Alive with Activity, Part 1: User experience, capabilities, and setting up live tiles and notifications

Kraig Brockschmidt

Live tiles and the related capabilities of badges, toast notifications, and push notifications, are one of the most distinguishing features of Windows 8 and Windows Store apps. They combine to create a system that is “alive with activity”: apps have a constant flow of information coming in from their services that’s visible on both the Start screen and the lock screen, even when those apps aren’t running.

Previous posts on this blog have covered a number of the essentials where live tiles and notifications are concerned. To review:

* [Creating a great tile experience, part 1](http://blogs.msdn.com/b/windowsappdev/archive/2012/04/16/creating-a-great-tile-experience-part-1.aspx): introduces tile design and choosing a template based on the function of your app.
* [Creating a great tile experience, part 2](http://blogs.msdn.com/b/windowsappdev/archive/2012/04/18/creating-a-great-tile-experience-part-2.aspx): discusses how an app sends local updates, sets up periodic notifications, and manages secondary tiles. It also introduces the structure of services in both PHP and ASP.NET.

To build on these resources, this three-part post will delve more deeply into several areas. In Part 1 (this post) we’ll cover:

* A brief visual tour of the user experience: what does this “aliveness” mean for the user?
* The XML schema for tiles, badges, and toast notifications, which reveal a number of features and capabilities that aren’t always obvious when looking at the template catalogs.
* The relationship between tiles, notifications, and background tasks.

Part 2 will discuss writing and debugging services in several languages, specifically employing the localhost along with using Visual Studio 2012 Express for Web/Visual Studio 2012. Part 2 will also touch on using Windows Azure Mobile Services for this purpose.

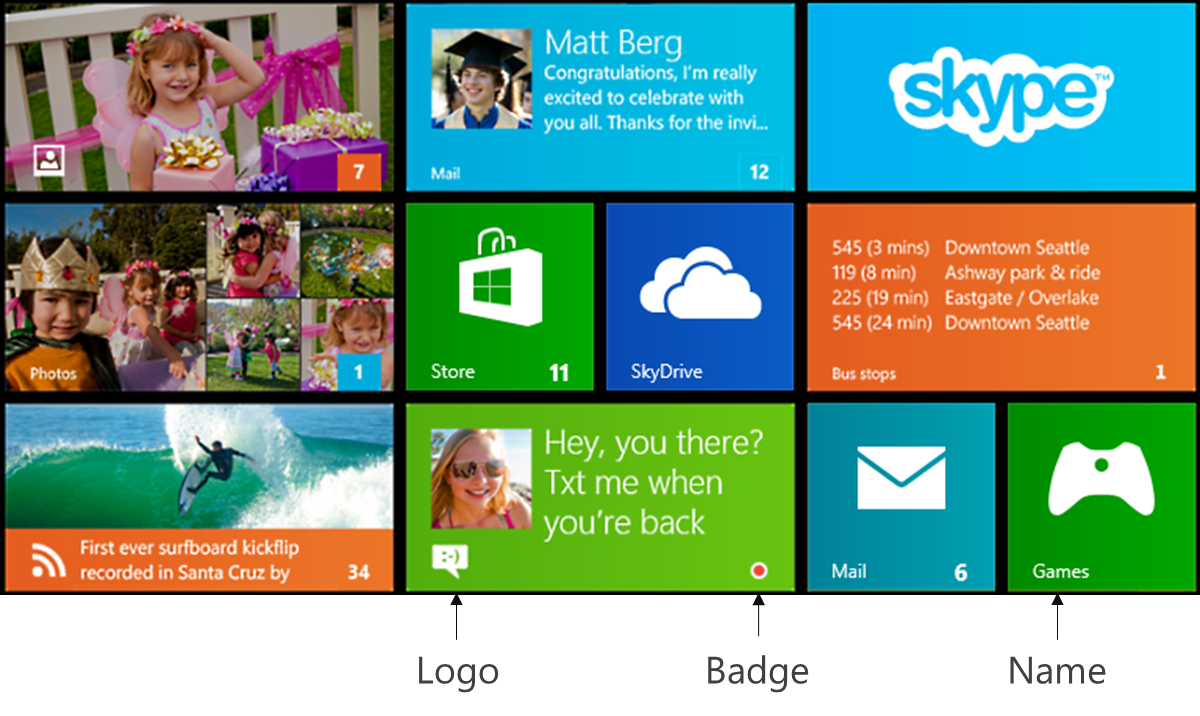
Part 3 will then explore the subject of push notifications. This includes how to work with the Windows Push Notification Service (WNS) and how to use Windows Azure Mobile Services to support push notifications in your app.

