result of the method is given in the last string in the message. In this case "Bath – Lights"

Note: Not all object and methods documented in the HCA User Guide Appendix are supported in this protocol. All the common ones are. If you receive an error (-104) and you are sure that the object method is spelled correctly, contact technical support.

Note: In the Object model there is a distinction between devices and controllers. In general, devices control a load – switches and modules – and controllers send transmissions. You need to use the Device object with devices and the Controller object to work with devices. What about devices that both send and receive? Are they controllers or devices? You just need to figure that out on a case by case basis.

4 HCAApp Messages

The other group you may want to use is HCAApp. These messages are much more complex but when used, they open up a much more detailed level of access to the HCA design. While you don't have to use this group, if you want to receive status updates for objects you must understand and work with this group.

A second advantage of using this group is that several of the HCAApp messages combine into a single message sent and a single message received what would take a number of HCAObject messages to be sent and received.

4.1 GetDesign

The GetDesign message provides complete access to the whole HCA design as a series of messages. Each message sent from the server to the requestor contains information on one or more objects in the design. To use this method, the client calls GetDesign with an increasing block number until GetDesign returns an error (-1) which indicates the end of the data.

To start the process the client sends:

```
HCAApp GetDesign 0
```

The server responds with:

```
<return code> HCAApp GetDesign <data>
```

After the first block of data is received and processed, the client should then send:

```
HCAApp GetDesign 1
```

And the client receives the next block of data. The client then sends the GetDesign message again – for block 2 – and continues this until an error is returned (return code is -1) which indicates the end of the data stream.

The server returns a message containing data for one or more objects. The data for each of those objects is represented by an array. Each array contains:

1. Object id. Number. Save this number if you want to work with object update messages.
2. Object name. String. Name of the object.
3. Icon name. String. The name of the icon. This is the same name as selected in the object properties and comes from the filename in the icon theme.
4. PopupName. String. Used by the client applications to determine how to show the action page for the object.
5. State. Number between 0 and 100. Then state of the device where 0 is off, 100 is on, and numbers in-between are for dim level.
6. Suspend. Number. 0 = not suspended, 1 = disabled or suspended, 2= suspended because of home mode.
7. Type. Number. 0 = Device, 1 = Program, 2 = group, 3 = Controller.
8. Rocker Count. Number. For some devices and controllers there are one or more transmit components. This tells you the number of rockers.
9. Button Count. Number. For some devices and controllers there are one or more transmit components. This tells you the number of buttons.
10. Rocker info if rocker count > 0, omitted if rocker count = 0. See below.
11. Button info if button count > 0, omitted if button count = 0. See below
12. Button state info. See below
13. Short tap action. Number
14. Long tap action. Number
15. Folder name. Text. Folder/Room where the object is stored.

16. Current Icon name. Text. If this is an empty string use the icon name given above
17. Current Icon label. Text. If this is an empty string use the object name
18. Current Icon representation. Number between 0 and 100. Where 0 is off, 100 is on, and in-between is dim. Only useful if Current Icon Name is not an empty string.
19. Wattage. Number. Wattage of the device as set on the "Power Track" device properties tab.
20. Object "no show" property. Number. In the properties of an object in HCA there is a "Don't show this on the control interface" checkbox. This array element is for that property.
21. Alexa name if Alexa support enabled for this device.

The rocker info, button info, and button state info are data supplying information about each rocker and about each button. If the object has no rockers or buttons then there are no additional elements in the array for this object – there are no parameters between "Button Count" and "Short tap action". Extract this information based upon the number of rockers and the number of buttons as:

```
for (iRocker = 0; iRocker < ctRockers; iRocker++)
{
       Extract from message rocker info
}

for (iButton = 0; iButton < ctButtons; iButton++)
{
       Extract from message button info
}

for (iButton = 0; iButton < ctButtons; iButton++)
{
       Extract from message button state info
}
```

The rocker info is a single string and is the name of the rocker. The button info is a single string which is the button name. The button state info is a number with a value of 0 or 1. If 0 the button is "off" and if 1 the button is "on".

How do you determine how many objects are contained in the message received from the server? That is determined from the number of parameters. There will be 'n' parameters for 'n' objects. Each of those parameters is an array as described above. You shouldn't assume that each message contains information on the same number of objects.

The short tap action and the long tap action are as the user configured them in HCA. It is up to the application to support these or not. The encoding is 0=Nothing, 1 = Toggle state, 2 = show popup. For programs you would use the start and stop methods to toggle. For devices you would use the on and off methods to toggle.

The icon shown for a device, program, group, controller, or folder shown on a display can become complex. When the object is first created or modified the user makes a choice of the icon. But that icon can be changed by a program using the *change icon* element. Thus each object has the user selected

icon – its representation, On, Off, or Dim determined by its state – and it has, if acted upon by the *Change Icon* element in a program, a "current" icon and a "current icon representation". Any application that uses the GetDesign should maintain the user selected icon and also the current icon. If the current icon is "" then the user selected icon should be used.

This is the same with the text below the icon. Unless acted upon by the *Change Icon* element in a program, the text below the icon is the object's name. Again, applications should maintain the "current icon text" and if that text is "" then the object name should be used.

The current icon name, current icon representation, and current icon text is first supplied by the Get Design results and is updated by the Update message.

It is up to the application to determine what information to save from Get Design. Again, if you want to receive object status updates you will have to save at least the object id as that id is used in the update messages.

For protocol "A" or "B" any object in the HCA design marked as "no show" is filtered out and never returned by Get-Design. In protocol "C" all objects are returned.

Note: The object id is not static across connections to the server. It shouldn't be saved in an applications persistent state as it may change upon the next connection.

4.2 GetDisplays

The GetDesign command, described above, retrieves information about each device, program, group, and controller. To retrieve information about displays, the GetDisplays command is used.

The GetDisplays message provides complete access to all the displays in the HCA design as a series of messages. Each message sent from the server to the requestor contains information on one or more displays in the design. To use this, the client calls GetDisplays with an increasing block number until GetDisplays returns an error (-1) which indicates the end of the data.

To start the process, the client sends:

```
HCAApp GetDisplays 0
```

The server responds with:

```
<return code> HCAApp GetDisplays <one or more arrays>
```

After the first block of data is received and processed, the client should then send:

```
HCAApp GetDisplays 1
```

And the client receives the next block of data. The client then sends the GetDisplays message again – for block 2 – and continues this until an error is returned (return code is -1) which indicates the end of the data stream.

