Customizing the Office 365 Profile Card with Graph Explorer

Posted on [**August 5, 2020**](https://office365itpros.com/2020/08/05/customizing-office-365-profile-card-graph-explorer/) by [**Tony Redmond**](https://office365itpros.com/author/thoughtsofanidlemind/)

Making the Office 365 Profile Card Your Own

In Office 365 notification MC217813 (published 1 July and updated on 15 July), Microsoft announced that: “*You will be able to customize profile cards according to your organization business needs. The profile card is sometimes referred to as a contact card or people card.*” (Microsoft 365 [**roadmap item 61502**](https://www.microsoft.com/en-ie/microsoft-365/roadmap?filters=&searchterms=61502)).

The new functionality sounds promising. The profile or people card displays information about someone in different places in Office 365 applications. For instance, you can select an email sender or recipient in Outlook or Outlook mobile to view the details of that person. If they’re someone in your organization, the profile card shows you information from their Azure Active Directory account. The information is more limited for external people.

Customizing the Profile Card

The announcement points to a [**page with details about how to customize the profile card**](https://docs.microsoft.com/en-us/graph/add-properties-profilecard) (the feature is marked as a preview). Essentially, you customize the **[ProfileCardProperty](https://docs.microsoft.com/en-us/graph/api/resources/profilecardproperty?view=graph-rest-beta" \t "_blank)** Graph resource to add new information to the profile card. Six standard Azure Active Directory properties can be added along with the fifteen custom attributes available for organizations to use as they wish (like *CustomAttribute11*, which Microsoft recently considered using to identify the [**room mailboxes needed for its Workspaces feature**](https://petri.com/exchange-online-workspaces)). Custom attributes are used for many purposes, including to hold organizational information (like cost centers and division names), mark mailboxes for special processing, and so on.

Nice as it is to be able to customize the profile card, the immediate barrier facing many tenant administrators is not knowing how to interact with the Graph to update the ProfileCardProperty resource. Not everyone has mastered Graph programming, and administrators who know PowerShell might not yet have ventured into interacting with the Graph (some examples of using PowerShell with the Graph are listed below).

Graph Explorer Delivers an Answer

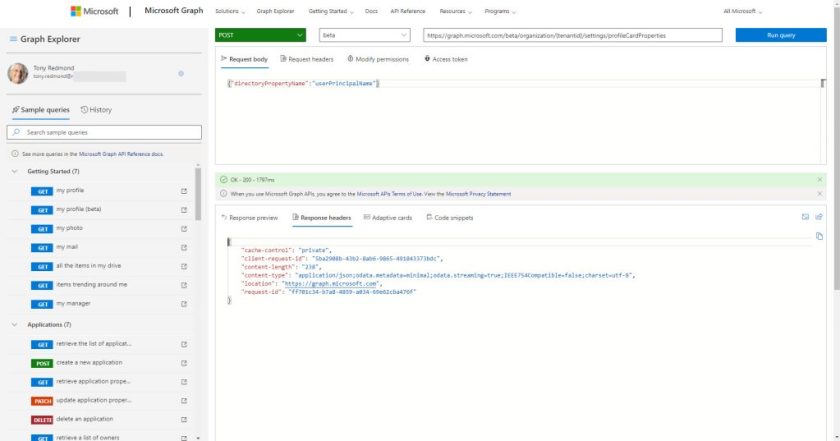
As it turns out, the [**Graph Explorer**](https://developer.microsoft.com/en-us/graph/graph-explorer) makes it easy to apply the necessary changes. This is a browser interface to allow people to interact with the Graph and get to know how Graph transactions work against different endpoints, such as Groups, Outlook, Planner, and SharePoint. You can run commands against test data or, after signing into a tenant account, against live data. Programmers can grab code snippets generated for transactions by the Explorer in C#, Java, JavaScript, and Objective-C and include the code in their programs. All in all, the Graph Explorer is a very useful tool.

Using Graph Explorer to Update the Profile Card

The key points about using the Graph Explorer to update the ProfileCardProperty resource are:

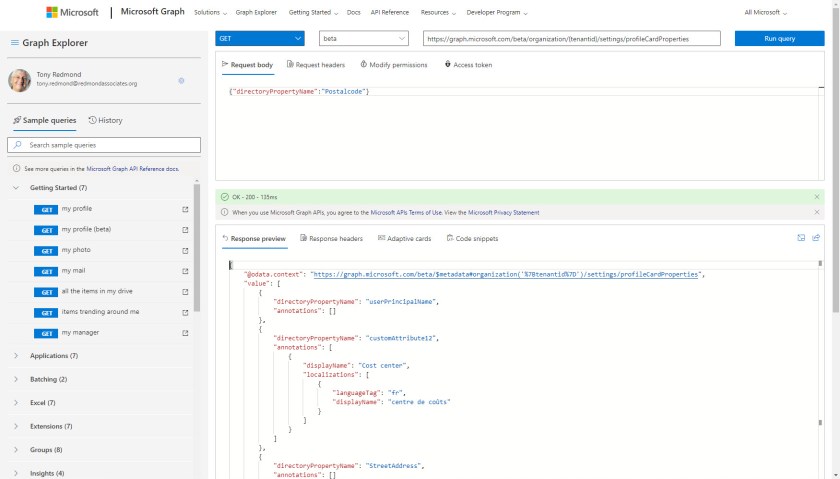
* Sign in with an administrative account for the tenant.
* Select *beta* from the drop-down list for endpoint types.
* Use a POST command to update the [***https://graph.microsoft.com/beta/organization/***](https://graph.microsoft.com/beta/organization/)*{tenantid}/settings/profileCardProperties* endpoint.
* Put the update you want to execute in the request body. For example, to add the PostalCode property to the profile card, I put {“directoryPropertyName”:”Postalcode”} in the request body.
* Click Run query when you’re ready. If everything’s been done right, you should see a 200 response. This means that the Graph has accepted the update.

