Ad Parameterizer



Displaying up-to-date and dynamic values in ad text can be very compelling for viewers. An advertiser who wants to advertise their prices benefits by keeping their ad text current. Deleting and re-adding an ad loses any historical information as well as having to wait for policy checks. Using AdWords scripts with ad parameters makes it possible to fetch live data and update an ad to reflect current prices or change any other variables.

This script uses US Geographical Survey earthquake data. We add and remove keywords so that we target earthquakes that happened in the last 30 days. The number of earthquakes and magnitude of the smallest one are dynamically updated in the ad text via ad parameters.

In real advertising campaigns, you'll have your own datasources—whether they be stored in Google Spreadsheets or other data feeds retrievable via UrlFetchApp. You can modify this script to incorporate your own data into this example instead of using earthquake data.

How it works

**Which ad group?**

The first thing you'll need is an ad group that has been set up for use with ad parameters. In the set-up phase, you configured a text ad with keyword insertion and ad parameters. The script looks up this ad group using a condition in an AdGroupSelector.

function getAdGroup(name) {  
  var adGroupIterator = AdWordsApp.adGroups()  
      .withCondition('Name = "' + name + '"')  
      .withLimit(1)  
      .get();  
  if (adGroupIterator.hasNext()) {  
    return adGroupIterator.next();  
  }  
}

**What data to use?**

The next step is to process the USGS earthquake JSON feed which we'll use later to set our keywords and ad parameters.

function getEarthquakeData() {  
  // We're looking at the past 30 days of significant quake activity.  
  var url = 'http://earthquake.usgs.gov/earthquakes/feed/geojson/significant/month';  
  var json = Utilities.jsonParse(UrlFetchApp.fetch(url).getContentText());  
  var earthquakes = {};  
  // Iterate over all earthquakes in the response json.  
  json.features.map(function(quake) {  
    var placeName = extractPlaceName(quake.properties.place);  
    // If we haven't seen this place yet, add an empty record.  
    if (!earthquakes[placeName]) {  
      earthquakes[placeName] = {count: 0, lowestMagnitude: 10.0};  
    }  
    // Increment the count  
    earthquakes[placeName].count += 1;  
    // Update lowest magnitude seen.  
    if (quake.properties.mag < earthquakes[placeName].lowestMagnitude) {  
      earthquakes[placeName].lowestMagnitude = quake.properties.mag;  
    }  
  });  
  return earthquakes;  
}

The place name returned in the feed looks like "2km NE of Yorba Linda, California". Since we want our ad to incorporate the number of earthquakes in a region, we use a regular expression to obtain the general region name. Using this value with the text "earthquake", we have a keyword that should trigger whenever someone searches for information about an earthquake in the area.

**Updating keywords**

The USGS data only includes the last month's worth of data. To avoid advertising with earthquakes that happened a long time in the past, we should remove keywords for earthquakes that don't appear in the feed. The logic below iterates over all the keywords in the adgroup—if the keyword doesn't exist in the list that we want to advertise for, we remove it from the ad group. Any keywords we see during the loop get removed from our keywords object, which leaves just the keywords we need to add.

// Removes keywords that are no longer relevant and adds new keywords.  
function updateKeywords(adGroup, earthquakeData) {  
  var keywords = {};  
  // Make a copy of all expected keywords so we can add ones that don't exist later.  
  for (var keyword in earthquakeData) {  
    keywords[keyword] = keyword;  
  }  
  var keywordIter = adGroup.keywords().get();  
  while (keywordIter.hasNext()) {  
    var keyword = keywordIter.next();  
    // If the keyword from the adgroup isn't in the set of ones we want, remove it.  
    if (!(keyword.getText() in keywords)) {  
      keyword.remove();  
    }  
    // Since we do want this keyword and it exists, remove it from the list so we  
    // add only the ones that don't exist.  
    delete keywords[keyword.getText()];  
  }  
  // Now create all the keywords that didn't already exist.  
  for (var keyword in keywords) {  
    // The "in" keyword will also include properties that aren't keyword.  
    if (keywords.hasOwnProperty(keyword)) {  
      adGroup.createKeyword(keyword);  
    }  
  }  
}

**Setting ad parameters**

Now that we've set up keywords, we can set the ad parameter values.

function updateAdParameterValues(adGroup, earthquakeData) {  
  // Note that due to the way previewing works, this script won't  
  // update ad params for newly created keywords.  
  var keywordIter = adGroup.keywords().get();  
  while (keywordIter.hasNext()) {  
    var keyword = keywordIter.next();  
    keyword.setAdParam(1, earthquakeData[keyword.getText()].count);  
    keyword.setAdParam(2, earthquakeData[keyword.getText()].lowestMagnitude);  
  }  
}

The keyword object allows us to update the ad parameter for this keyword by specifying the index and value. In our example ad, we're setting the number of earthquakes as parameter 1 and the lowest magnitude seen as parameter 2.

As mentioned previously, preview mode doesn't create any keywords, so you have to run this script for real to update the ad parameter values.

You can also get a list of existing ad parameters via the .adParams() method on AdWordsApp, Ad Groups and Keywords objects. You can directly call.setInsertionText(s) on these objects. We also provide .getAdGroup() and .getKeyword() to provide context when updating a list of ad parameters, as well as .remove() to delete it.

