Category Pages Technical Guide

Sitecore Discover
## Table of Contents

Chapter 1 Introduction ........................................................................................................... 3  
Chapter 2 Category Page Template .......................................................................................... 5  
Chapter 3 Category Feed ......................................................................................................... 7  
Chapter 4 Category URL Design ............................................................................................ 10  
  4.1 Refining results within category pages ............................................................................. 11  
Chapter 5 Category URL Configuration .................................................................................. 12  
  5.1 Using attribute values only .............................................................................................. 13  
  5.2 Hierarchical attribute values ............................................................................................ 14  
  5.3 Using attribute name and value ....................................................................................... 15  
  5.4 Using a unique string literal ............................................................................................. 16  
  5.5 Using soft filters ............................................................................................................. 17  
Chapter 6 Attribute Names and Values .................................................................................. 18  
  6.1 Attribute names and values when used as part of URL paths (Hard Filters) ................. 19  
  6.2 Attribute names and values when used as query parameters (Soft Filter) .................... 21  
  6.3 Sitecore Discover reserved words ................................................................................... 23  
Chapter 7 Distinguishing URLs for Sitecore Discover .......................................................... 24  
  7.1 URL Path ...................................................................................................................... 25  
  7.2 Query parameter ............................................................................................................ 26  
  7.3 Subdomain .................................................................................................................... 27  
Chapter 8 CDN/Reverse Proxy Configuration ...................................................................... 28  
  8.1 Origin Host .................................................................................................................... 29  
  8.2 Configuration Options ................................................................................................. 30  
  8.3 CDN/Reverse Proxy Cache Control .............................................................................. 32  
  8.4 CDN/Reverse Proxy Cookies Passthrough .................................................................... 33  
  8.5 Ramping up Traffic to Hosted Sitecore Discover Service ............................................ 34  
  8.6 CDN Pseudo Logic ....................................................................................................... 36  
  8.7 CDN Example: CloudFront ............................................................................................ 37  
  8.8 CDN Example: Varnish for Magento Cloud ................................................................. 43  
Chapter 9 B2B / Resellers / Stores Considerations ................................................................. 45  
  9.1 Hosted By Sitecore Discover / JS Implementation ....................................................... 46  
  9.2 API Implementation ..................................................................................................... 47  
Chapter 10 SEO Considerations ............................................................................................. 48  
Chapter 11 Category Pages Integration Options ................................................................. 49  
  11.1 Hosted by Sitecore Discover ....................................................................................... 50  
  11.2 API Integration ........................................................................................................... 51  
Chapter 12 Notes .................................................................................................................. 52  
Chapter 13 What to do when a URI is not identified by Sitecore Discover ............................... 53  
Chapter 14 Example API Integration to manage your own landing pages .............................. 54
Chapter 1 Introduction

This document describes the technical details of Sitecore Discover Category pages service. Category pages allow you to serve category pages (also called product listing pages), brand pages, special occasion pages, any form of landing pages, etc.

The following figure shows the request flow from the browser for category pages when a customer has a CDN and can be programmed to map the browser requests to Sitecore Discover service. More detailed sequence diagrams are provided later in the document. See Web Sequence Diagrams.

Customer CDN (ex: Akamai)
Customer's pass through, no caching, can have fallback to call customer's e-Commerce server. Adding any kind of caching can add significant latency to serving the page.

Customer Reverse Proxy Server (ex: NginX)
Customer's reverse proxy service is an alternative to customer's CDN on how the category page requests can be proxied to the Sitecore Discover service. It should be setup as a pass through, no caching, and can have fallback to call customer's e-Commerce server. Adding any kind of caching can add significant latency to serving the page.

Customer eCommerce Service
Customer's main e-commerce servers

Sitecore Discover Web Service
Sitecore Discover pass through, and redirects to fallback as required
Sitecore Discover

Sitecore Discover Page Service
Main Sitecore Discover service to serve personalized category pages

Sitecore Discover Fallback Cache
Category pages generated offline used as fallback and for bots. These pages are not personalized.
The following figure shows the flow of generating category page cache. The cache may be customized for segments of users but is not personalized to individual users.
Customer provides a URL to an empty category page template that can be retrieved by Sitecore Discover service at any time. The template should include:

- Header and footer of the page
- Sitecore Discover beacon (init.js) at the top of the HTML header
- JavaScript code for any quick view and add-to-cart functionality.
- JavaScript code to update any user-personalized information that is included in the template (e.g., cart status, user login, user-specific banner, etc.). See image of possible dynamic portions of typical header that must be rendered via your own JavaScript.

- RFKID placeholder tag in the body of the page. This tag will be replaced by Sitecore Discover with product results, filters/facets, sort options, pagination, etc.
  - `<div data-rfkid="rfkid_container"></div>`
- RFKID placeholder tag in the HTML header for Sitecore Discover to include any SEO/meta information for the category. Corresponding data to include must be provided as part of the category feed
  - `<meta data-rfkid="rfkid_meta"/>

The following meta elements must not be included in the template. These elements should be configured as part of the page SEO widget within the Sitecore Discover Customer Engagement Console:

- `<title>...</title>`
- `<meta name="description" content="...">`  
- `<link rel="canonical" href="...">`

Additionally, if you use Open Graph protocol, then talk to Sitecore Discover Support to help you setup these meta tags. Please do not include the following ogp tags in your template:

- `<meta property="og:title" content="...">`
- `<meta property="og:description" content="...">`
- `<meta property="og:url" content="...">`
If there are other meta information on the page that is specific to each category, please talk to Sitecore Support to help set up these additional attributes. For example, you may have a separate image to represent each of the categories (og:image property).