Information about each display in the design – rooms, folders and user created displays - is returned by an array of information. You can determine the number of displays returned in the response message by the number of parameters in the returned message. Each parameter returned is an array or the form:

1. Display Name. Text. The name of the display
2. Object Id. Number. Used in update messages
3. Current state. Number. 0 = off, 100 = on.
4. Icon name. Text. The icon selected for this display in the TUI. It is the filename of the icon.
5. Short tap action. Number
6. Long tap action. Number
7. 0 | 1 : The state of the "do NOT show an icon for this display on the constructed TUI home page" option in the "Control Interface" tab for this display
8. 0 | 1: 0 = show object name, 1 = show two part name ("folder – name")
9. Current Icon name. Text. If this is an empty string use the icon name from the object state
10. Current icon label. Text. If this is an empty string use the display name
11. Display type. Number
    - 0 = icons
    - 1 = static HTML
    - 2 = Dynamic HTML
    - 3 = Graph
    - 4 = Text
    - 5 = Tiles
    - 6 = MRU
12. 0 | 1: If the display should label with two-part names. Note that this duplicates a setting given above. It is this way for legacy reasons.
13. An array of one or more items each of which is the id of an object shown on that display. Each of these items is a number and corresponds to an id of an element of the design retrieved from GetDesign.

14. Icon representation. Number between 0 and 100. Where 0 is off, 100 is on, and in-between is dim. Only useful if Current Icon Name is not an empty string.

15. Background type. Number. Always 0 for non icon displays
    - 0 = None
    - 1 = Image
    - 2 = Linked DXF
    - 3 = Imported DXF

16. Background path. String. Will be “” if background type is “None” or “Imported DXF”

17. No-Auto Close property. Number. 0=close by timer if set, 1 = never close by timer

18. Display label name. String. The label is as given in the display properties on the “Name” tab of its properties. The label may be “”. This is typically shown in the header of the page to identify the floor, room, or area.

19. The result path for a HTML display. Given as "" for non-HTML displays

20. The result URL for a HTML display. Given as "" for non-HTML displays

21. The name of the theme used by the display (New in V13)

22. "0" if this object is a folder, "1" if this object is a room, "2" is this object is a display. (New in V13)

23. The Alexa name if Alexa support is enabled for the room. (New in V13)

For the short and long tap actions the encoding is: 0 = nothing, 1 = toggle, 2 = show new page. For toggle you would use the room on and off methods. The “show a new page” value means to push the current page on the page stack and navigate to that page to show the icons on that page.

As explained in the Get Design section, displays also have a user selected icon and a current icon. This should be implemented in the same manner as explained above for devices.

In protocol ‘A’, all the icons on a display that are for other displays (folders, rooms, or displays) are filtered out and not returned. For protocol ‘B’ all displays whose icons appear on the display are in the list of icons on that display. Rooms can be on or off and the current state shows that. The update message keeps that status up to date.

To determine the number of displays returned in the message from the server you use the number of parameters in the preamble. To determine the number of elements in the array for each design – it will be different for each display – use the preamble in the array.

4.3 SetClientOptions

The SetClientOptions configures the server for this client connection. This message is of the form:

```
HCAApp SetClientOptions <number> <clientName>
```

The returned message is:

```
<return code> HCAApp SetClientOptions <number>
```

The number in the message sent by the client is a bitmap of options. Set the bit to enable the option and clear the bit to display the option: The bitmap encoding is:

Bitmap value	Description
0x0001	Send to the client object state changes
0x0002	Send to the client notify messages
0x0004	Send to the client server status messages
0x0008	Send to the client tile update messages
0x0010	Send to the client User UI requests (dialogs requested by the User-UI program element)
0x0020	Send to the client sound requests (Programmer elements Speak and Play-Sound)
0x0040	Send to the client complete ExtServerStatus messages. With schedule info.
0x0200	Send to the client log updates
0x2000	Send to the client shorter ExtServerStatus messages. Without schedule info.

Figure 5: SetClientOptions Bitmap values

The number returned in the message from the server is the client options prior to changing them with the values in the client message. In this way a client can set the options and then restore them to what they were before the set. Options are maintained as part of the state for each connected client. So what you set for your client doesn't affect other clients.

The default at the time of client connection is that clients do not receive any update messages.

The client name – an optional parameter – is whatever text string desired to identify this client connection.

Note: Receiving update messages can make an application much more complex. This is because the notification and update messages are received asynchronously. Except for these types of messages the client is fully in control. The client sends a message and the server responds. The server never initiates a conversation. When notifications or updates are enabled, then the server, without being prompted by the client, can send a message to the client. There is no client reply to these messages but the client must always be listening for them. This usually requires a multi-tasking solution which will increase the complexity of the client.

4.4 GetHomeModeNames

This method returns the name of the home modes for the current design. The message sent is:

HCAApp GetHomeModeNames

The return message is:

```
<return code> HCAApp GetHomeModeNames <name1> <name2> <name3> <name4>
```

NOTE: The number of returned parameters depends upon the number of home modes defined by the user. It could be 1, 2, 3, or 4. The mode names are all text.

4.5 GetHomeMode

This method returns the current home mode. The message sent is:

HCAApp GetHomeMode

The return message is:

```
<return code> HCAApp GetHomeMode <mode number>
```

4.6 SetHomeMode

The message sent is:

HCAApp SetHomeMode <mode number 1 - 4>

The return message is:

```
<return code> HCAApp SetHomeMode <mode number before set>
```

4.7 ThermostatState

The message sent is:

```
HCAApp ThermostatState <name>
```

Where the name is a two-part name that must reference a thermostat.

The return message is:

```
<return code> HCAApp ThermostatState name <data>
```

The return code is 0 if the message was processed successfully and -1 if not. An error return means that the name supplied isn't found in the design or doesn't reference a thermostat. The data consists of 10 numbers – not as an array, just as 10 numbers:

1. Heat setpoint current
2. Heat setpoint range high
3. Heat setpoint range low
4. Cool setpoint current
5. Cool setpoint range high
6. Cool setpoint range low
7. Current temp
8. Current mode
9. Current Fan
10. String: "F" or "C" – temperate units
11. Humidity if supported by the unit (new in V13)
12. Error Text. If the return code from the method is non-zero then this message is suitable for displaying to the user. (new in V13)

The encoding of these numbers is in the same manner as the HCA device object thermostat method and is documented in the HCA User Guide Appendix on the Object Model.

The advantage of this function over using the HCAObject methods is that, in one client to server transaction all the various settings and values are returned. Note that since this involves communication with the thermostat – to retrieve, for example, the current room temperature, it may take some additional time to process.

Some units only change the setpoints depending upon the mode and the thermostat type. If the value for a setpoint has a value of -1, don't show that value and the user should be prevented from adjusting it.

If the user changes the mode, all the data should be retrieved again. This way the client can see what setpoint data was returned (-1 or some real value) and adjust the UI to show what can be modified or

not. This way a client doesn't need to know about how the thermostats work with the modes. Changing modes isn't common so it can be a bit slow.

The humidity is either reported as -1 or a value. If you get a value then when you show the temperature then also show the humidity. Otherwise don't show it.

The last piece of info – the error text - is important.

The nest thermostat and others are great error reporters and have many conditions hard for the server and clients to classify. Please check the error return from the function and if non-zero then report an error and if you get something besides a null string you can report that as part of the message. It is very important to do this as there is so much between HCA and the thermostat now.

Just to be clear, you will get back all the values even if the method fails – they will just be all -1 or some such – and you will ignore all but the last – errorText.