Figure 1 shows the Graph Explorer after running a successful update. You can see the 200 response.

Figure 1: Using the Graph Explorer to update the ProfileCardProperty resource with a POST command

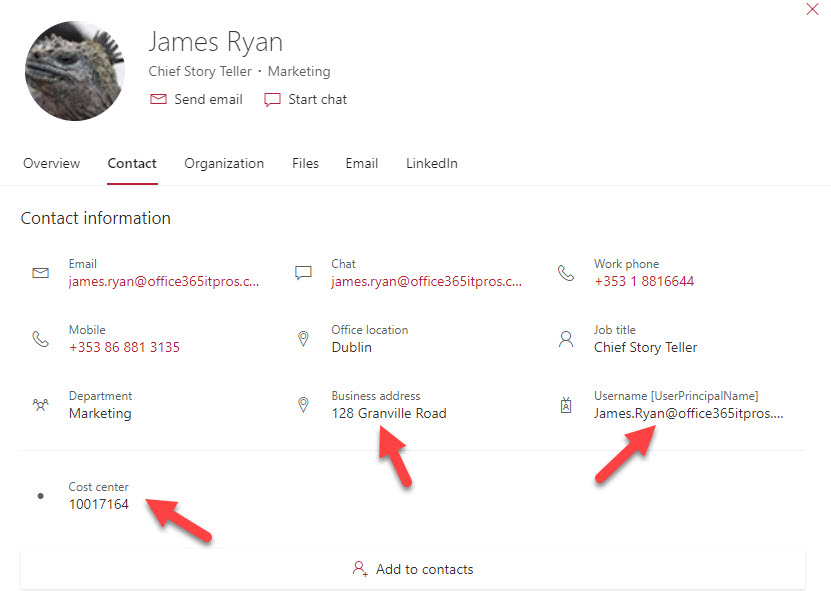
Microsoft warns that it takes up to 24 hours for the changes to show on profile cards. It also seems that you need to add updates one by one and wait until an update is active before applying another. To check the current state of the ProfileCardProperty resource, you can run a GET command to see what’s returned by [***https://graph.microsoft.com/beta/organization/***](https://graph.microsoft.com/beta/organization/)*{tenantid}/settings/profileCardProperties.*

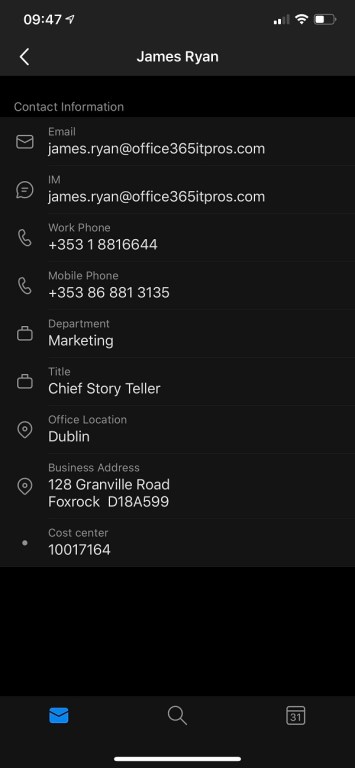
Figure 2 shows that I have added two of the standard properties (UserPrincipalName and StreetAddress) and CustomAttribute12. Note that the custom attribute is assigned a display name of “Cost Center” to make its use more obvious to end users.

Figure 2: Checking updates applied to the ProfileCardProperty resource with a GET command

Viewing Custom Profile Cards in Applications

After Graph processes everything and applications refresh their cache, you’ll see the customizations show up in the profile cards displayed by applications. Figure 3 shows how OWA displays customized contact information while Figure 4 shows how the profile card appears in Outlook Mobile.

Figure 3: OWA displays a customized profile card

Figure 4: Outlook Mobile displays a customized profile card

The customized attributes don’t appear for guest accounts.

If you make a mistake and add the wrong attribute or want to start over, you can remove an attribute from the organization settings by running a DELETE command in the form:

*https://graph.microsoft.com/beta/organization/[tenantID]/settings/profileCardProperties/[propertyName]*

Graph Explorer Gets Real Work Done

The notion of having to write code to interact with the Graph might well have turned people away from even considering customizing the profile card. The Graph Explorer allows you to do the job without writing a single line of code. It’s a great example of using an available tool to achieve a goal in a way that the authors of the feature probably never considered.