Sitecore Discover will cache the template. Note that Sitecore Discover does not refresh templates automatically when your site changes to avoid any unexpected issues. If you change the template of the page on your website, you must return to CEC and refresh the template manually.

The suggested category page template URL is: <customer-domain.com>/rfk_category_template.html
Chapter 3  **Category Feed**

Customers provide category feed with the following information. The category feed is also used to populate the category pages to be SEO friendly when Sitecore Discover is hosting the pages. Please refer to Sitecore Discover Data Feed Specs for exact fields to provide in the feed.

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccid</td>
<td>Y</td>
<td>Customer Category ID (id or name should also be included in the product feed)</td>
</tr>
<tr>
<td>parent_ccid</td>
<td>N</td>
<td>Category's Parent CCID. This is used to determine hierarchy information. The topmost category will have empty value. <strong>Note</strong>: Sitecore Discover does not support a category to have multiple parents.</td>
</tr>
<tr>
<td>name</td>
<td>Y</td>
<td>Category name (id or name should also be included in the product feed)</td>
</tr>
<tr>
<td>seo_name</td>
<td>N</td>
<td>Category URL SEO name (SEO friendly), name to use when synthesizing the category URL. <strong>Note</strong>: This field can also be used to map the URL to a category by extracting a portion of the URL to match the seo_name. In this case, the names must uniquely identify a category.</td>
</tr>
<tr>
<td>title</td>
<td>N</td>
<td>Category title to display in Category pages (SEO friendly). If not provided, name is used for titles.</td>
</tr>
<tr>
<td>desc</td>
<td>N</td>
<td>Category description to display below the title (SEO friendly, may be text or simple html snippet).</td>
</tr>
<tr>
<td>meta_title</td>
<td>N</td>
<td>Category page meta title (if different from Category title) to include as part of the meta-data on the page.</td>
</tr>
<tr>
<td>meta_desc</td>
<td>N</td>
<td>Category meta-description to include in page meta-data. If not provided, desc is used.</td>
</tr>
<tr>
<td>meta_keywords</td>
<td>N</td>
<td>Category meta keywords to include in page meta data.</td>
</tr>
<tr>
<td>Field</td>
<td>Required</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>url_path</td>
<td>Y</td>
<td>Category page URL path. SEO friendly, full URL of the page that represents this category. This should not contain any query parameters. Any query parameters in this field will be ignored. If not provided it may be generated by Sitecore Discover using <strong>seo_name</strong> (see below).</td>
</tr>
<tr>
<td>secondary_url_paths</td>
<td>N</td>
<td>Additional list of URL paths.</td>
</tr>
<tr>
<td>breadcrumbs</td>
<td>N</td>
<td>Category breadcrumb information, i.e path category-&gt;sub-category-&gt;... if not provided it will use category hierarchy information to generate the path.</td>
</tr>
<tr>
<td>image_url</td>
<td>N</td>
<td>Category image/thumbnail URL</td>
</tr>
<tr>
<td>banner_url</td>
<td>N</td>
<td>Category banner URL, if you want to display a banner in the category pages. If you want to have dynamic banners or manage banners through CEC, please use Content Widgets in CEC.</td>
</tr>
<tr>
<td>is_active</td>
<td>N</td>
<td>An optional field to say if the category is active or not. Default is active.</td>
</tr>
<tr>
<td>is_primary</td>
<td>N</td>
<td>An optional flag to represent if this category represents a primary or true product hierarchy independent of other ways you might organize a set of products. Default is true.</td>
</tr>
<tr>
<td>cross_category_ccids</td>
<td>N</td>
<td>A list of customer category ids that are complementary matching product categories. This is useful when building recommendations of products that are complementary to a given product. Complete The Look is a typical recommendation that is powered by the cross categories.</td>
</tr>
</tbody>
</table>
For example: If the category is shirt, cross-categories could be pants, earrings, shoes, etc.

Customer may also want to specify rules for displaying categories or category hierarchy in category navigation and provide additional information specific to the customer.
Chapter 4  Category URL Design

There are many ways to specify categories in URLs. A common method of creating category page URLs is to add the category name to the base URL. For example, the category page for category=shirts could be:

www.domain.com/shirts
www.domain.com/category/shirts

Further, Category pages are not restricted to categories only. You can create a page that is restricted on multiple attributes. For example, you can have a page that shows red shirts by specifying color=red and category=shirts. You must design the URLs carefully to build such flexibility while being SEO friendly. Some potential suggestions to consider for category=shirts and color=red are:

www.domain.com/shirts/red
www.domain.com/category/shirts/color/red

Note that all the above URLs have a pattern. The Sitecore Discover system can be configured to support many such patterns. You can also define a custom URL and create filters that apply to it in the configuration. For example:

www.domain.com/red-shirts (filter: category=shirts and color=red)
www.domain.com/christmas-sale-for-red-shirts (filter: tag=christmas and category=shirts,sale and color=red)

Note: Google recommends that you use hyphens (-) instead of underscores (_) in your URLs.
4.1 Refining results within category pages

The above examples restrict results so a user cannot override the given restriction (also referred as Hard Filters). However, you can also augment the URLs to define a filter that can be unset (also referred as Soft Filters), to allow users to see broader results if desired. For example, if the intent was to restrict the search results to category=shirts and apply the color=red filter to show only red shirts, but you also want to allow the user to unset the color filter so he/she could see all shirts, you could set the filter as a parameter in the URL as follows:

www.domain.com/shirts?color=red

**Note:** Category Page URLs in the category feed should not contain any query parameters. For example, a URL design such as www.domain.com?category=shirts is not allowed.
Chapter 5  Category URL Configuration

The following are common ways to configure category URLs:

- Using attribute values only
- Hierarchical attribute values
- Using attribute name and value
- Using a unique string literal
- Using soft filters
5.1 Using attribute values only

www.domain.com/attributeValue
www.domain.com/attributeValue1/attributeValue2/...