4.8 ThermostatChange

The message sent is:

```
HCAApp ThermostatChange <name> <code> <value>
```

Where the name is a two-part name that must reference a thermostat. The code is the setting to change and the value to change it to. The codes are:

- 1 = Heat setpoint
- 2 = Mode
- 3 = Fan
- 4 = Economy
- 7 = Cool setpoint

The return message is:

```
<return code> HCAApp ThermostatChange name <code> <value> <error text>
```

If the return code is non-zero then the error text contains a message suitable for displaying to the user.

4.9 IRKeypad

The IRKeypad message retrieves all the information needed by a client to render an image of an IR keypad.

The message sent is:

```
HCAApp IRKeypad <name>
```

Where the name supplied is the two-part name that must reference an IR device.

The message returned is:

```
<return code> HCAApp IRKeypad <data>
```

Where <data> is one or more arrays. Each array represents the information for one IR keypad button. You can determine the number of buttons on the keypad by the number of parameters returned. Each IR button is an array of 7 items. These are:

- Button name. Text. The name of the button
- Button Label. Text. The text that should appear on the button when displayed to the user
- X. Number. The X location of the button.
- Y. Number. The Y location of the button.
- Width. Number. The width of the button.
- Height. Number. The height of the button.
- Type. Number. 0 = a normal button. 1= text on the keypad – a non functional button.

The X, Y, width, and Height are in pixels and may be need to be scaled for your application.

4.10 Ping

The Ping command is a short message sent from the client to the server to keep communications open.

The format of the message is:

```
HCAApp Ping <number>
```

Where:

- Number. The number of minutes that the periodic Ping takes place. This gives the server the capability to watch for client disconnections that it wouldn't normally see. If the next Ping doesn't show up after 'n' minutes, the server could consider the client "gone away".

The server responds with:

```
<return code> HCAApp Ping <number>
```

4.11 Terminate

The Terminate command is a short message sent from the client to the server to say that the client is terminating. This is an optional message as the server disconnects the client if it gets notification of the socket closing.

The format of the message is:

```
HCAApp Terminate
```

4.12 GetDisplayText

To retrieve the text shown in a text display.

The format of the message is:

```
HCAApp GetDisplayText <displayName>
```

The message returned is:

```
<return code> HCAApp GetDisplayText <text>
```

4.13 GetDisplayMRU

To retrieve the list of ids on a MRU Display.

The format of the message is:

```
HCAApp GetDisplayMRU <displayName>
```

The message returned is:

```
<return code> HCAApp GetDisplayMRU <ids>
```

Where <ids> are zero or more ids. One for each object that has an icon on this display. The number of icons is the number of returned parameters minus 3 (for the return code, “HCAApp” and the method name).

4.14 GetDisplayHTML

This method is used to return both the URL – in the case of a static HTML display or the HTML file if a dynamic HTML display.

The format of the message is:

```
HCAApp GetDisplayHTML <displayName> <block#>
```

The result depends upon if this is for a static or dynamic HTML display.

The message returned for a static HTML display is:

```
<return code> HCAApp GetDisplayHTML <URL text>
```

For dynamic HTML – a HTML file generated by HCA from a template – the GetDisplayHTML method returns the file as a series of messages. Each message contains one block of the file. To use this, the client calls GetDisplayHTML with an increasing block number until GetDisplayHTML returns an error (-1) which indicates the end of the data.

To start the process, the client sends:

```
HCAApp GetDisplayHTML <displayName> 0
```

The server responds with:

```
<return code> HCAApp GetDisplayHTML <block #> <text>
```

After the first block of data is received and processed, the client should then send:

```
HCAApp GetDisplayHTML <displayName> 1
```

And the client receives the next block of text. The client then sends the GetDisplayHTML message again – for block 2 – and continues this until an error (-1) is returned which indicates the end of the data stream.

It is assumed that the client saves this data in a file and then points a browser at it.

4.15 GetDisplayGraph

To retrieve the data necessary to construct a graph similar to the one shown by a HCA graph display.

The format of the message is:

```
HCAApp GetDisplayGraph <spec#> <displayName>
```

The results depends upon if this is a current power graph or a historical power graph. Currently the <spec #> must be 0.

For a current power graph the message returned is

```
0 HCAApp GetDisplayGraph <title> <#bars> <max value> <arrays>
```

There are ‘n’ arrays where ‘n’ is the number of bars.

The “max value” is the largest value for any bar. This can be used to set the Y axis scale. It is important to handle the case where this value is zero. This will happen for a current power graph and no devices are on. In this case scale the Y axis to some default.

The values for the current power graph are in watts. For the Historical Power Graph they are watt/hours.

Each array contains these elements:

1. Bar Label (text)
2. Id of the device or folder for this bar (number).
3. Bar value (number: Watts)
4. Bar color RGB red (number)
5. Bar color RGB green (number)
6. Bar color RGB blue (number)

For a historical graph the data returned is more complex since it could be rendered as a stacked bar graph.

There are still ‘n’ arrays where ‘n’ is the number of bars but each array has a variable number of elements:

1. Bar Label (text)
2. Total Bar value (number: watt/hours)
3. Section id (number)
4. Section value (number: watt/hours)
5. Section color RGB red (number)
6. Section color RGB blue (number)
7. Section color RGB green (number)

3 to 7 is repeated for each section of the bar. The number of elements in the array is:
2 + (count of bar sections * 5)

If the graph isn’t shown as a stacked bar, the total bar value can be used and the section data discarded.

The “section id” is the id of the device that contributed this section of the bar.

4.16 GetFile

The GetFile method retrieves the contents of a file from the server. The data is returned in “chunks” so more than one call to this method may be necessary to retrieve the complete file.

To start the process, the client sends:

```
HCAApp GetFile <retrieve type> <name> 0 [filesize] [filechecksum]
```

The server responds with:

```
<return code> HCAApp GetFile <retrieve type> <name> 0 <file type> <data>
```

After the first block of data is received and processed, the client should then send:

```
HCAApp GetFile <retrieve type> <name> 1
```

And the client receives the next block of data. The client then sends the GetFile message again – for block 2 – and continues this until an error is returned which indicates the end of the data stream.

The <data> is a single string. The string represents binary data rendered into text form as hex characters. This has to be converted into binary data before being written to the file. The "file type" in the returned data, given as a string (for example "jpg" or "png"), is the type of file.

What is being retrieved is supplied by two parameters, <retrieve type> and <name>. Both are strings.

Retrieve Type	
IconOff	Off representation of an icon. The name argument is the icon name.
IconOn	On Representation of an icon. The name argument is the icon name.
IconDim	Dim Representation of an icon. The name argument is the icon name.
Display	Background Image of a display. The name argument is the display name.
File	Any file on the server. The name argument is the path to the file. **There are two optional arguments**: the size and checksum of the file on the client. If these match the file on the server then the retrieve need not be done and a status return code tells this. The checksum is computed as a 32 bit sum of all bytes in the file. If the file doesn't exist on the server then set these both to zeros.

