

Show Random Pages or Questions, Least Fill (Quotas)

Scripting Solutions

Additional scripting solutions will be added in the future. Please reach out to Alchemer with comments and suggestions on solutions you'd like to see via the link [here](#).

Scripting and Other Custom Solutions

We're always happy to help you debug any documented script that is used as is. That said, we do not have the resources to write scripts on demand or to debug a customized script.

If you have customization ideas that you haven't figured out how to tackle, we're happy to be a sounding board for Alchemer features and functionality ideas that might meet your needs. Beyond this, check out our [Professional Services](#); these folks have the scripting chops to help you to achieve what you are looking for!

Goal

Show a limited number of random pages (or questions). Use a quota-based Least Fill strategy to ensure pages (or questions) are presented as evenly as possible.

If you are not using the options below it will be easier to use the [Basic Solution](#).

If you have a license that provides Lua script and will be limiting with 10 or fewer Checkbox options and collecting 1,000 or fewer responses, it will be easier to use the [Least Fill \(Query\) Solution](#) (link coming soon).

Limit based on selections made in a Checkbox question Option Show a limited number of random follow up pages (or questions) based on selections made in a Checkbox question (which could be a list of Concepts or Brands). A **Prioritization Option** ensures the respondent will *always* be asked about a specific selection if checked (for example: the Brand you're researching).

Respondent Condition Option Ensure pages (or questions) are presented evenly to respondents who fall into one or more Conditions (such as Gender, Race, or Age Bracket). Note that this is limited as Conditions can quickly generate a large number of Quotas.

See also: [Show Random Pages or Questions](#)

Effort:  (mostly adding quotas and page logic)

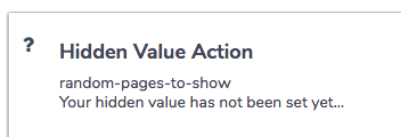
Solution

This solution fills a **Hidden Value Action** with a list of random name you define using quotas. Later pages or questions use this list to determine if they should be shown or hidden based on whether it contains a specific name or not.

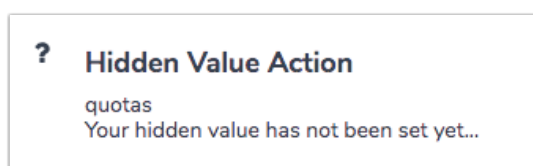
Step 1: Add Two Hidden Value Actions

(1) Add a new page.

(2) Add a **Hidden Value Action** with the title **random-pages-to-show**. The script will set this to a list of randomly selected quota names.



(3) Add a second Hidden Value Action with the title **quotas**. The script will set this to the quota counts that were found at the time this respondent was taking the survey. This is useful if you want to understand why a response was presented specific pages.



Step 2: Add Quotas





If one is using the **Respondent Condition Option** skip this step and see directions for this option at the end of the article.

The example below use the names A, B, C, D. If one is using other names, ensure they are short single words that are not a substring of one another (ie: "SPS" and "XSPS" aren't allowed).

The Quotas **must** follow the naming convention of *fill-Name* (ie, they must start with "fill-").

Set each quota to check if **random-pages-to-show** "contains" the Name. And set each quota to **Continue** collecting responses on the **Complete Actions** tab.

Logic Based Quotas

Name ▾	Type
 fill-A	Logic
 fill-B	Logic
 fill-C	Logic
 fill-D	Logic

Setup

Name

Edit Description

Logic
 Remove All Logic

random-pages-to-show ▾ contains ▾ A

[+ Add Condition](#)



Edit Logic Quota

SETUP COMPLETE ACTIONS

Complete Actions

When a Quota has Been Met

Stop collection of responses & show a

Stop collection of responses & redirect

Continue collecting responses

Step 3: Add a Webhook Action to get the quotas

Add a **Webhook Action** to the top of the page added in Step 1 to get the quotas:

1. Choose **GET Method**
2. Enter the URL below for your data center (US, CA, or EU), updating the highlighted sections with the survey ID and API Token/Secret

US - https://api.alchemer.com/v5/survey/1234567/quotas?api_token=***&api_token_secret=***

CA - https://api.alchemer-ca.com/v5/survey/1234567/quotas?api_token=***&api_token_secret=***

EU - https://api.alchemer.eu/v5/survey/1234567/quotas?api_token=***&api_token_secret=***