Example:

www.domain.com/shirts/red
AttributeName1 = category, attributeValue1 = shirts
AttributeName2 = color, attributeValue2 = red

In the above example, only attribute values are part of the URLs, attribute names are not part of the URL.

Note: You cannot use above formats if there is a common attribute value for two different attributes for which you want to generate landing pages. For example, if you have category with a value of gucci, and you have a brand named Gucci, then you cannot generate both category and brand pages with the above format.
5.2 Hierarchical attribute values

attributeValue can be a simple string, or it can be hierarchical. Sitecore Discover can optionally generate URLs to use hierarchical paths with attributes that have a notion of hierarchy (like category).

For example, if you have the following hierarchical categories:

<table>
<thead>
<tr>
<th>Clothing</th>
<th>Mens</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clothing</td>
<td>Womens</td>
</tr>
</tbody>
</table>

and the hierarchy separator is `-`, the paths generated would be:

- www.domain.com/clothing-mens
- www.domain.com/clothing-womens
5.3 Using attribute name and value

www.domain.com/{attributeName}/{attributeValue}/…

Example

www.domain.com/category/shirts/color/red
AttributeName1 = category, attributeValue1 = shirts
AttributeName2 = color, attributeValue2 = red

In the above example, both attribute names and values are part of the URL.

Note: You cannot use the above format if the attribute name and one of the attribute values is the same. It is rare to have an attribute name and attribute value be the same.
5.4 Using a unique string literal

www.domain.com/{string-literal}

Example

www.domain.com/thanksgiving-black-friday-2017

In the above example, there is no correlation to any attribute name or attribute value.
5.5 Using soft filters

For all the above URLs, you can also add additional filters as parameters.

Example:

```
www.domain.com/shirts?color=red
AttributeName1 = category, attributeValue1 = shirts
AttributeName2 = color, attributeValue2 = red
```

In the above example, the first attribute (category=shirts) is part of the URL path (hard filter) and cannot be changed. However, color=red is passed as query parameter (soft filter) and can be unset by the user.
Chapter 6  Attribute Names and Values

When using attributes as part of the URL or as query parameters, the attribute names and values are normalized to be SEO friendly.
6.1 Attribute names and values when used as part of URL paths (Hard Filters)

When using the attribute names as part of the URL path, attribute names are normalized as follows:

1. Remove apostrophe and unicode left+right apostrophes
2. ASCII folding (removes accents)
3. Replace all symbols, and unprintable characters like tab and newline with space
4. Trim, and replace space with dash (-)
5. Collapse: remove duplicate dashes and remove any dashes from beginning and end.
6. Convert to lower case

Examples of normalized attribute names when used in URL paths:

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Normalized Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brand</td>
<td>brand</td>
</tr>
<tr>
<td>Product Type</td>
<td>product-type</td>
</tr>
<tr>
<td>MP3 Model</td>
<td>mp3-model</td>
</tr>
<tr>
<td>Modèle de téléphone</td>
<td>modele-de-telephone</td>
</tr>
<tr>
<td>Material/Cloth</td>
<td>material-cloth</td>
</tr>
</tbody>
</table>

When using the attribute values as part of the URL path, attribute values are normalized as follows:

1. Remove apostrophe and unicode left+right apostrophes
2. ASCII folding (removes accents)
3. Replace all symbols, and unprintable characters like tab and newline with space
4. Trim, and replace space with dash (-)
6. Remove any dashes from beginning and end.
7. Convert to lower case
Examples of normalized attribute values when used in URL paths:

<table>
<thead>
<tr>
<th>Attribute Value</th>
<th>Normalized Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shirt</td>
<td>shirt</td>
</tr>
<tr>
<td>Long sleeve shirts</td>
<td>long-sleeve-shirts</td>
</tr>
<tr>
<td>Men’s shirt</td>
<td>mens-shirt</td>
</tr>
<tr>
<td>Home Décor</td>
<td>home-decor</td>
</tr>
<tr>
<td>MP3 Players</td>
<td>mp3-players</td>
</tr>
<tr>
<td>Books &amp; Music</td>
<td>books-music</td>
</tr>
<tr>
<td>BrandX®</td>
<td>brandx</td>
</tr>
<tr>
<td>Men &gt; Shirt</td>
<td>men-shirt</td>
</tr>
<tr>
<td>men&gt;shirt</td>
<td>men-shirt</td>
</tr>
<tr>
<td>red/Green - greenish</td>
<td>red-green-greenish</td>
</tr>
</tbody>
</table>
6.2 Attribute names and values when used as query parameters (Soft Filter)

When filters are specified as part of the query parameters

(www.domain.com/url_path?name=value)

- name is the normalized attribute name.
- value is the UTF-8 encoded and URL encoded attribute value

Sitecore Discover performs the following normalization steps for attribute names:

1. Removes symbols
2. Converts to lower case
3. ASCII folding (removes accents)
4. Joins multiple words with a separator "_"

Example:

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Normalized Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight</td>
<td>weight</td>
</tr>
<tr>
<td>Product Type</td>
<td>product_type</td>
</tr>
<tr>
<td>ProductType</td>
<td>producttype</td>
</tr>
<tr>
<td>Malzeme Türü</td>
<td>malzeme_turu</td>
</tr>
</tbody>
</table>

The attribute value should be sent exactly as sent in the product catalog feed. The value should be UTF-8 encoded and URL encoded.