Error Return values are:

-1	All file data retrieved
-2	No such name
-3	No such retrieve type

-4	File size and checksum match

4.17 TimeStamp

The TimeStamp method returns the time when the design was last updated and the time when any object last changed state. The time is based upon the clock on the computer running the server and is not corrected for client local time.

The client sends:

```
HCAApp TimeStamp
```

The server responds with:

```
<return code> HCAApp TimeStamp <design change time> <state change time>
```

Each time is a string of the form yyyymmddhhmmss

4.18 RefreshState

The RefreshState method requests that an Update message be sent for all objects in the design – devices, programs, groups, controllers, and rooms – that have changed state since the time given.

The client sends:

```
HCAApp RefreshState timestamp
```

The server responds with:

```
<return code> HCAApp RefreshState
```

The timestamp must represent a valid time or you get an argument error. As long as you supply a valid time, the return code is always zero. After receipt of the response the client should expect a flood of Update messages.

The TimeStamp and Refresh methods give you a way to handle brief disconnects. If you are disconnected you can totally reload the design or you can request the timestamp and see if the timestamp matches what you have. There are two parts to the timestamp. When the design as last modified and when the last state update happened.

You don't need to understand the data in the timestamp to do this. After you start up and have processed GetDesign and GetDisplays you should invoke the TimeStamp method and save the result. Each Update messages contains a timestamp and you should use that to keep your "state change time" timestamp up to date.

If you disconnect and then reconnect you can see if the design timestamp matches and if it does you don't have to reload. Also if the state timestamp matches then you are done. If it doesn't match then you can use the *RefreshState* method supplying the time you last received in an update message. The server then sends on any updates since then.

Again, you don't have to understand the values in the timestamps. Just save them and pass them to RefreshState as needed.

4.19 Get Inspector Report

The GetInspectorReport method returns a list of all the messages displayed in the HCA troubleshooter that are not "checked off" by the user.

The client sends:

```
HCAApp GetInspectorReport
```

The server responds with:

```
<return code> HCAApp GetInspectorReport <message array>
```

Where the message array contains 3 elements for each message:

- Message level. Number, 0=green, 1=yellow, 2=red
- Message text. String.
- Time message created. DateTime. Formatted as YYYYMMDDHHMMSS

Note: There may be zero messages to return in which case the server response doesn't contain the message array item.

4.20 Get Alert Report

The GetAlertReport method returns a list of all the messages displayed in the HCA alert viewer.

The client sends:

```
HCAApp GetAlertReport
```

The server responds with:

```
<return code> HCAApp GetAlertReport <ctAlerts><AlertColor><message array>
```

The ctAlerts and AlertColor are the count and color shown in the HCA ribbon with the "Alert" button.

"Message Array" contains 2 elements for each message:

- Message text. String.
- Time message created. DateTime. Formatted as YYYYMMDDHHMMSS

Note: There may be zero messages to return in which case the server response doesn't contain the message array item.

4.21 FormatText

The FormatText method processes text with replacement sections – enclosed in "%" characters into text with the replacement sections replaced by the computed results.

The client sends:

```
HCAApp FormatText <text with replacement sections>
```

The server responds with:

```
<return code> HCAApp FormatText <resultant text>
```

4.22 GetDisplayTiles

The GetDisplayTiles method returns information about the tiles on a tiled display.

The client sends:

```
HCAApp GetDisplayTiles <displayName>
```

The server responds with:

```
<return code> HCAApp GetDisplayTiles <tileArray>
```

Where:

- DisplayName. String. The name of the display.

The TileArray contains information about each tile. Each array element describes one tile. The data returned for each tile is different depending upon the tile type but the first items in the array are the same for each tile type.

- Tile Type.
    - 0 = Blank Tile

    - 1 = HTML Tile
    - 2 = Icons Tile
    - 3 = Image Tile
    - 4 = Power Graph Tile
    - 5 = Text Tile
    - 6 = Power Meter tile
    - 7 = Alerts Tile
    - 8 = Status Tile

- Tile Id. Number.
- Name. String
- Label. String
- X. Number. X, Y, Width, and Height are in the same units as specified in the Tile Properties in HCA. That is, ¼ of the large icon theme cell size.
- Y. Number
- Width. Number
- Height Number
- Stretch Option. Number. 0 = no stretch, 1 = Horizontal, 2 = Vertical, 3 = Both directions
- Tile Color. Number. RGB value of tile color.
- Tile text Color. Number. RGB value of tile text color.
- Short Tap Action. Number. 0 = Nothing, 1 = Show Display, 2 = run program / Toggle load / Toggle room
- Short tap display id. Number.
- Short tap program id. Number.
- Long Tap Action. Number. 0 = Nothing, 1 = Show Display, 2 = run program / Toggle load / Toggle room
- Long tap display id. Number.
- Long tap program id. Number.
- Refresh Time. Number. Time in seconds on how often to update the tile. 0 = no auto refresh

The next data items depend upon the type of the tile.

Blank Tile
No additional info

HTML Tile

- Source option. Number. 0 = HTML from display, 1 = HTML from URL
- Display id. Number. Id of HTML display
- URL. Text

Icons Tile

- Override theme spacing. Number. 0 = No, 1 = Yes
- Spacing X direction. Number.
- Spacing Y direction. Number.
- MRU. Number. 0 = Not a MRU tile, 1 = MRU Tile

- Show devices in MRU? Number. 0 = No, 1 = Yes.
- Show programs in MRU? Number. 0 = No, 1 = Yes.
- Show groups in MRU? Number. 0 = No, 1 = Yes.
- MRU Count. Number. Number of icons to show in the display.
- Display Id. Number. Id of the display if not a MRU tile.

Image Tile
- Path. String. Path to the image file.
- Scale. Number. Scale to fit? 0 = No, 1 = Yes.

Graph Tile
- Id. Number. Graph display id

Text Tile
- Text. String. The text to display.
- ScaleToFit. Number. 0=No, 1 = Yes. Should the point size be expanded so the text fills the tile?

After these 2 items, 14 more follow. These are the elements of the Windows LOGFONT structure.

Power Meter Tile
- Max Wattage. Number. The max wattage of the meter
- Max Wattage Override. Number. 0=No, 1=Yes. If the override then use the specified max wattage. If not compute the max wattage from the devices in the design.

AlertTile
No extra data

StatusTile
No extra data

4.23 GetScheduleNames

The GetScheduleNames method returns the names and ids of the schedules in the loaded design.

The client sends:

```
HCAApp GetScheduleNames
```

The server responds with:

```
<return code> HCAApp GetScheduleNames <name1><id1><name2><id2>…
```

Where:

- Name. String. Schedule name

- ID. Number. The id of the schedule. This needs to be saved and used if the SetCurrentSchedule method is used.

Note: When processing the reply, make sure you handle the case where the design has no schedules.

4.24 SetCurrentSchedule

The SetCurrentSchedule method changes the current schedule on the server.