3. Choose **Display it** at the bottom of page for what to do with the returned data.

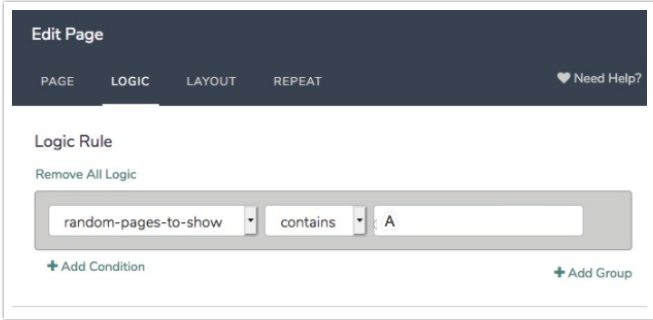
Step 4: Add a Javascript Action

1. Add the **Javascript** at the end of this article to the bottom of the page added in Step 1.
2. Set the highlighted values

Step 5: Add random pages (or questions)

Add the pages (or questions) to be randomly shown.

Set the Display Logic for each to only show if the **random-pages-to-show** Hidden Value Action added in Step 1 *contains* one of the quota Names. The text must match exactly, including case.



The screenshot shows the 'Edit Page' interface with a 'Logic Rule' section. The rule is configured as follows:

- Logic Rule: random-pages-to-show
- Operator: contains
- Value: A

Buttons for '+ Add Condition' and '+ Add Group' are visible below the rule configuration.

Step 6: Test

There's a lot going on here. Users do this in Preview by selecting **Fire Actions**. In Chrome users **right-click** > choose **Inspect** > click the **Console** tab. In the test below Alchemer has both Gender Condition and Aware-of questions. After answering those questions and clicking **Next**, the Javascript page finds the quotas based on the **Condition** and **Aware of** selections. It then determines which Concepts to show using the **Prioritization** of any Brand with an asterisk at the end of it's reporting value (Southwest in this example).

The example below chooses two of four brands and runs a Least Fill *separately* for Male and Female respondents.

Gender

1. Gender *

Male

Female

2. I'm aware of these airlines *

United

Southwest

Delta

Spirit

Calculate which Concepts to show this respondent

3. concepts-to-show

```

getConceptQuotas() -
  > Array(8) [ (-), (-), (-), (-), (-), (-), (-), (-) ]
pre
getConceptQuotas(), conditionsString = Female pre
getConceptQuotas(), awareOfConcepts = pre
  > Array(3) [ "A", "D", "B*" ] pr
conceptQuotas = pr
  fill-Female-A - 0 pr
  fill-Female-B - -1 pr
  fill-Female-D - 0 pr
toShow = pr
  B
  D

```

Above we can see the Console tab displays the reasoning of the Javascript code. In this case the Condition is **Female** and the **Aware of Concepts** are **A, B*, D**. Users see that three Quotas were evaluated, but since there are no responses yet all quotas are Zero.

However, because **Concept B** has an *asterisk* in the Aware of Reporting Value, it is **Prioritized**. To prioritize within a Least Fill, the code sets the Brand response count to -1, ensuring it moves to the start of the Least Fill list. The remaining Concepts are randomly chosen based on the lowest response counts. In the above example, this results in brand **B** and **D** being selected.

To see the quota counts increase, use the **Share tab** and create **Complete responses**. To quickly test quotas, add Skip/DQ logic at the end of the Javascript page to send respondents to the Thank You page to not spend time answering the Concept/Brand pages. There may be a few second delay after completing a response before the Javascript shows the updated quota counts in the Console window.

Step 6: Hide the page from Step 1

When testing is complete hide the page added in Step 1:

1. Enter **sg-hide** for the page's **Layout tab > CSS Class Name**. This class will hide the page.
2. Uncomment the line at the end of the Javascript that clicks the Next button by removing the leading backslashes highlighted below:

```
// $("#sg_NextButton").click();
```

3. Note: Do **not** set the page as Auto-Submit, this does not allow the **Javascript** to function.

Limit based on selections made in a Checkbox question Option (otherwise skip this step)

1. Add a Brand Awareness Checkbox Question to the survey with an option for each Brand.
2. The Reporting Values must match the names used for the Quotas later. Set Reporting Values to Custom and edit them to ensure they are short, single words and that none is a substring of another (ie: "SPS" and "XSPS" aren't allowed). Pro tip: simply set the Reporting Values to letters of the alphabet.
3. **Prioritization Option:** Add an *asterisk* at the end of the Reporting Values (ex: B* below) to ensure the Brand is always presented if the respondent is aware of it.

The screenshot shows a survey question editor for a checkbox question. The question text is "What question do you want to ask?" with the example "I'm aware of these airlines". The "Require this question" checkbox is checked. Under "Multiple Choice Options", there are four options: "United", "Southwest", "Delta", and "Spirit". To the right, under "REPORTING VALUE", the corresponding values are "A", "B*", "C", and "D".