Examples of UTF-8 and URL encoded attribute values:

<table>
<thead>
<tr>
<th>Attribute Value</th>
<th>UTF-8 encoded and URL encoded value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shirt</td>
<td>Shirt</td>
</tr>
<tr>
<td>Shirt Sale</td>
<td>Shirt+Sale</td>
</tr>
<tr>
<td>Books &amp; Music</td>
<td>Books+%26+Music</td>
</tr>
<tr>
<td>BrandX®</td>
<td>BrandX%C2%AE</td>
</tr>
</tbody>
</table>
6.3 Sitecore Discover reserved words

There are several attribute names that are part of the Standard Product Specifications. These have specific purpose and meaning. Please see Sitecore Discover Data Feed Specifications under Customer Engagement Console -> Developer Resources -> Documentation for more details.
Chapter 7  Distinguishing URLs for Sitecore

You may choose to identify all URLs that should be redirected to Sitecore Discover. You may also want to provide a simple way to serve the same pages via your own eCommerce servers. Further, there will be a transition phase during which your CDN/Reverse Proxy will need to support your existing category page serving as well Sitecore Discover. You will need a mechanism for your CDN/Reverse Proxy to identify if a particular URL should be directed to Sitecore Discover Service. To do this, you may employ any of the following techniques.
7.1 URL Path

You may include a specific path like `rfk` (or `cat` or anything else you prefer) as part of your URL path. Your CDN can then look for this URL path and set the origin source to Sitecore Discover if it is present. This is the easiest way to configure your CDN/Reverse Proxy. Note that the path can be anything you like.

Example:

```
<customer-domain.com>/rfk/<category-name>?<additional-params>
```

Note:
Introducing a subpath would require you to do a 301 redirect for the original path for SEO traffic until search engines update their indexes.
7.2 Query parameter

Add an additional query parameter to the category page URLs. For example, you can add \( rfk=1 \) to the URLs. Your CDN/Reverse Proxy can then look for this query parameter and set the origin source to Sitecore Discover if this parameter is present. This requires a more complex setting on the CDN. Please ensure your CDN supports query parameter redirects.

Example:

\[
<\text{customer-domain.com}/<\text{category-name}>?<\text{additional-params}>&rfk=1
\]

Note:

Make sure that you do not block search engines from traversing paths with query parameters.
7.3 Subdomain

You may introduce a special subdomain as part of your host name as part of the category page URL. Your CDN/Reverse Proxy can then look for this subdomain in the host name and set the origin source to Sitecore Discover if the subdomain is present.

Example:

```
rfk.<customer-domain.com>/<category-name>?<additional-params>
```

Note:
Please make sure your management is ok with introducing a subdomain into the URL
Chapter 8  CDN/Reverse Proxy Configuration
8.1 Origin Host

To configure your CDN, you will need origin to be set to the Sitecore Discover service.

Protocol: https
Host: <rfk-page-host>
URI: <rfk-page-uri>
Method: GET

Note: To use Sitecore Discover APIs, go to Customer Engagement Console -> Developer Resources -> API Access for page host and URI path.
8.2 Configuration Options

Customers will map the category URL paths to be passed to Sitecore Discover servers. You may choose to:

1. Proxy from a CDN
2. Proxy from a reverse proxy server like NginX or Apache, or
3. Route using DNS (if using a subdomain like rfk.customer.com)
4. By making a request to Sitecore Discover load balancers directly by any other mechanism

Sitecore Discover will advise on the exact strategy based on Customer's CDN/Reverse Proxy setup including any SSL certificate requirements. Customers may provide their own fallback that serves the pages from its eCommerce servers.

Customer CDN/Reverse Proxy would map all intended requests for:
<customer-domain.com>/<category-name>?<additional-params>

to Sitecore Discover servers as follows:

**OPTION 1 (Recommended)**

<rfk-page-host>/<rfk-page-path>/<category-name>?<additional-params>

This is the simplest of all options. All you need to do is to set the origin host and add the path prefix to the origin URI path as shown above.

**OPTION 2 (Deprecated - Will be removed on 1/1/2021)**

<rfk-page-host>/rfkj/<version>/<ckey>/pg

and pass the original URI in the header:

x-rfk-uri: <category-name>?<additional-params>

**OPTION 3 (Deprecated - Will be removed on 1/1/2021)**

The URI can be URI encoded, and passed as query parameter as:

<rfk-page-host>/rfkj/<version_number>/<ckey>/pg
?uri=<uri-encoded-category-name>&<additional-params>

Example

www.domain.com/shirts?color=red would be mapped to:
<table>
<thead>
<tr>
<th>Original Request</th>
<th><a href="http://www.domain.com/shirts?color=red">www.domain.com/shirts?color=red</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Option 1</td>
<td><code>&lt;rfk-page-host&gt;/rfk-page-path&gt;/shirts?color=red</code></td>
</tr>
<tr>
<td>Option 2 (Header)</td>
<td><code>x-rfk-uri: /shirts?color=red</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;rfk-page-path&gt;/rfkj/&lt;version_number&gt;/ckey/pg</code></td>
</tr>
<tr>
<td>Option 3 (Parameter)</td>
<td><code>&lt;rfk-page-path&gt;/rfkj/&lt;version_number&gt;/ckey/pg?uri=%2Fshirts%3Fcolor%3Dred</code></td>
</tr>
</tbody>
</table>
8.3 CDN/Reverse Proxy Cache Control

Customer must disable all caching for the category pages on their CDN/Reverse Proxy servers. We will also send cache control headers to not cache the response, and Customer's CDN must honor the headers.
### 8.4 CDN/Reverse Proxy Cookies Passthrough

Customer must pass through all cookies that start with `__r` (two underscores and letter r). These cookies are required to:

- track the user session, and
- preview settings if the customer (merchandiser) wants to change configuration in Sitecore Discover Customer Engagement Console and test its impact on the website before publishing it.