The client sends:

```
HCAApp SetCurrentScheudle <id>
```

The server responds with:

```
<return code> HCAApp SetCurrentSchedule <id>
```

Where:

- Id. Number. The id of the schedule to make current.

4.25 GetServerStatus

The GetServerStatus method returns the server status. This is similar to the ServerStatus update message.

The client sends:

```
HCAApp GetServerStatus
```

The server responds with:

```
<return code> HCAApp GetServerStatus
<lights><ctRed><ctYellow><ctAlerts><mode><scheduleId><now><designLoadTime>
<sunriseTime><sunsetTime>
```

Where:

- Lights. Number. 0=Unknown, 1=Green, 2=Yellow, 3=Red. Overall status level of the HCA Server.
- ctRed. Number. Count of red level messages.
- ctYellow. Number. Count of yellow level messages.
- ctAlerts. Number. Count of alerts

- Mode. Number. Current home mode. This is the index into the home mode names returned by GetHomeModes.
- ScheduleId. Current schedule. This is the id of the current schedule. The id of each schedule is retrieved by GetScheduleNames.
- Now. DateTime. Current time on the server.
- DesignLoadTime. DateTime. Time when the server loaded the current design.
- SunriseTime. DateTime. Time of sunrise on the server.
- SunsetTime. DateTime. Time of sunset on the server.

4.26 GetExtServerStatus

The message sent is:

```
HCAApp GetExtServerStatus
```

The server responds with:

```
HCAApp GetExtServerStatus <results>
```

The results are a variable number of parameters consisting of:

- Time on the server
- Sunrise time on the server
- Sunset time on the server
- Running time of the server
- Time display – the “Today Is” text shown in the HCA status dialog
- Date display – the season info shown in the HCA status dialog
- Zero or more schedule entry images. These are the next ‘n’ schedule entries to be executed by the server. You can determine the number of schedule entries returned by the number of returned arguments minus 6.

4.27 GetThemeIcons

The GetThemeIcons method gives the client a way to enumerate the icons in a given theme and return information about the images in that theme.

The client sends:

```
HCAApp GetThemeIcons <themename> <state>
```

On the first call pass 0 as the state. Capture the value returned from the server and pass it to subsequent calls to GetThemeIcons until the method returns an error. Each return provides into about one or more image files. The number returned for each call can vary.

The server responds:

```
HCAApp GetThemeIcons <themename> <state> <image info>
```

For each image file this info is returned: The filename, the file size in bytes, and the file checksum. The file size and file checksum is there so that a client can see if that image file exists in the local store. Check for the file existence, check the file size, and then get the checksum. The file size may make getting the checksum moot. The checksum is just the sum of all bytes in the file into a 32 bit unsigned quantity.

The important point is that the filename of the image file is returned not just the icon name that HCA shows. HCA accepts images in PNG, JPG, and other image file formats. The filename returned includes the file type. For example, “Table Lamp_On.png”

4.28 GetThemeIcon

The GetThemeIcon method gives the client a way to retrieve the image file from a specified theme.

The client sends:

```
HCAApp GetThemeIcon <themename> <filename> <block>
```

The server responds with:

```
HCAApp GetThemeIcon <themename> <filename> <block> <hex as text>
```

On the first call, set block to 0. On each subsequent call increment block by one. Continue calling until the method returns an error.

The contents of the file are returned as a stream of hex represented as text. The caller must convert the text into binary and save to the image file. Thus the total number of characters returned when all blocks are read is twice the number of bytes in the image file.

4.29 GetLogFilters

The GetLogFilters method returns the names of all named log filters in the loaded design. These can be used with the GetLog method to filter the log during retrieval.

The client sends:

```
HCAApp GetLogFilters
```

The server responds with:

```
<return code> HCAApp GetLogFilters <filter array><filter array>
```

Where each filter array contains three elements:

- Name. String. Filter name
- Sort field. Number. 0-5
- Sort direction. Number. 0-1 where 0 is Ascending, 1 is descending

The encoding of the sort field is:

0: Entry type
1: Date Time
2: Interface
3: Command
4: Address
5: Name
6: Info

Note: When processing the reply, make sure you handle the case where the design has no filters. If the application supports letting a user select a filter then the application must reload the log from the server as the application doesn't have the necessary info about what a filter does to apply the filter to the log.

4.30 GetLog

The GetLog method returns can be used to access one of the server logs. Since the log is large this method returns a block of data and is then called repeatedly to retrieve the whole log.

The client sends:

```
HCAApp GetLog <logId> <filtername> <state>
```

Where:

- LogId. Number 0 – 2. In HCA there are three logs.
- Filter name: String. The name of the filter to use. An empty string "" retrieves the whole log

- State. Number. On the first call set to zero. On subsequent calls pass the value returned in the last response.

The server responds with:

```
<return code> HCAApp GetLog <filteredCount><totalCount><state><entry1><entry2>…
```

Where:

- Filtered Count. Number. This is the number of entries in the log with the filter applied if one was specified. Note that this value is specified only for the response to the GetLog call with the state = 0. For other GetLog responses the value is zero.
- Total Count. Number. This is the number of entries in the log with no filter applied. Note that this value is specified only for the response to the GetLog call with the state = 0. For other GetLog responses the value is zero.
- State. Number. The value to be passed in to the next call to GetLog.
- Entry'n': String. The log entry as a comma separated string. You can determine the number of log entries in the block from the number of arguments in the response.

A log entry is represented by 8 fields in the CSV string. These are:

1. Entry type. Number. There are 32 log entries types.
2. Entry Name. String. This is the name of the entry type
3. Date Time. String. Presented in a displayable format
4. Interface. String. The interface hardware associated with this log entry if there is one
5. Cmd. String. The command (ON, OFF, etc)
6. Address. String. The address where the command was sent to or received from.
7. Name. String. The name of the object associated with this log entry
8. Info. String. Any information.

The log Entry types are as given by this enumeration.

```
typedef enum
 {
  LogEntryNote = 0,
  LogEntryX10Send,
  LogEntryX10Receive,
  LogEntryProgram,
  LogEntryError,
```

```
  LogEntryX10ReceiveNotDesign,
  LogEntryUnused,
  LogEntryStartStop,
  LogEntryProgramError,
  LogEntryMMSend,
  LogEntryMMReceive,
  LogEntrySecurityReceive,
  LogEntryUPBSend,
  LogEntryUPBReceive,
  LogEntryUPBReceiveNotDesign,
  LogEntryIRSend,
  LogEntryInsteonSend,
  LogEntryInsteonReceive,
  LogEntryInsteonReceiveNotDesign,
  LogEntryInsteonError,
  LogEntryGCSend,
  LogEntryGCReceive,
  LogEntryUPBError,
  LogEntryZWaveSend,
  LogEntryZWaveReceive,
  LogEntrySerialSend,
  LogEntrySerialReceive,
  LogEntryRoomOn,
  LogEntryRoomOff,
  LogEntryAutoOff,
  LogEntryX10Error,
  LogEntryGCError
} LogEntryType;
```

5 Update Messages

The server can send to the client messages asynchronously. That is, messages not in response to a request. The server will not send these messages if not requested to by the client – see the SetClientOptions method. This section describes those messages.