These three parts echo and extend the session I presented at //Build 2012, [3-101 Alive With Activity](http://channel9.msdn.com/Events/Build/2012/3-101), for which the video and slide deck are available on Channel 9. It also reflects and extends the material in Chapter 13 of my free ebook from Microsoft Press, [*Programming Windows 8 Apps in HTML, CSS, and JavaScript*](http://blogs.msdn.com/b/microsoft_press/archive/2012/10/29/free-ebook-programming-windows-8-apps-with-html-css-and-javascript.aspx) (which, despite the title, is also helpful to developers working in other languages).

# What Does “Aliveness” Mean for the User?

The “aliveness” that we’re talking about in this series appears to the user in three places: the Start screen, the lock screen, and anywhere else when a toast notification appears.

On the Start screen, you’re by now undoubtedly familiar with the idea of **tiles**. As shown in the examples below, tiles can be square or wide, with a variety of layout templates to display text, images, or both together. Colors are defined by the app’s manifest; the user can choose whether a tile is square or wide provided that an app supports both. The lower left of the tile can show the app name, a logo, or nothing; and the lower right may contain a small ‘badge,’ either a number from 1-99 (as for mail, messages, etc.) or one of a number of pre-defined glyphs (see the [Badge image catalog](http://msdn.microsoft.com/library/windows/apps/Hh761458.aspx)).

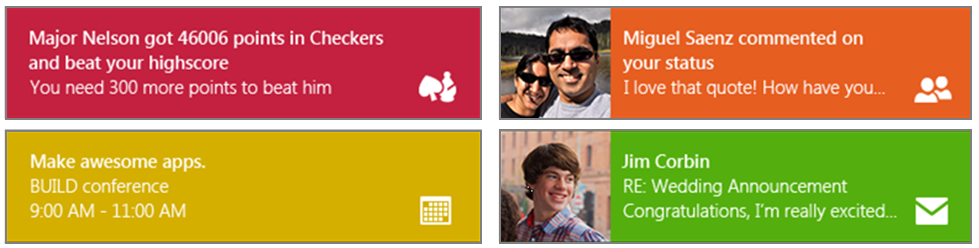


The tile templates also include a number of “peek” designs that flip between one part of the content, like an image, and a second part, like text (in the [Tile template catalog](http://msdn.microsoft.com/library/windows/apps/hh761491.aspx) they look like double-height tiles, but that just shows the two halves together). The system will animate the transitions between these two parts in harmony with all other tile animations. Additionally, every tile can have a queue with up to five updates that automatically cycle, which are again animated by the system in coordination with other tiles. (Each update in the queue can be given a tag so you can replace each one individually, otherwise the queue is first in-first out.)

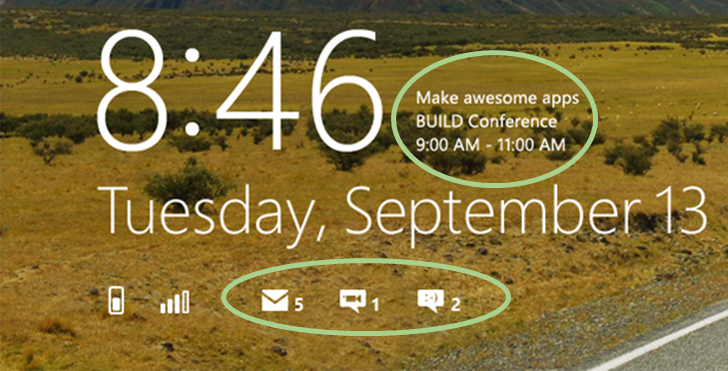
This coordination between animations helps us understand why Windows restricts tile updates to a set of pre-defined templates. Consider what the Start screen experience would be like if apps could just do whatever they wanted on their tiles. Besides the fact that the freedom to play videos on tiles would drain the battery much more quickly, such a free-for-all would be a sure recipe for utter chaos. It would be the counterpoint to a wholly static start screen, for sure, but not particularly pleasant to look at! Windows’ designers wanted a Start screen that you’d enjoy looking at for potentially long periods of time, so they created an experience that would be alive and active but also express a sense of unity and harmony across all the tiles. Using templates and system-controlled animations provides some degree of sameness, while still allowing quite a bit of variation between apps. (And remember that you can always use an image-only template and place any design you want therein, but avoid creating imagery like buttons that suggest different behaviors for different parts of the tile, which aren’t supported.)