**Note:** When setting up a CDN, ensure that your A/B split test tool is set up to split traffic based on the appropriate portion of the URL path. For example, when navigating back to a given category page, the buckets must be maintained even when additional parameters are added to the URL path.
### 8.5 Ramping up Traffic to Hosted Sitecore Discover Service

This section is applicable only if Sitecore Discover is hosting your category pages. When you launch Category pages, you may choose to ramp up the traffic to Sitecore Discover service slowly, or to switch to Sitecore Discover service instantly for all traffic.

If you decide to ramp up traffic slowly or prefer to do A/B test between Sitecore Discover and non-Sitecore Discover experience, Sitecore provides a traffic management service to help decide on a per-request basis whether traffic should be sent to Sitecore Discover or to the customer’s service.

To make such a decision, Sitecore Discover uses the two cookies described below.

<table>
<thead>
<tr>
<th>Cookie</th>
<th>Description</th>
</tr>
</thead>
</table>
| __rpg  | String: A Sitecore Discover managed cookie to track if the user is selected to be in the Sitecore Discover experience or not (i.e. served by Sitecore Discover), and some book-keeping information to manage the traffic ramp up. The cookie has the following format:  

\[
__rpg=isRfkExperience-TrafficTimestamp-CookieTimeStamp-Traffic
\]

| isRfkExperience: 0|1,  
| if 0, user is not in Sitecore Discover Experience.  
| if 1, user is in Sitecore Discover experience  
| Traffic: integer from 0 to 100. A number representing the traffic split between Sitecore Discover and non-Sitecore Discover experience.  
| TrafficTimestamp: timestamp in milliseconds, indicating the time when the traffic split was last modified. If the traffic split has been modified since the cookie was created, redistribute the customer and assign a new cookie  
| CookieTimestamp: timestamp in milliseconds, indicating time at which the determination was made if a user should be in the Sitecore Discover experience or not.  
| Example:  

\[
__rpg="1-1575683837-1575683837-37"
\]  

The cookie should not be modified in any way, as the format of the cookie and
its content is critical for the Traffic Manager to work properly.

<table>
<thead>
<tr>
<th>__rsu</th>
</tr>
</thead>
</table>

String: A cookie to force a user into or out of the Sitecore Discover experience. This is useful for testing Sitecore Discover vs. non-Sitecore Discover experience before rolling out the Sitecore Discover experience.

Value:
- 1, force user to be in Sitecore Discover experience.
- 0, force user to NOT be in Sitecore Discover experience.
- not set, traffic manager will determine if the user should be in the Sitecore Discover experience or not.

You may set the cookie manually, or you may execute `rfk.selected(0)` in the browser console. You may delete the cookie if you want the Sitecore Discover Traffic Manager to determine the user’s experience.

Make sure your CDN is programmed to check for these cookies and then forward the traffic appropriately.

**Note:** Sitecore Discover ramp up for Category Pages is independent of any other ramp ups and does not preserve the user’s bucket when increasing or decreasing the traffic percentage.
8.6 CDN Pseudo Logic

Here is pseudocode of the logic to use within your CDN to correctly implement Category page proxy logic.

```javascript
if (request.uri is a candidate for proxying to RFK) {
  // A uri is candidate for proxying to RFK when
  // the uri contains one of the strategies as described in
  // Distinguishing URLs for Sitecore Discover.

  var isRfkExperience = 1
  var cacheTimeInMs = 360000 (1 hr * 60 min * 60 sec * 1000ms)
  var launching_category_pages = 1

  if (launching_category_pages) {
    if (!cookie.exists(__rsu)) {
      if (cookie.get(__rsu) == 0) {
        isRfkExperience = 0
      } else {
        if (!cookie.exists(__rpg)){
          // call traffic manager if the user should be in RFK experience
          cookie.set(__rpg) = trafficManager()
        } else if (__rpg - cookieTimestamp < (time.now() - cacheTimeInMs)) {
          cookie.set(__rpg) = trafficManager(__rpg cookie value);
        }
        isRfkExperience = cookie.value(__rpg).extractFlag()
      }
    }
  }

  if (!isRfkExperience) {
    return;  // do not change anything, call customer's service.
  }

  // make request to Sitecore Discover Service
  // ...
}
```

**Note:** The section on launching_category_pages should be removed after Sitecore Discover Category Pages has been fully launched so it doesn't incur unnecessary requests to Sitecore Discover Traffic Manager or set the variable launching_category_pages to 0.
### 8.7 CDN Example: CloudFront

CloudFront, the CDN provided within Amazon Web Services, is capable of handling multiple origins to fulfill requests. In addition to having the ability to route via URI path, CloudFront also allows for URI and header rewrites via Lambda@Edge function calls.

Steps to successfully implement Sitecore Discover Category Pages via CloudFront:

1. Set up a CloudFront distribution for your site with your own servers or S3 as the primary Origin.
2. Set up a secondary Origin for the Sitecore Discover Page service. The origin domain name should be the same as the primary origin.