5.1 Update message

The update message is used by the server to inform the client of a state change of an object in the design. The client must not respond to this broadcast message it just processes it.

The format of the message is:

0 HCAApp Update <id> <state> <suspend> <ctButtons> <button state> <icon name> <text below the icon> <Icon representation><timestamp><wattage>

Where:

1. Id. Number. The id of the object for the state change. This is the same id supplied in the GetDesign response.

2. State. Number between 0 and 100. Where 0 is off, 100 is on, and in-between is dim.
3. Suspend. Number. 0 = not suspended, 1 = disabled or suspended, 2 = suspended due to home mode.
4. ctButtons. Number. The number of button transmit components for this object. May be zero.
5. buttonState. One or more numbers. If ctButtons > 0, supplies the state of each button. Where 0 is off, and 1 is on.
6. Icon name. Text. If this is an empty string use the icon you saved during GetDesign / GetDisplays
7. Icon label. Text. If this is an empty string use the object or display name.
8. Icon representation. Number between 0 and 100. Where 0 is off, 100 is on, and in-between is dim. Only useful if Icon Name is not an empty string.
9. Server timestamp. String of the form yyyymmddhhmmss. This is the time on the server when the message was generated.
10. Wattage. Number. Wattage of the device. Zero if not a device.

Note: You will not receive update messages unless the SetClientOptions bitmap contains 0x0001

5.2 Notify message

The Notify message is used by the server to information the client of a notification. This message isn't fully developed in this version. It is used, for now, to communicate alerts from the alert manager to clients. Subsequent versions of this interface may expand its use.

The format of the message is:

```
0 HCAApp Notify <action> <id> <level> <text>
```

Where:

- Action. Number. For now, always a 1.
- Id. Number. The object id that this notice applies to. May be zero if the notification doesn't apply to an object.
- Level. Number. 0 = Green, 1 = Yellow, 2 = Red.
- Text. String. The text associated with the alert.

Note: You will not receive this message unless the SetClientOptions bitmap contains 0x0002

5.3 UserDialog message

This message is sent from the server to the client when the server executes a Request-Input Visual Program element. The client, upon receiving this message displays the type of dialog requested, waits for input and then returns the result using the UserDialogReport message.

The format of the UserDialog message is:

```
0 HCAApp UserDialog <dialogType><headerText><optionText*4><numberText*12>
<doTimeout><timeoutSeconds><killTimerOnUserInput><showOSK><resultPreload><Res
ultId>
```

Where:

- DialogType. Number.
    - 0: Numeric Dialog
    - 1: Text dialog
    - 2: Time dialog
    - 3: Option dialog
    - 4: Security complete dialog
- HeaderText, String. Text at the dialog top.
- OptionText. String. Four strings for the options used in the option dialog.
- NumberText. String. Twelve strings for the text on the buttons of the Security Complete dialog.
- DoTimeout. Number. 0 = No timeout, 1 = Timeout.
- TimeoutSeconds. Number. If "DoTimeout" is true then this is the length of time in seconds before the dialog auto closes.
- KillTimerOnUserInput. Number. 0=No, 1 = Yes. If yes, the timeout stops after the 1st user input of text or button push.
- ShowOSK. Number. 0-No, 1=Yes. Show the Windows On-Screen-Keyboard?
- ResultPreload. String or Number. If the *time* dialog is being shown then this supplies the time to preload the dialog result with. This allows the user to see the "old" value before they enter the new. If the *option* dialog is being shown then this is the initial choice selected. May be "" in which case there is no preload.
- ResultId. Number. Saved and returned as part of the UserDialogReport message.

Note: You will not receive this message unless the SetClientOptions bitmap contains 0x0010

5.4 UserDialogReport message

Unlike the other messages in this section, this message is sent **from the client to the server** as part of processing a request for user input started by the UserDialog message.

The format of the UserDialogReport message is:

```
0 HCAApp UserDialogReport <code><rcDialog><ResultId><ResultType><Result>
```

Where:

- Code. Number. 0=First user action,1=User closed dialog
- rcDialog. Number. 1= closed the dialog with OK, 2=closed the dialog with Cancel
- ResultId. Number. The same value as was sent by the server in the UserDialog message.

- ResultType. Number. 1=Number (numeric dialog, choice dialog), 2=time (time dialog), 3=text (text dialog, security complete dialog)
- Result. String.

There are two uses of the UserDialogReport message. The first – code 0 – is sent by the client to the server when the user does anything with the dialog. For example, entry of the first digit of a number. This allows the server to cancel its timeout in support of the "cancel timeout on user action" setting in the Request-Input visual programmer element.

The second use is to return the result back to the server when the dialog is closed or it times out. The reDialog tells if the dialog was closed by OK or by Cancel - a timeout results in a Cancel. The id is the result id passed to the client in the UserDialog message. The result type and result arguments pass to the server the data entered by the user into the dialog.

5.5 TileUpdate message

This message is sent from the server to the client if the client has requested (SetClientOptions) tile update messages. This message is created when a Visual Program Update Tile element is executed. The messages contents are identical to the properties of that element.

The format of the message is:

```
0 HCAApp TileUpdate
<tilename><changeLabel><label><changeImagePath><path><ChangeText><text>
<changecolor><back color><text color><refresh>
```

Where:

- TileName. String. The name of the tile to update.
- ChangeLabel. Number. 0 = No, 1 = Yes.
- Label text. The tile label.
- ChangeImagePath. Number. 0 = No, 1 = Yes.
- Image file path. The path to the file shown in an image tile.
- ChangeText. Number. 0 = No, 1 = Yes.
- Text. String. The text displayed in a text tile.
- Change Tile colors. Number. 0 = No, 1 = Yes.

- Background color. Number as a RGB value
- Text color. Number as a RGB value
- Refresh. Number. 0 = No, 1 = Yes. Should the tile be refreshed after the update?

Note: You will not receive this message unless the SetClientOptions bitmap contains 0x0008

5.6 PlaySound message

This message is sent from the server to the client when the Play-Sound visual program element is executed. A client, though the user settings, can choose to make sounds or not.

The format of the message is:

```
0 HCAApp PlaySound <path>
```

Where:

- Path. String. The path supplied by the user in the PlaySound element. The server may have to request this file from the server before playing it.

Note: You will not receive this message unless the SetClientOptions bitmap contains 0x0020

5.7 TextToSpeech message

This message is sent from the server to the client when the Visual Programmer Speak element is executed. A client, though the user settings, can choose to make sounds or not.

The format of the message is:

```
0 HCAApp TextToSpeech <op><text><voice><name><rate><volume><priority>
```

Where:

- Op. Number. 0 = speak now. 1 = kill current speak. 2 = kill current speak and empty queue.
- Text. String. The text to speak.
- Voice. String.
- Name. String.
- Rate, Volume, Priority: Number.