With **toast notifications**, these appear on top of the Start screen, the lock screen, the desktop, or whatever app happens to be running. Tapping a toast activates its associated app with any arguments that were specified when the toast was issued. A toast appears in the color define by the associated app’s manifest and includes the app’s logo on the lower right; both the color and the logo help the user identify the app that will be activated if they tap the toast.

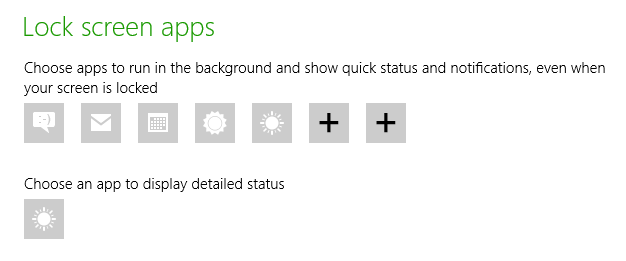
As with tiles, the layout of each toast is defined by a template from the [Toast template catalog](http://msdn.microsoft.com/library/windows/apps/hh761494.aspx), where the choices provide for text-only and text+image options. Toasts appear for a time and then fade away, can be accompanied by a sound, and can repeat themselves as with a meeting reminder.



On the **lock screen**, app activity is displayed alongside the date, time, and system badges. As shown in the example below, app content consists of up to seven logos + badges along the bottom, and one text update next to the time:



The user controls what appears on the lock screen through PC Settings > Personalize > Lock Screen, an example of which is shown below. On the first row, tapping one of the squares brings up a list of apps that are lock-screen capable and will appear as a logo and badge. On the second row you can choose the app that provides the detailed text. This text specifically comes from the most recent tile update for that app.



# Update XML

Now that we’ve seen how the user experiences aliveness, the next question is: what exactly are these updates made of? That is, what describes a tile update, a badge, or a toast notification?

In all cases except for what are called *raw* notifications, updates are simply pieces of XML that Windows can process to update a tile or display a toast. (Raw notifications contain app-specific text or binary data. An app or background task that receives these notifications typically process that data and issue tile updates, badge updates, or toasts in response.)

The templates we’ve been referring to are just specific XML structures for these updates. For example, a badge update for the number 7 as seen in the upper left tile we saw earlier looks like this:

<badge value="7"/>

A similar *payload*, as these bits of XML are often called, for the red dot badge on the lower middle tile we saw earlier is as follows.

<badge value="busy"/>

The XML payload for that same wide tile, which includes a reference to its image, the text, and the logo/name options, would be more or less as follows (the image URI is fictitious), using the *TileWideSmallImageAndText01* template:

<tile>

<visual>

<binding template="TileWideSmallImageAndText01" branding="logo">

<image id="1" src="http://friends.contoso.com/sally.png" alt="Sally's picture"/>

<text id="1">Hey, you there? Txt me when you’re back</text>

</binding>

</visual>

</tile>

In this payload, you can see that we identify the template within the *binding* element. Windows validates the rest of the XML against that template. The *branding* attribute of the *binding* element indicates whether to show the *logo*, the app’s *name*, or *none*. The child elements of *binding* then describe the other necessary parts of the template.