#### Origin Settings

<table>
<thead>
<tr>
<th>Origin Domain Name</th>
<th>yourdomain.com</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin Path</td>
<td></td>
</tr>
<tr>
<td>Enable Origin Shield</td>
<td>Yes</td>
</tr>
<tr>
<td>Origin ID</td>
<td>Custom-yourdomain.com</td>
</tr>
<tr>
<td>Minimum Origin SSL Protocol</td>
<td>TLSv1.2</td>
</tr>
<tr>
<td>Origin Protocol Policy</td>
<td>HTTP Only</td>
</tr>
<tr>
<td>Origin Connection Attempts</td>
<td>3</td>
</tr>
<tr>
<td>Origin Connection Timeout</td>
<td>10</td>
</tr>
<tr>
<td>Origin Response Timeout</td>
<td>30</td>
</tr>
<tr>
<td>Origin Keep-alive Timeout</td>
<td>5</td>
</tr>
<tr>
<td>HTTP Port</td>
<td>80</td>
</tr>
<tr>
<td>HTTPS Port</td>
<td>443</td>
</tr>
</tbody>
</table>
3. Set the headers for the origin

<table>
<thead>
<tr>
<th>Header Name</th>
<th>Required</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rfk-subdomain</td>
<td>yes</td>
<td>-</td>
<td>The Sitecore Discover subdomain used for APIs. You may obtain the subdomain from CEC &gt; Developer Resources &gt; API Access. Typical subdomain is: (&lt;nickname&gt;.rfk.yourdomain.com)</td>
</tr>
<tr>
<td>rfk-traffic-split</td>
<td>no</td>
<td>(Controlled by Sitecore Discover)</td>
<td>The percentage of matching traffic to be sent to the Sitecore Discover page (0-100). Set this header if you would like to overwrite the value provided by Sitecore Discover.</td>
</tr>
<tr>
<td>rfk-path</td>
<td>no</td>
<td>/rfk/</td>
<td>Match all traffic where the URL path starts with the value of the header. Required if your URL path starts with a specific string and all such urls are to be served by Sitecore Discover. Note that you may configure Sitecore Discover system to use a custom string like /cat/</td>
</tr>
<tr>
<td>rfk-header</td>
<td>no</td>
<td>rfk</td>
<td>Match all traffic where the request has header &lt;value&gt;: 1. The default value matches header rfk:1</td>
</tr>
<tr>
<td>rfk-query</td>
<td>no</td>
<td>rfk</td>
<td>Match all traffic where the request has a query parameter of ?&lt;value&gt;=1. The default value matches query parameter ?rfk=1</td>
</tr>
</tbody>
</table>

4. Using this new Origin, create a Behavior in CloudFront which sends the appropriate requests to the Sitecore Discover backend. For example, if your pages served by Sitecore Discover exist within the /rfk/ path in your site URIs, you would choose /rfk/* for this
behavior. Your existing site backend should remain as the final / default (*) behavior. This value must match the rfk-path header configured in step 3.

5. When configuring the behavior, ensure that you are passing all headers, cookies, and query string parameters:
### Cache Behavior Settings

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path Pattern</td>
<td>/rfk/*</td>
</tr>
<tr>
<td>Origin or Origin Group</td>
<td>Custom-page.uat.rfksrv.com</td>
</tr>
<tr>
<td>Viewer Protocol Policy</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>Allowed HTTP Methods</td>
<td>GET, HEAD, OPTIONS</td>
</tr>
<tr>
<td>Field-level Encryption Config</td>
<td></td>
</tr>
<tr>
<td>Cached HTTP Methods</td>
<td>GET, HEAD (Cached by default)</td>
</tr>
<tr>
<td>Cache Based on Selected Request Headers</td>
<td>All (Improves Caching), Whitelist, All</td>
</tr>
<tr>
<td>Minimum TTL</td>
<td>0</td>
</tr>
<tr>
<td>Maximum TTL</td>
<td>31536000</td>
</tr>
<tr>
<td>Default TTL</td>
<td>86400</td>
</tr>
<tr>
<td>Forward Cookies</td>
<td>All</td>
</tr>
<tr>
<td>Query String Forwarding and Caching</td>
<td>Forward all, cache based on all</td>
</tr>
<tr>
<td>Smooth Streaming</td>
<td>No</td>
</tr>
<tr>
<td>Restrict Viewer Access</td>
<td>No</td>
</tr>
</tbody>
</table>

**NOTE:** If traffic split and query string (?rfk=1) is not needed, proceed to step 10.
6. Create a Lambda function with the Author from Scratch option and choose an existing or new Lambda role with Basic Execution access. Choose NodeJS 10.x as the Runtime.

7. Download the NodeJS 10.x code from Customer Engagement Console > Developer Resource (https://cec.rfksrv.com/developer-resources) by clicking Download CDN Lambda Function and selecting cloudfront as the CDN. This code will modify the request URI to match the Sitecore Discover Page backend and perform traffic splitting.

Copy the contents of the file to the ‘Function code’ section. The handler should be index.handler
8. Save the Lambda function and publish an initial version using this option:

Copy the full Lambda ARN from the new version you published. In the above example:

**ARN** - arn:aws:lambda:us-east-1:607987562740:function:ReflectionPageService:1

9. Go to the Behavior you previously created in CloudFront, and attach the above Lambda Function as both an origin request and origin response event:

10. Once your CloudFront distribution has finished deploying, you should be able to make requests in the format of:

   1) `https://www.yoursite.com/rfk/<category-path>` and the requests will be fulfilled by the Sitecore Discover Page backend service.