Setup

* Create a new ad group with a text ad. Use the description lines below which have markers for the keyword and ad params to be inserted.
  + Description1: {param1:Many} {KeyWord:earthquakes} {param2:of large} magnitude
  + Description2: and up. Evacuation checklists here.

Use any headline or destination URL you like. Don't worry about creating keywords, this script will dynamically update them based on live data.

* Create a new AdWords script with the source code below.
* At the top of the script replace the value of the AD\_GROUP\_NAME variable with the name of the ad group you created.
* Due to the way **Preview** mode works, this script will not update the ad parameters of newly added keywords. Click **Run script now** to run the script and see the results. If you create a new ad group just for testing this script, you shouldn't have to worry about making undesired changes to your account.

[Creating an AdWords script](https://developers.google.com/adwords/scripts/docs/solutions/parameterizer)

Source code

// Change this to match the name of your Ad Group you'll be using with Ad Parameters.  
var AD\_GROUP\_NAME = 'Earthquakes';  
  
/\*\*  
 \* For this example, we expect that the Ad Group named above will have  
 \* an Ad set up with Ad Parameters.  Use these descriptions:  
 \* "{param1:Many} {KeyWord:earthquakes} {param2:of large} magnitude"  
 \* "and up. Evacuation checklists here."  
 \*  
 \* The script will automatically add keywords.  
 \*/  
  
function main() {  
  var adGroup = getAdGroup(AD\_GROUP\_NAME);  
  // If we couldn't find the Ad Group, log an error and halt.  
  if (adGroup) {  
    var earthquakeData = getEarthquakeData();  
    updateEarthquakeData(adGroup, earthquakeData);  
  } else {  
    Logger.log('Could not locate AdGroup with name ' + AD\_GROUP\_NAME);  
  }  
}  
  
function getAdGroup(name) {  
  var adGroupIterator = AdWordsApp.adGroups()  
      .withCondition('Name = "' + name + '"')  
      .withLimit(1)  
      .get();  
  if (adGroupIterator.hasNext()) {  
    return adGroupIterator.next();  
  }  
}  
  
function updateEarthquakeData(adGroup, earthquakeData) {  
  updateKeywords(adGroup, earthquakeData);  
  updateAdParameterValues(adGroup, earthquakeData);  
}  
  
// Removes keywords that are no longer relevant and adds new keywords.  
function updateKeywords(adGroup, earthquakeData) {  
  var keywordsToAdd = {};  
  // Make a copy of all expected keywords so we can later add ones that don't exist in this Ad Group.  
  for (var keyword in earthquakeData) {  
    keywordsToAdd[keyword] = keyword;  
  }  
  var keywordIter = adGroup.keywords().get();  
  while (keywordIter.hasNext()) {  
    var keywordInAccount = keywordIter.next();  
    // If the keyword from the adgroup isn't in the set of ones we want, remove it.  
    if (!(keywordInAccount.getText() in keywordsToAdd)) {  
      keywordInAccount.remove();  
    }  
    // Since we do want this keyword and it exists, remove it from the list so we  
    // add only the ones that don't exist.  
    delete keywordsToAdd[keywordInAccount.getText()];  
  }  
  // Now create all the keywords that didn't already exist.  
  for (var keywordToAdd in keywordsToAdd) {  
    // "in" may return properties other than the keywords we've provided.  
    if (keywordsToAdd.hasOwnProperty(keywordToAdd)) {  
      adGroup.createKeyword(keywordToAdd);  
    }  
  }  
}  
  
function updateAdParameterValues(adGroup, earthquakeData) {  
  // Note that due to the way previewing works, this script won't  
  // update ad params for newly created keywords.  
  var keywordIter = adGroup.keywords().get();  
  while (keywordIter.hasNext()) {  
    var keyword = keywordIter.next();  
    keyword.setAdParam(1, earthquakeData[keyword.getText()].count);  
    keyword.setAdParam(2, earthquakeData[keyword.getText()].lowestMagnitude);  
  }  
}  
  
function getEarthquakeData() {  
  // We're looking at the past 30 days of significant quake activity.  
  var url = 'http://earthquake.usgs.gov/earthquakes/feed/geojson/significant/month';  
  var json = Utilities.jsonParse(UrlFetchApp.fetch(url).getContentText());  
  var earthquakes = {};  
  // Iterate over all earthquakes in the response json.  
  json.features.map(function(quake) {  
    var placeName = extractPlaceName(quake.properties.place);  
    // If we haven't seen this place yet, add an empty record.  
    if (!earthquakes[placeName]) {  
      earthquakes[placeName] = {count: 0, lowestMagnitude: 10.0};  
    }  
    // Increment the count  
    earthquakes[placeName].count += 1;  
    // Update lowest magnitude seen.  
    earthquakes[placeName].lowestMagnitude =  
  Math.min(earthquakes[placeName].lowestMagnitude, quake.properties.mag);  
  });  
  return earthquakes;  
}  
  
// Takes a place like '2km NE of Yorba Linda, California' and returns  
// 'California earthquake'  
function extractPlaceName(place) {  
  return /(,)?.\*?([a-zA-Z \-]+)$/.exec(place)[2].trim() + ' earthquake';  
}