What’s instructive now is to look not so much at the specific templates for such updates but rather their general XML schema, as those schema reveal additional capabilities that can be utilized with each update. That is, the [Tile template catalog](http://msdn.microsoft.com/library/windows/apps/hh761491.aspx), [Badge image catalog](http://msdn.microsoft.com/library/windows/apps/Hh761458.aspx), and [Toast template catalog](http://msdn.microsoft.com/library/windows/apps/hh761494.aspx) only show the required elements, the schema, on the other hand, show up all the additional options.

The schema are documented on the [Tile, toast, and badge schemas](http://msdn.microsoft.com/library/windows/apps/br212853.aspx) section of the documentation. However, the details are spread across many separate topics, so let me condense them here into pseudo-schema that allow us to examine the structures as a whole.

**Badge updates** are the simplest, so let’s start there:

<?xml version="1.0" encoding="utf-8" ?>

<badge value = "1-99" | "none" | "activity" | "alert" | “attention” | "available" |  
 "away" | "busy" | "newMessage" | "paused" | "playing" | "unavailable" | "error"   
 version? = "integer" />

Technically, all XML payloads should have the *<?xml>* header tag; Windows works fine without them, but I like to include them for completeness.

Next you can see that the *badge* element just contains a *value* attribute that can be a number (anything greater than 99 actually displays as “99+”), or one of a set of pre-defined words that identify specific glyphs. These are again documented on the [Badge image catalog](http://msdn.microsoft.com/library/windows/apps/hh761458.aspx) page in the documentation.

The only other attribute is the optional *version* (the *?* after the name means optional), which allows for future changes to the schema. Being optional you can omit it in your payloads at present.

**Tile updates** have the following pseudo-schema:

<?xml version="1.0" encoding="utf-8" ?>

<tile>

<visual version? = "integer" lang? = "string" baseUri? = "anyURI" fallback? = "string"  
 branding? = "none" | "logo" | "name" addImageQuery? = "boolean" >

<!-- One or more binding elements -->

<binding template = "TileSquareImage" | "TileSquareBlock" | "TileSquareText01" | ...

fallback? = "string" lang? = "string" baseUri? = "anyURI"

branding? = "none" addImageQuery? = "boolean" >

<!-- Some combination of image and text elements -->

<image id = "integer" src = "string" alt? = "string" addImageQuery? = "boolean" />

<text id = "integer" lang? = "string" />

</binding>

</visual>

</tile>

In the *visual* element, which contains attributes that apply to the update as a whole, we see the optional *version* attribute, which you can again omit at present. The other attributes are described as follows:

|  |  |
| --- | --- |
| Attribute | Description |
| *lang* | An optional BCP-47 language string identifying the language of the update’s data, such as “en-US” or “de-DE”. |
| *baseUri* | An optional URI that will be prepended to all other URIs in the payload, allowing you to omit such redundant information when constructing the payload. The default is “ms-appx:///”. |
| *branding* | Indicates whether to show the app’s logo on the tile (“logo”, the default), its name (“name”), or nothing (“none”). |
| *addImageQuery* | If set to “true” (default is “false”), instructs Windows to append query parameters to every request made to URIs in the payload, that is, to each image URI, to identify the current language, scaling factor, and contrast setting. The parameters have the form:  ?ms-scale=<scale>&ms-contrast=<contrast>&ms-lang=<language>  For more details, see [Globalization and accessibility for tile and toast notifications](http://msdn.microsoft.com/library/windows/apps/Hh831183.aspx). |

Within *visual,* the payload then has one or more *binding* elements, one for each tile size. That is, a single payload can (and ideally should) contain an update for both square and wide tiles simultaneously, which is important because the user can change the size of the tile at any time. If you omit one size or the other, then nothing will appear for that update if the user has set the tile to that particular size. Do note, however, that you can send updates for the different sizes separately, and Windows will retain both.

Within *binding*, you can see that *template* identifies the XML template to validate, and you see the same attributes that exist for *visual* except that they apply only to those elements in this *binding* (and override those of *visual*). The one other attribute is *fallback*, which identifies a template to use if the primary template cannot be found. This is for future use with backward compatibility so there’s no need to use it with Windows 8.