In the Javascript code, uncomment the **AWARE_OF_CONCEPTS** section and set the merge code question ID to the Aware of question (to uncomment, remove `//` from the start of the line of code).

```
document.addEventListener("DOMContentLoaded", function() {
    // ... (previous code) ...
    // Uncomment the following line to save what the quotas looked like to this response (so we can audit that the
    const CURRENT_QUOTAS_QID = 130
    // Array of the Reporting Values of the "Aware of brands" checkbox
    let AWARE_OF_CONCEPTS = []
    // AWARE_OF_CONCEPTS = `[question("value"), id="134"]`.split(',')
    // Quota names are in the form 'fill-[CONDITION-[CONDITION-]]CONCEPT'
    // where the CONDITIONS are optional.
    let CONDITIONS = []
    //CONDITIONS = [
    //  `[question("value"), id="134"]`,
    // ]
})
```

Remove slashes

Set to QID of checkbox question from earlier page

Respondent Condition Option (otherwise skip this step)

1. Add one or more Radio Button questions, like Gender.
2. Set the Reporting Values to Custom and edit them to be short single words .

Question Type

Radio Buttons

What question do you want to ask?

Gender

Require this question

Multiple Choice Options Common As

OPTION	REPORTING VALUE
Male	Male
Female	Female

Create a Quota for each Name / Concept combination. Each quota checks one Condition and Name to see if it's contained in the **random-pages-to-show** added in Step 1. The Quotas follow the naming format of **fill-Condition-Name**. If one is using two conditions, use the form **fill-Condition1-Condition2-Name**.

Logic Based Quotas

Name ▾

- fill-Female-A
- fill-Female-B
- fill-Female-C
- fill-Female-D
- fill-Male-A
- fill-Male-B
- fill-Male-C
- fill-Male-D

Setup

Name Limit

Edit Description

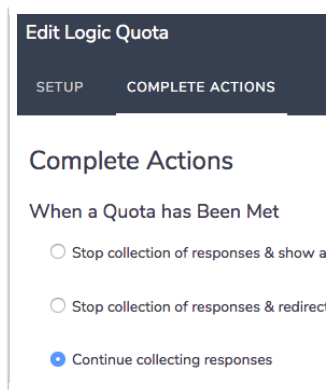
Logic

Remove All Logic

Q 1. Gender Male Female

and

Q 3. concepts-to-show



In the Javascript code, uncomment the **CONDITIONS** array. Add one or more Condition question merge codes to the array for the earlier Condition questions. If there is more than one Condition, the order needs to match the order of the Conditions listed in the Quota names.

For example: the Quota **fill-Male-Black-A** would need the *CONDITIONS array* to have the Gender merge code first and the Race merge code second:

```
CONDITIONS = [  
  `[question("value"), id="2"]` , // gender question merge code  
  `[question("value"), id="3"]` , // race question merge code  
]
```