The Voice, Name, Rate, Volume, and Priority values are used to control the SAPI Windows facility. Refer to the example client source to see how these are used.

Note: You will not receive this message unless the SetClientOptions bitmap contains 0x0020

5.8 ServerStatus message

This message is sent from the server to the client if the status of the server changes. The format of the message is:

```
0 HCAApp ServerStatus <lights><ctRed><ctYellow><ctAlerts><mode><scheduleId>
```

Where:

- Lights. Number. 0=Unknown, 1=Green, 2=Yellow, 3=Red. Overall status level of the HCA Server.
- ctRed. Number. Count of red level messages.
- ctYellow. Number. Count of yellow level messages.
- ctAlerts. Number. Count of alerts
- Mode. Number. Current home mode. This is the index into the home mode names returned by GetHomeModes.
- ScheduleId. Current schedule. This is the id of the current schedule. The id of each schedule is retrieved by GetScheduleNames.

Note: You will not receive this message unless the SetClientOptions bitmap contains 0x0004

5.9 ExtServerStatus message

This message is sent from the server to the client if the status of the server changes. The format of the message is:

```
0 HCAApp ExtServerStatus <results>
```

The results are a variable number of parameters consisting of:

- Time on the server
- Sunrise time on the server
- Sunset time on the server

- Running time of the server
- Time display – the “Today Is” text shown in the HCA status dialog
- Date display – the season info shown in the HCA status dialog
- Zero or more schedule entry images. These are the next ‘n’ schedule entries to be executed by the server. The number of echedule entries returned is the number of arguments sent minus 6.

These messages will not be sent to the client unless they are requested in the SetClientOptions. The schedule info is only sent if the full extended status is requested. If the simpler form is requested then the schedule info is omitted.

5.10 LogAdd message

This message is sent from the server to the client when a new log entry is generated. The format of the message is:

```
0 HCAApp LogAdd <logId><LogEntryCSV>
```

Where:

- LogId. Number 0-3. Which log this entry goes to
- LogEntryCSV. String. Log entry encoded in the same format as is used with GetLog

Note: You will not receive this message unless the SetClientOptions bitmap contains 0x0200

There is two ways to handle the log. An application can enable this notification only when the log is on the display. The log is retrieved when the log display is opened and then updates keep it up to date.

Or the application could keep a local log and keep it updated with these update messages as long as the application is connected.

This is an application decision.

5.11 DisplayChange message

This message is sent from the server to the client when a program executes containing the “Show display” element. The format is:

```
0 HCAApp DisplayChange <id><op><autoReturn?><return time>
```

Where:

- Id. Number. The id of the display to show
- Op. Number. The op is as given below.

- AutoReturn. Number. 1 or 0. If after a set time the display should return to the display in effect before the DisplayChange message arrived
- Return Time: Number. Seconds. How long to wait after the display change before returning to the previous display.

The code argument is:

- 0 = show display. Display id as provided.
- 1 = Return to the Home page
- 2 = Perform a "Back" operation.

Note: You will not receive this message unless the SetClientOptions bitmap contains 0x0008

5.12 TextDisplayChange message

This message is sent from the server to the client when a program executes containing the "Show message" element. The format is:

```
0 HCAApp DisplayChange <id><text><back color><text color><make current><display time>
```

Where:

- Id. Number. The id of the display
- Text. String. The text to show in the display.
- Back Color. Number. The RGB value of the background color
- Text Color. Number. The RGB value of the text color
- Make Current. Number. 0 or 1. If the display should become the current display
- Display Time. Number. Number of seconds until the display reverts to the display that was current before the TextDisplayChange message arrived.

Note: You will not receive this message unless the SetClientOptions bitmap contains 0x0001

6 Getting more info

As stated above, this protocol is the same protocol use by *HCA for Android*. If you have an Android device you can listen in on the conversation by opening the Remote Access Viewer from the server menu. Select from the server application menu *Tools – Remote Access Viewer*.

The message sent to the server and its response appears in the viewer as long as it is open.

Last big note: The HCA client and server also communicate using the same IP port used by HCA for Android and any other applications created. The HCA to HCA communication is done in a much different protocol. That protocol is binary and internal. It will not be documented. If you use the Remote Access Viewer and have HCA clients connected you may see elements of this protocol decoded but look away as it will be of no use to you.

7 Web Sockets

Messages in the above protocol can also be sent into the HCA server over a web socket connection. For example:

```
var ws = new WebSocket("ws://192.168.2.8:2000/websocket");
ws.onopen = function()
{
   // Web Socket is connected, send data using send()
   ws.send("HCA000B011002001");
   ws.send("002500340044    HCAObjectDevice.OnDen - Lamp");
   alert("Message sent");
};
```

The same port number is used for all HCA clients. The initial web-socket protocol message tells the server that this is a web socket connection and data sent back and forth should be framed according to the web socket specification. The web socket protocol implemented by the HCA Server matches RFC-6455. HCA does not implement any of the extensions for per-message compression nor does it implement secure connections (use of “wss:”).

8 Additional Info

There are two consumers of this interface. One use is for individuals working to construct pieces of their own private automation solution. They can use this interface and “hard code” names of devices, programs, etc into their application. In this case they probably will not use the GetDesign method.

Other users may want to use this interface to construct a full application that can work with an arbitrary design. In that case they must use GetDesign. What follows are some notes that may help users create these type of applications.

8.1 GetDesign

Everything starts with GetDesign. The application needs to construct a data structure to hold all the data it returns. The most important piece of data returned is the id. These should be considered an opaque 32 bit handle to the object. The id does not change as long as the application is connected but could change across connections so you can’t store the data and hope to use it next time.

Also in the object state is the folder name. When needed, to form the complete 2-part name take both the object name and the folder name and “glue” them together with <blank><dash><blank>.

If an object has an icon on a display, that icon has an image – the picture, a representation. What is that picture when the device is on or off? What is the text – or label – is below it? These questions are answered initially by what is returned by GetDesign but they may be changed by Update messages. It is important to know that the object name may not be the same as the text below the icon – it is in most cases it is but not always.

How does an icon show a different picture, representation, or label text? It changes by the user using the Change-Icon element in a program or by the device changing state – on to off, for example.

Just concentrating on what is seen on the display, an icon for an object has these pieces of info:

- Icon Picture
- Icon representation
- Text below the icon
- Current icon picture
- Current Icon representation
- Current text below the icon

A possible implementation is that after retrieving information about each object using GetDesign copy the values of the first three listed above into the second three or have the second three be set to “” and that means to use the GetDesign values. Which you choose is an implementation decision.

In the Get Design info is also the state of the object. This sets what is called the icon representation. In the stock set of icons that an application uses, some have multiple representations. For example the

table lamp has an “on” and an “off” representation. Others have only one, like the water pump. When shown on the display if the object has two representations use the “off” one when off and the “on” one when on or dim. For objects with only one representation, show in some way that the object is “on”. The text below the icon could be shown in a “on text” color. That color needs to be part of the application configuration – see below.