Within each *binding*, then, there is some combination of *image* and *text* elements depending on the template; the attributes for each of these are, I trust, self-explanatory or have already been described. Note that any attributes in *image* or *text* that appear also in *binding* or *visual* override the values of those in the parent elements, as you’d expect.

For **toast notifications**, we have a similar structure to the tiles, but with some additional features:

<toast launch? = "string" duration? = "long" | "short" >

<visual version? = "integer" lang? = "string"

baseUri? = "anyURI" branding? = "none" | "name" | "logo" addImageQuery? = "boolean" >

<!-- One or more bindings -->

<binding template = "ToastImageAndText01" | "ToastImageAndText02" | ...=""

fallback? = "string" lang? = "string" baseUri? = "anyURI"

branding? = "none" addImageQuery? = "boolean" >

<!-- Some number of child elements -->

<image id = "integer" src = "string" alt = "string" addImageQuery? = "boolean" />

<text id = "integer" lang? = "string" />

</binding>

</visual>

<!-- Optional audio -->

<audio src? = "ms-winsoundevent:Notification.Default" | ...=""

loop? = "boolean" silent? = "boolean" />

</toast>

All of the *lang, baseUri, branding,* and *addImageQuery* attributes within *visual* have the same meaning and defaults that they have for tiles, as do all the attributes within the *binding, image,* and *text* attributes. In this part of the payload, then, the only real difference between a tile update and a toast notification are the supported templates.

What differentiates a toast payload are the attributes of the outermost *toast* element, and its optional *audio* child.

With *toast*, the *launch* attribute can be assigned a string that will be passed to the app’s activation handler as arguments, exactly as happens with a secondary tile. The *duration* attribute can have values of “short”(five seconds or the value from PC Settings > Ease of Access) or “long” (25 seconds or the value from PC Settings > Easy of Access, whichever is longer.

In *audio*, its *src* attribute is a string that indicates one of a set of pre-defined sounds as found on the [Toast audio options catalog](http://msdn.microsoft.com/library/windows/apps/Hh761492.aspx). At present, you can only choose from these sounds, which are generally mapped into whatever system sounds you’ve configured in Control Panel. (Note: if you turn off all sounds, as I typically do, you won’t hear anything—remember that when you’re testing your app!). This limitation exists for a number of reasons. For one, including custom sounds from a remote source would generate a great deal more network traffic, which would be a concern when the device is on the lock screen and trying to conserve power in connected standby mode. It also prevents potential abuse, such as having including audio ads with toasts.

Separately, the *silent* attribute, if set to “true”, will always mute the audio. As for *loop,* if the toast’s *duration* attribute is set and you set the audio’s *src* to one of the four “Looping” sounds in the catalog (two variations each on ‘alarm’ and ‘call’), you can also set *loop* to “true” to repeat the sound or “false” to play the sound only once, the default.

Aside from looping the audio, it’s also possible to schedule a recurring toast. This isn’t part of the XML payload, however, but rather something you indicate when creating a [*ScheduledToastNotification*](http://msdn.microsoft.com/library/windows/apps/br208614.aspx) object. In these cases, the same payload (audio and all) will appear at each interval.

# Alive with Activity without Apps Running

Knowing which XML payloads will generate which kinds of updates, the next matter to address is how those payloads get to the system at the appropriate times, and how those payloads are created. Both of these topics are discussed in the earlier posts, [Creating a great tile experience, part 1](http://blogs.msdn.com/b/windowsappdev/archive/2012/04/16/creating-a-great-tile-experience-part-1.aspx) and [Creating a great tile experience, part 2](http://blogs.msdn.com/b/windowsappdev/archive/2012/04/18/creating-a-great-tile-experience-part-2.aspx), which show how to generate the payloads in code. This includes using the [Notifications Extensions Library](http://msdn.microsoft.com/library/windows/apps/hh969156.aspx) that’s included with a number of the samples and is also available through Visual Studio as a NuGet package.