Code to add to Javascript Action

```
/* Alchemer v02  
  
Determine concepts to show based on Least Fill. Uses quotas returned from  
the Webhook Action to get current count of completed surveys.  
  
Different quotas are applied for the respondent's Condition. The Condition is  
how the respondent answered a question, like Gender.  
  
Documentation and updates: https://help.alchemer.com/help/least-fill-quotas  
*/  
  
document.addEventListener("DOMContentLoaded", function() {  
  
  // how many pages (questions) to show  
  const NUM_TO_SHOW = 2  
  
  // QID to save the vertical-bar-separated list of Names to show  
  const SAVE_TO_QID = 129  
  
  // QID to save what the quotas looked like to this response (so we can audit that that the javascript choice made sense)  
  const CURRENT_QUOTAS_QID = 130  
  
  // Array of the Reporting Values of the "Aware of Brands" checkbox  
  let AWARE_OF_CONCEPTS = []  
  
  // Checkbox is on an earlier page  
  AWARE_OF_CONCEPTS = `[question("value"), id="134"]`.split(',')
```



```

// Checkbox is on this page and is based on logic from questions earlier in the survey
// Contains a pre-checked 'do not delete' element to ensure it isn't confused for the default empty array
// AWARE_OF_CONCEPTS = 134

// Quota names are in the form 'fill-[CONDITION-[CONDITION-]]CONCEPT'
// where the CONDITIONS are optional.
let CONDITIONS = []
//CONDITIONS = [
// ` [question("value"), id="2"]`,
//]

// *****
// * no changes needed below *
// *****

const LOG = true

/**
 * Helper to display error dialog and throw new Error
 */
const assert = (bool, msg) => {
  if (!bool)
    throw new Error(msg)
}

/**
 * Helper: Get a SurveyGizmo element on the page.
 * Ex: In survey 1234567 on page ID 12 the call getSgId(123, "element")
 * returns HTML element for "sgE-1234567-12-123-element"
 */
function getElemByQid(qid, oid = "element") {
  let surveyInfo = SGAPI.surveyData[Object.keys(SGAPI.surveyData)[0]]
  let id = "sgE-" + surveyInfo.id + "-" + surveyInfo.currentpage + "-" + qid + "-" + oid
  let elem = document.getElementById(id)
  assert(elem, "Javascript error: can't find element with id = " + id)
  return elem
}

/**
 * Get the reporting values for the CHECKED options of
 * a radio button or checkbox QID.
 *
 * qid (int / string) question ID
 * return (array of string) array of reporting values
 */
const getCheckedByQid = (qid) => {
  const checkedElems = getElemByQid(qid, "box").querySelectorAll('.sg-question-options input:checked')
  if (LOG) console.log("checkedElems = ", checkedElems)
  const checkedOptionIDs = [...checkedElems].map(elem => elem.value)
  // get object that maps optionID to reporting value, ex: { "10014": "reporting value1", "10015": "value2"}
  const optionsObj = SGAPI.survey.surveyObject.questions[qid].options
  // map the object array to array of just the reporting values
  const checkedReportingValues = checkedOptionIDs.map(optionId => optionsObj[optionId].value )
  if (LOG) console.log("checkedReportingValues = ", checkedReportingValues)
  return checkedReportingValues
}

/**
 * Sort two strings in alpha order
 */

```

```

/
const alphaSortFn = (a, b) => a.localeCompare(b)

/**
 * Parse all quotas from webhook call.
 *
 * return {quota array} quota object format:
 *   {
 *     "id": "2600",
 *     "name": "fill-Male-A",
 *     "description": "",
 *     "responses": "0",
 *     "limit": "100",
 *     "distributed": "false"
 *   }
 */
const getAllQuotas = () => {

  // get JSON from webhook call
  const quotasJSON = document.querySelector('.sg-http-content').innerText

  // test
  if (!quotasJSON)
    throw new Error("getAllQuotas(), no .sg-http-content")

  // parse webhook response
  const parsed = JSON.parse(quotasJSON)

  // test
  if (!parsed.result_ok === "ok")
    throw new Error("getAllQuotas(), result not ok: " + parsed)
  if (LOG) console.log("getAllQuotas() = ", parsed.quotas)
  if (!parsed.quotas)
    throw new Error("no quotas")

  return parsed.quotas
}

/**
 * Shuffle / randomize an array in place
 *
 * array {array} - array is mutated
 * return {array} - shuffled array
 */
function shuffle(array) {

  // Knuth shuffle (https://stackoverflow.com/questions/2450954/how-to-randomize-shuffle-a-javascript-array)

  var currentIndex = array.length, temporaryValue, randomIndex;

  // While there remain elements to shuffle...
  while (0 !== currentIndex) {

    // Pick a remaining element...
    randomIndex = Math.floor(Math.random() * currentIndex);

```

```

    randomIndex = Math.floor(Math.random() * currentIndex);
    currentIndex -= 1;

    // And swap it with the current element.
    temporaryValue = array[currentIndex];
    array[currentIndex] = array[randomIndex];
    array[randomIndex] = temporaryValue;
}

return array;
}

/**
 * Parse the Concept name from a quota
 *
 * @param {object} quota {quota object}
 * @return {string} - The Concept name from the end of the quota.name
 */
const parseConceptFromQuota = quota => {
    const concept = quota.name.substring(quota.name.lastIndexOf('-') + 1)
    if (!concept) {
        console.log("parseConceptFromQuota() couldn't parse quota: ", quota)
        throw new Error("parseConceptFromQuota() couldn't parse quota.name: " + quota.name)
    }
    return concept
}

/**
 * Get the specific quotas that apply to respondent's Condition.
 *
 * @param {array of string} conditions Optional Condition(s) selected by respondent. Quotas are
 * named 'fill-CONDITION-[CONDITION-]CONCEPT'.
 * @param {array of string} awareOfConcepts Optional Concepts selected in previous checkbox question
 * of which Concepts/Brands respondent is aware of
 * @return {quota array} quotas that match the respondent's Condition(s)
 */
const getConceptQuotas = (conditions, awareOfConcepts) => {

    let conceptQuotas = getAllQuotas().filter(quota => quota.name.startsWith(`fill-`))

    // filter Concepts if using Condition(s)
    if (conditions.length) {
        // Create string of Condition Reporting Values, ex: "Female-Black" or empty string if Conditions aren't being used
        let conditionsString = conditions.join('-')
        if (LOG) console.log("getConceptQuotas(), conditionsString = ", conditionsString)
        conceptQuotas = conceptQuotas.filter(quota => quota.name.startsWith(`fill-${conditionsString}`))
    }

    // filter Concepts if using an "Aware of" checkbox
    if (awareOfConcepts.length) {
        if (LOG) console.log("getConceptQuotas(), awareOfConcepts = ", awareOfConcepts)
        conceptQuotas = conceptQuotas.filter(quota => {
            return awareOfConcepts.find(s => s.replace('*', '') == parseConceptFromQuota(quota))
        })
    }

    // Prioritize Concepts if they have an asterisk at the end of their Aware of Reporting Value.
    // TODO: This is a bit of a hack, but it works. It will prioritize Concepts that have an asterisk at the end of their

