Hi, I’m Ian Cottam, a senior software engineer on the iSite product team, I’m going to talk about the ToolsFramework js library today, specifically its use on iSite to integrate Castaway as an activity and changes that have been made to the library recently.

You might be asking, what is ToolsFramework, hopefully you’ll have a vague idea from a presentation before Christmas, but basically it’s a JavaScript library that allows you to open an application from within another application. The idea behind it is to improve user experience, so if you have users that often jump between castaway and isite for example in different windows, tools framework will allow them to do everything in one window and one workflow. So that’s the basic idea.

The example I’ll be using today is integrating Castaway into iSite. So user’s of iSite can upload videos without have actually navigate away from iSite, they can use castaway from within isite.

A bit of terminology for you is that the Host is the application that the user starts on and is doing their main work in. e.g. iSite. The Client is the application that the user wants to do a specific task in whilst in the middle of using the host. E.g Castaway. The ‘Activity’ is whatever that specific task is. E.g uploading a video.

Let me show an example of this now to provide some context and show something interesting!

This is the iSite form builder, here you can link to video and image files so previous you’d upload to iBroadcast, then browse iBroadcast here and upload your file. But with tools framework you can do it all through the iSite sidedrawer.

I can click ‘upload to castaway’, open castaway in a sidedrawer, select and upload a file, fill in all the meta data, publish and the castaway will start the publishing process and generate a PID.

Whilst it’s generating the PID is being generated, we want to prevent users from closing down the sidedrawer here. If they did, the upload process would continue, but they won’t know what the PID generated for their upload is without going into iBroadcast so we have a ‘complete actvity’ setting which we use to warn the user if they’re about to close down the activity without having completed it properly. When you click either the grey area or the close button a ‘close request’ is sent. If the activity isn’t marked as finished then this bar will appear.

For instance, if I click the grey area, I get this notification that prevents me from closing the activity, same for clicking on the close button here.

Once the PID has been generated a notification appears to inform the user that they can complete the activity. The user clicks the ‘Complete’ button and sends the PID to isite, the video picker populates and we’re finish with the activity.

**5MINS**

You might be wandering, which application does what in this process? How much coding is needed on the Client and how much is needed on the Host?

The idea of tools framework is to keep integration simple and not have to do much coding at all. This is all the code needed on the host (isite).

Explain what the settings are….

‘Action’ and ‘type’ are used to specify which acticity to launch – Tools Framework will check the manifest.js file and work out which activity to launch from these two options – show the file.

‘onComplete’ is the call back that is fired when we click the ‘Complete activity button’. In this case it’s a function that expected a PID as a parameter, and puts the PID into the video picker

‘Launcher’ is the type of UI component to use to house the activity. Here we specify a sidedraw, you can also specify a modal and an element to launch into on the page. The css is shipped with TF if you want to use it. (SAY SOMETHING ABOUT IN GEL MATTER).

Client:

clientUi is used to a create an application bar the overlays the default one. It is used to blank out features on the native bar and prevent the user from getting distracted from their activity. You can specific an icon and the native app bar which needs replacing.

Call promptComplete when the activity is finished, in this case that’s when a new PID is created. It takes two parameters:

1. A message to send to display to the user
2. Some data to send to the host, which is passed to a call back on the Host

That’s all the code on this activity.

What’s next for Tools framework? Probably the biggest planned use of TF in the future to use it to integrate search as an activity. At the moment an editorial platform search is being developed and we want to be able to provide an interface for this that can be customised through options and integrated into applications with TF. Blaise is going to say a bit more about this later on, but I’ve stolen one of his slides to help show roughly what we want to achieve. The idea is that application developers can specify which features of the activity they want, what kind of data to expect back from the activity and where to launch it, so there’s minimal effort to integrate a search.

Tools framework was originally launched as a beta and so we’ve spent some time since Christmas improving the library and adding new features. Probably the most crucial feature that’s been added is end to end tests. We have a demo app that launches a very basic activity, but it allows us to test out all the launchers, all the transfer of data between host and client, the error message and lots of other features. These e2e tests are hopefully going to help stop us breaking things!

As we’re adding new features, especially around sending messages from the host to the client apps there’s a possibility that if the Host and Client are running different versions of the TF library those versions will be incompatible and won’t work properly together, so we’ve added a version check, you have to be using the same major release of TF or an error will be thrown. This way we fail loudly to the user, provide them a message saying there are incompatible versions and avoid issues when they’re half way through an activity.

We’ve added a modal launcher, it’s pretty similar to the sidedrawer launcher, but gives users another option for how to integrate an activity into their UI.

Error reporting to sumo logic, we attempt to catch all the errors that might occur and log these in sumo logic.

And finally we provide a CSS file that contains all the styles for the various ui components. This can be required, or just the sass files that you need.

The final thing to mention is related to the UI components for Tools Framework. You might have been wandering where the CSS for the side drawer comes and the notification bars come from. You could write all your own CSS in your applications but that would of course be extra effort and so we’re using a component library called INT GEL MATTER. This a library that has CSS for a side drawer, notification bar, dialog, date picker, time picker and more. IF you want to hear more, we’ll probably do a presentation on that soon too so come again ☺