   2) `https://www.yoursite.com/<category-path>?rfk=1` and the requests will be fulfilled by the Sitecore Discover Page backend service
8.8 CDN Example: Varnish for Magento Cloud

Magento Cloud relies on Varnish 2 (a pretty old version) to cache content and perform request normalization. Varnish can be used as a caching HTTP reverse proxy with header and URI rewrite.

To set up category pages, you need to perform following steps.

**Step 1: Create the origin**

After logging in Admin in Magento, navigate to *Store > Configuration*.

Then in the left-hand panel navigate to *Advanced > System*. In the *Full Page Cache > Fastly Configuration > Backend Settings* section, click the *Create* button to create a backend.

In the form, you must fill out the origin name (i.e. `reflektion_origin`), address page service as provided to you by Sitecore Discover. You can obtain the appropriate URL from Customer Engagement Console (CEC) in the Developer Resources section. For example, in our UAT environment origin would be https://page.uat.rfksrv.com.
Step 2. Setup request routing

Once the origin is created, we need to specify how to route. Fastly relies on Varnish for this activity and the behavior is specified using the VCL spec.

First, we need to create a VCL snippet by clicking the Create button on Custom VCL Snippet.

Custom VCL Snippets

This option allows you to manage your own custom VCL snippets. You can learn more here. Please note after you have created your snippets you have to click on Upload VCL to Fastly to upload them along side stock Fastly VCL.

Create Custom Snippet

Next, assign a name (i.e. rfk_route), set recv type and use the following snippet to create the routing rule that routes all URLs containing rfk=1:

```vcl
# we check if it is a RFK URL. In this case if it contains ?rfk=1 or &rfk=1
if (req.url ~ "(?|&)rfk=1") {
  # it sets the RFK origin created in step 1
  set req.backend = F_reflektion_origin;
  # it changes the URL to RFK url and passes through the actual category URL
  # replace the CKEY with your CKEY
  set req.url = "/rfkj/1/<CKEY>/pg/uri"+req.url;
  # it filters out all but rfk cookie
  if (req.http.Cookie) {
    set req.http.Cookie = regsuball(req.http.Cookie, ";(__r\w*|__json_debug)="", "");
    set req.http.Cookie = regsuball(req.http.Cookie, ";[^\ ][^;]*", ");
    if (req.http.Cookie == ") {
      remove req.http.Cookie;
    }
  }
  return(pass);
}

#FASTLY recv
if (req.method != "HEAD" && req.method != "GET" && req.method != "FASTLYPURGE") {
  return(pass);
}
return(lookup);
```

Notes:

- It does not need the recv function definition.
- The origin is referenced with the "F_" prefix.
- To disable the rule, you need to clean the code in the snippet definition. You cannot just delete it because the full definition is still available in Varnish and accessible using API.

After saving, you need to scroll up and click the Upload VCL to Fastly button.

Some customers have reported that they cannot use the UI provided by Fastly using the Magento Cloud account. In that case, you need to use the API approach.

Find technical docs as follows:

- Varnish cookies: [https://varnish-cache.org/docs/2.1/tutorial/cookies.html](https://varnish-cache.org/docs/2.1/tutorial/cookies.html)
Chapter 9  B2B / Resellers / Stores

Considerations

Sitecore Discover supports restrictions (also referred to as availability) based on business-to-business (b2b) customers, resellers, and stores seamlessly across the Sitecore Discover platform. Restrictions are based on customers providing us the restriction identifier that is the same identifier provided as part of the product feed for restrictions. Restrictions are modeled as store ids within Sitecore Discover.

Customer should provide the information to Sitecore Discover about the restriction to apply. This can be done either by calling rfk.push() to update context in the browser, or by specifying it as part of the API call.

See Sitecore Discover Widget Context Technical Guide on how to call rfk.push() with appropriate restriction identifiers [aka store identifiers].
9.1 Hosted By Sitecore Discover / JS Implementation

If Sitecore Discover is hosting the pages, then you must build out a mechanism to provide the restriction identifier to Sitecore Discover from the browser after Discover has served the page. Usually this is managed as part of the user's login credentials either using a local storage (see: https://www.w3schools.com/jsref/prop_win_localstorage.asp) or cookie. Following is one way you can implement it.

1. Define a mechanism to store and retrieve restriction id(s) from the browser in the local storage, cookie, or any other mechanism.
2. When the page is loaded in the browser, verify the user's login credentials, and set the restriction id(s) corresponding to the user's credentials by calling rfk.push(). How a user's credentials are managed, is up to you.
3. Any time the user signs in after the page is loaded:
   1) Retrieve the restriction id from your servers and store it in the local storage, cookie or any other mechanism.
   2) Call rfk.push() to apply the restriction id(s). Sitecore Discover will update the results immediately.
4. Anytime the user signs out:
   1) Delete the restriction id(s) information stored in localStorage, cookie or via any other mechanism.
   2) Call rfk.push() to remove the restriction id(s). Sitecore Discover will update the results immediately.
   3) Have JavaScript code included as part of the category page template that you have provided to Sitecore Discover that reads the local storage, cookie or any other mechanism used to store the restriction id(s), and call rfk.push() with these restriction id(s).