Here, let’s focus on summarizing the methods of delivering these payloads. There are three ways that they can be issued:

1. A running app or an app’s background task can issue **direct** or **local updates**. These can be immediate or be scheduled for some time in the future. Tiles, badges, and toasts can also have an expiration time after which they’ll be automatically removed or removed from the schedule if they haven’t been delivered by that time.
2. A running app can give Windows up to five URIs for **periodic updates** and specify intervals between 30 minutes and one day. (When multiple URIs are used, each one corresponds to one of the five slots in the update queue.) Once a periodic update is configured, Windows sends a request (HTTP GET) to that URI at each interval. If Windows receives back a valid payload, it then sends that update to the appropriate tile on the app’s behalf, just as if a running app had issued it directly. We’ll look at how to create update services in Part 2 of this series.  
     
   The primary benefit of periodic updates is that the requests will continue even when the app isn’t running, allowing the tile to receive ongoing updates and badges for a long time without the user launching the app at all. Of course, I hope your updates are interesting enough to encourage the user to engage with your app again!
3. An app can request a Windows Push Notification (WNS) channel URI and send that URI to its own back-end services. Those services can then issue specific updates of any kind through that channel URI through WNS, which will in turn send that update to that specific device when it’s online. Windows receives these updates and applies them to the appropriate tile or displays the toast. Alternately, an app can configure a background task to receive these push notifications (which is required for a raw notification). We’ll discuss push notifications in Part 3 of this series.

The following table summarizes these options, and includes the cycling queue feature of tiles and the recurring/audio capabilities of toasts:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Notification type | Queue | Scheduled | Expiring | Recurring | Audio | Periodic | Push |
| Tile | ✔ | ✔ | ✔ |  |  | ✔ | ✔ |
| Badge |  |  | ✔ |  |  | ✔ | ✔ |
| Toast |  | ✔ | ✔ | ✔ | ✔ |  | ✔ |
| Raw |  |  |  |  |  |  | ✔ |

It’s important to note that everything you see here for tiles and badges applies to an app’s primary and secondary tiles alike. The earlier blog posts referenced above provide details for secondary tiles. You can also refer to the [Secondary tiles sample](http://code.msdn.microsoft.com/windowsapps/Secondary-Tiles-Sample-edf2a178).

Speaking of samples, let me also refer you to the [App tiles and badges sample](http://code.msdn.microsoft.com/windowsapps/App-tiles-and-badges-sample-5fc49148), the [Scheduled notifications sample](http://code.msdn.microsoft.com/windowsapps/Scheduled-notifications-da477093), and the [Push and periodic notifications client-side sample](http://code.msdn.microsoft.com/windowsapps/Push-and-periodic-de225603) for complete examples on all the different scenarios. For any work with tiles and notifications in your app, it’s very much worth taking the time to become familiar with the code in these samples.

In all of this, remember too that tile, badge, and toast updates can be issued from background tasks. I like to think of background tasks as a form of app code that gets to run even when the full app itself is either suspended or not in memory at all. Issuing an update from a background task happens with the same code as used the main app, so there’s no real difference there.

Background tasks are great for checking the status of various conditions, which can be done quickly and thus respects the CPU quotas placed on those tasks. When conditions demand it, the background task can generate an appropriate notification. In fact, the primary actions of a background task are to (a) issue such updates or (b) save values into the app’s AppData folders or settings container that the app will process when it next runs.

Background tasks are also essential for handling raw push notifications when an app isn’t running, which we’ll again return to in Part 3.

For more on background tasks, refer to the earlier post on this blog, [Being productive in the background](http://blogs.msdn.com/b/windowsappdev/archive/2012/05/24/being-productive-in-the-background-background-tasks.aspx), along with the more general overview, [Being productive when your app is offscreen](http://blogs.msdn.com/b/windowsappdev/archive/2012/05/16/being-productive-when-your-app-is-offscreen.aspx).

And with that, we’re now ready to leave the familiar confines of client apps and explore the role of online services with your tiles, badges, and toasts, as we’ll do in Parts 2 and 3.

Kraig Brockschmidt  
Program Manager, Windows Ecosystem Team  
Author, [*Programming Windows 8 Apps in HTML, CSS, and JavaScript*](http://blogs.msdn.com/b/microsoft_press/archive/2012/10/29/free-ebook-programming-windows-8-apps-with-html-css-and-javascript.aspx)