```

```

// To do this set the quota response count to "-1" so it'll be sorted to the front.
conceptQuotas = conceptQuotas.map(quota => {
  if (awareOfConcepts.includes(parseConceptFromQuota(quota) + '*'))
    quota.responses = "-1"
  return quota
})
}

// test
if (!conceptQuotas)
  throw new Error ("no concept fill quotas ", (conditionsString ? `for condition(s): ${conditionsString}` : ""))

// log
if (LOG) {
  console.log("conceptQuotas = ")
  console.log(
    conceptQuotas.map((quota) => ` ${quota.name} - ${quota.responses}` )
    .sort(alphaSortFn)
    .join("\n"))
}

return conceptQuotas
}

/**
 * Get vertical bar-separated list of Concepts to show (Ex: 'B|D')
 *
 * conceptQuotas_leastFirst {quota array} array is sorted with Least First, so pull from front of array
 * numToShow {int} how many concepts to show
 * return {string} with vertical bar-separated list of Concept names to show
 */
const getConceptsToShow = (conceptQuotas_leastFirst, numToShow) => {
  let toShow = []
  for (let i = 0; i < numToShow && i < conceptQuotas_leastFirst.length; i++) {
    // Assumes the quota names are in the form 'fill-CONDITION-CONCEPT'.
    toShow.push(parseConceptFromQuota(conceptQuotas_leastFirst[i]))
  }
  toShow = toShow.sort(alphaSortFn)
  if (LOG) console.log(`toShow = \n ${toShow.join("\n ")}`)
  return toShow.join('|')
}

/**
 * main()
 */
try {

  // AWARE_OF_CONCEPTS is one of:
  // - empty array
  // - array of string from a checkbox on a previous
  // - QID for a checkbox on this page on this page pre-checked and using display
  // logic based on previous questions
  let awareOfConcepts = AWARE_OF_CONCEPTS
  if (!Array.isArray(AWARE_OF_CONCEPTS)) {
    awareOfConcepts = getCheckedByQid(AWARE_OF_CONCEPTS)
  }
}

```

```

}

// get the quotas specific to respondent's Conditions and Aware-of choices
let conceptQuotas = getConceptQuotas(CONDITIONS, awareOfConcepts)

// randomize, ensure randomized Concepts selected even if all have the same number of Completes at this time
conceptQuotas = shuffle(conceptQuotas)

// sort based on number of responses received for the Concept
conceptQuotas_leastFirst = conceptQuotas.sort((a,b) => parseInt(a.responses) - parseInt(b.responses) )

// save which concepts to show
getElemByQid(SAVE_TO_QID).value = getConceptsToShow(conceptQuotas_leastFirst, NUM_TO_SHOW)

// save the current quotas so we can validate that the code was making the right choices at the time of this response
if (LOG) console.log(">>> conceptQuotas_leastFirst = ", conceptQuotas_leastFirst)
getElemByQid(CURRENT_QUOTAS_QID).value = conceptQuotas_leastFirst.map((quota) => ` ${quota.name} - ${quota.responses}`)
    .sort(alphaSortFn)
    .join("\n")
if (LOG) console.log(">>> getElemByQid(CURRENT_QUOTAS_QID).value = ", getElemByQid(CURRENT_QUOTAS_QID).value)

}
catch (err) {
  console.error("ERROR,message ", err.message)
  console.error("ERROR ", err)
  alert("Javascript error in catch: " + err.message)
}

// move to next page
$("#sg_NextButton").click();
})

```

Related Articles