Note:

Ensure that this code executes before the line where Sitecore Discover beacon is included to avoid page flicker.
9.2 API Implementation

Customers implementing Category Pages can simply pass the restriction id(s) via the API.
Sitecore Discover implements several SEO-related features that make the official bots to index the pages smartly. Following are few considerations:

- We support SEO friendly URLs (See Category URL Design)
- We include several SEO related tags including meta-title, meta-content, meta-keywords, title, page heading, and description (see Category Feed)
- We support any plain text or HTML text that Customer would like to include on the page
- We include www.schema.org markup for all product items listed on the page
- We support generation and serving of fully formed pages for the first page (i.e. page is rendered on server-side) so BOTs can easily index these pages even if they do not support JavaScript. Subsequent pages (“Load More...”, 1,2,3... pagination is supported through JavaScript)
- Most modern bots support JavaScript for crawling and indexing. The hashbang/escaped fragment URLs have been deprecated, and Sitecore Discover does not support them.
- We support HTTPS protocol. http protocol is deprecated and should not be used.
- We support responsive pages for different devices. We also support different pages customized for a given device (desktop, tablet, mobile).
- All pages are served within the customer’s domain. There is no HTTP redirect required.

We will also accommodate any special requirements that a customer may have for SEO. We encourage customers to discuss their SEO strategy with us.
You can choose to power Category Pages via APIs or have Sitecore Discover host them. Following are the steps to follow to launch Category pages.
11.1 Hosted by Sitecore Discover

You will be responsible for following tasks when launching Category pages that are hosted by Sitecore Discover.

1. Provide category feed. See Category Feed and Category feed specs. Ensure products in product feeds are tagged with proper categories.
2. Provide a URL to a template page hosted on your website with appropriate tags. See Category Page Template.
3. Determine a common Category URL Path/pattern to identify pages that would be powered by Sitecore Discover. See URL Path.
4. Configure your CDN or your ecommerce service to forward the Category page requests to Sitecore Discover servers. See CDN Configuration.
5. Generate menus and links on your website using the URL Path strategy you chose in the previous step.
6. Set up Category Pages within Customer Engagement Console.
7. If rolling out the solution gradually, then additional work is required by Customer to only forward portion of the requests.
11.2 API Integration

You will be responsible for the following tasks when launching Category pages when integrating via Sitecore Discover APIs.

1. Provide category feed. See Category Feed and Category feed specs. Ensure products in product feeds are tagged with proper categories.
2. Set up Category Pages within Customer Engagement Console.
3. Follow all the steps required to integrate via APIs.
Chapter 12  Notes

- All products that can show on Category pages should also be searchable.
- Sitecore Discover does not support IE8 and below browsers. If customer desires to serve these browsers, requests from these browsers should go to Customer’s eCommerce server. Our experience shows traffic from such browsers is < 1% and is sufficient to show a message to the user to upgrade their browsers.
- Bots will always see non-personalized results.
- If Sitecore Discover is hosting the pages, it will serve category pages in one of the ways below:
  - With full HTML DOM
  - With embedded JSON data and render it in JS in browser.
- Sitecore Discover will provide SEO friendly URL link so bots can crawl.
- Sitecore Discover can implement a Load More button or pagination. If a visitor goes back to the category page, Sitecore Discover will attempt to scroll the user to the same physical position. However, we cannot guarantee the exact same position on the page. This is because Discover continuously changes results based on aggregate user activity as well as current user’s activities. Hence, the results shown may not have the same order when the page is refreshed.
- You must not cache pages on your end if personalized results are desired.
- If a category page returns zero products, the category page will be displayed with zero products.
- If a category is deleted (i.e., the category link might still be bookmarked or indexed by search engines), customer should redirect to its own page. Such a request should not reach Sitecore Discover.
Chapter 13  What to do when a URI is not identified by Sitecore Discover

There are times when a page with a given URL is not mapped within the Sitecore Discover system. This can happen when, for example, a category associated with a given URL was removed from the category feed, and once the feed is processed in Sitecore Discover, the URL associated with the non-existent category will also be removed. However, a user might have bookmarked the page or search engines may have indexed the expired pages.

In such cases, you may configure CEC to redirect to a specific page (302 redirect) like a Page Not Found page, or you may configure for Sitecore Discover to send a 404. If a 404 is returned, please ensure your CDN handles it in some user-friendly meaningful way.

![Hosted Pages Settings](image)

*Figure: CEC Domain Settings - Hosted Pages Settings*
Sitecore Discover provides a convenient way for you to build dynamic pages via APIs while allowing merchandisers significant flexibility to manage and modify these pages through Customer Engagement Console without developer intervention. Merchandisers can manage the page's content, search results, recommendations, SEO headers, etc.

This approach provides the best of both worlds. It allows custom functionality like managing stores, user login, etc. that is unique to you once, and allowing merchandisers to modify most of the visible content through CEC.

To achieve it, you would do the following:

1. Start with a page template and identify a tag within the page template.
2. Define the landing page in CEC, including all the widgets you would like to allow control of to the merchandiser / marketer.
3. Make an API call to retrieve all the widgets for the given landing page identified by the page URL you assigned when creating the landing page above. In the API call, make the request to retrieve products as well as html and CSS template content for a specific device. You will be returned the products and/or templates for all widgets on the given landing page in the same order you have defined in CEC as though you made a batch request.

Example request:

```json
{
    "context": {
        "page": {
            "uri": "/"  // uri of the page
        }
    },
    "widget": {
        "all": true  // ask for all widgets in the page
    },
    "content": {
        "product": {}
    }
}
```
4. Iterate over all with widgets under the batch key for each widget, create a div for each widget, and place the html content of the widget in this div. Add the CSS Names to the div for the widget that is returned as part of the widget results. You can also apply the CSS returned as part of the above request through any mechanism convenient to you. For product widgets, render the products and facets etc. just the way you would do for search results page.

5. That's it. You now have the full page rendered and integrated with your system.

6. Now merchandizers can make any modification to the widget or the widget's appearance, and when the API call is made the modified changes will be seen on your website!