Also in the GetDesign info is the *suspend* state of the object. In some way the application needs to show the suspend state. The HCA Control Interface shows this with a black box around the object to show suspended and a green box to show “green suspended”. The Android application uses background colors behind the icon.

In the GetDesign info is an indication of what the object is: Device, Program, Group, or Controller. This info is needed to know what object to use when invoking a method on that object – like Device.On or Controller.On. To a HCA user there is no difference between a device and a controller but at the object level there is. You don’t need to know what the differences are, just invoke the correct methods for the object.

In the GetDesign info are the short and long tap actions for the object. These are encoded as:

- 0 = no action
- 1 = control load
- 2 = show popup

The idea is that for devices they can toggle state when tapped (typically the short tap action) or open a page with controls for On, Off, and Dim (typically the long tap action). Your application should handle these.

There is a lot of configuration by the user in HCA to decide upon what popup is seen when they long tap an object. There are system wide defaults but also the ability to override those defaults on a per device basis. You need not concern yourself with those since the object interfaces determines the popup for you. In the GetDesign info is the “popup name”. This is a text string and tells what the popup page looks like. For example, for a dimmable device the popup name may be “onoffdim”. For a non dimmable it may be “onoff”. There are an fixed set of what all these strings are and there is an example HCA file that has one of each. There are popup names for all of the object types. For programs, for example, the popup name would be “ProgramStart” or “ProgramStartStop”. In the former it would have a single “Start” button and in the latter two buttons: “Start” and “Stop”. You can work with this example file and you will see what all the popup names are when you do a GetDesign.

One big area that takes some exploration is support for “Glass keypads”. Please make sure you understand how they work using the HCA Control UI. The majority of popup types are to handle these kinds of devices. They key thing is that some of these types have rockers (like a switch or keypad), some have buttons, and some have both. A rocker doesn’t show state but a button does. The application

must keep track of the state of each button if the object has buttons. Also rockers and buttons have names. These are in the GetDesign info and the application needs to save that too when it builds a data structure to save the GetDesign info.

8.2 GetDisplays

In addition to Device, Program, Group, and Controller objects there are also displays. A display shows icons for objects. When GetDisplays is used it retrieves info about each display in the design. Displays are also objects and share many properties with the other kinds of objects – devices, programs, etc. They also have an id, a name, state, icon name, short and long tap actions, and the icon label. Like the other objects the application must keep a copy of some of these as the "current" values as they can change: The icon, representation, and icon label.

Display objects have a few other properties. One of which is the display type. This tells if the display is a "field of icons" or the display is HTML, a graph, or text. Once you know what a display type is, if the user "taps into" that display if it is HTML or text then the application must use the appropriate objects and methods to use to get the info to display.

Also with a display object is an array of ids. These are the ids of the objects to show in the "field of icons" that make up what is displayed. They are passed in the order the user wants to see them. The order is upper left to lower right. Don't forget that displays have icons and can be placed on other displays.

8.3 The App "Home Page"

There are two ways to construct the "home page" of the application. One is that the user has named a home page. That should be part of the application properties and is set by the user on a per platform basis. If the user has named a display as the home page then the application shows that display. The display state tells what icons to show and in what order.

In the user has not set a home page then the application has to construct the home page.

In the HCA UI there is a checkbox in the object's properties that says "Don't show an icon for this object in the Control UI". If that checkbox was set for an object that that object is filtered out in what GetDesign returns.

While displays have that same "Don't show" property, in the list of displays returned by GetDisplays, all of the displays are returned regardless of the fact that this property is enabled or not. One of the pieces of data returned by GetDisplay is how that checkbox is set.

If the application is constructing the home page rather than using one named by the user, show an icon for each display returned by GetDisplay except for any of these that have the "don't show" bit set.

Note: Displays with that “don’t show” bit set can’t be filtered out by GetDisplays or discarded by the application as they are returned by GetDisplays because the user could name one of these displays as their home page.

8.4 Update message

The Update message provides changes from the server to the client. Mostly this is to show state changes: On to Off or Off to On. For example when invoking Device.On, the application should not update its internal state for the device. Wait for the update message to provide that update.

The update message gives the id of the object being updated and provides the current icon name, icon representation, and icon label. These may be supplied as ”” in which case use the ones from the GetDesign info for the object.

Again, for the “glass keypads” you may also get button state info for devices that have buttons.

It is important to get this dynamic changing of the icon name, icon representation, and icon label working correctly as this is a key feature that many users rely upon.

8.5 App Configuration

The app should have the ability for the user to customize it.

- Name of Home page display – could be “” in which case the application must construct it
- Background color
- Text color. Color of the text below an icon when not “on”
- “On” label color. Color of the text below the icon when “on”
- Suspend color. Color of the background or box when suspended
- Mode suspend color. Color of the background or box when home mode – or “Green mode” suspended

In addition to this, the setup needs to have the IP address of the server and port number. There should be two IP addresses listed. Try connecting to the 1st and if that fails connect to the second. This lets a user have an internal address and an external address.

8.6 Glass Keypad

There really isn’t much magic in these. All the application needs to do is to show the appropriate page when asked. The “popupname” determines the appropriate page. The rockers should be labeled and the buttons should show their current state. The rocker press and button press methods – in the controller and device objects – are used to carry out the user action.

8.7 Set Client Options

No update messages are delivered unless this method is used to turn them on. After the design is fully loaded then turn on update messages.

8.8 Home Modes

The application must have some method for the user to view and change the current home mode. There is a method to get the names of the modes, a method to get the current mode, and a method to set the current mode. How the user gets access to a UI for this is up to the application. One way would be something from the app menu.

8.9 Thermostats

There needs to be special processing for thermostats. To know if a device is a thermostat, look at the "popupname". There are several that are thermostat specific.

8.10 IR Keypads

IR Keypads can be difficult. There is a method to get the keypad and it reports what buttons exist and where they are. The coordinates returned are in pixels from the HCA User Interface and they must somehow be mapped into the coordinates of the platform display space. Again, the application knows it has an IR device because of its popup name.

8.11 Reconnection

In implementing the Android application there were a lot of connectivity issues. It isn't known how other platforms handle this. For example, when going from 3G to 4G or from wireless to 3G the Android system would disconnect and reconnect.

If the application disconnects the design can be totally reload or it can request the timestamp and see if the timestamp matches what is in the application's state. There are two parts to the timestamp. When the design as last modified and when the last state update happened.

The application doesn't need to understand the data in the timestamp. After start up and processing of GetDesign and GetDisplays the application should invoke the TimeStamp method and save the result. Each Update messages contains a timestamp and the application should keep its "last state updated" timestamp up to date from the value in the Update message.

During the reconnect the application can check if the design timestamp matches and if it does then it doesn't have to reload. If the state timestamp also matches then there is nothing else that needs to be done. If the state timestamp doesn't match then the application can invoke the Refresh state method supplying the saved timestamp received in the last update message. The server then sends update messages for any updates since then.

Just to say it again: The application doesn't have to understand the values in the timestamps. Just save them and pass them to RefreshState as needed.