Query strings in URLs are useful and important in many ways, for example, when passing various values between pages or retrieving data from the database. In some cases though, an unauthorized user could obtain sensitive data or harm the system by entering a URL with **tampered query string parameters**.

To prevent tampering with query string parameters, the system adds the result of a **hash function** at the end of each URL. Such URLs can look like this:

http://localhost/Kentico/cms/getfile/2d003444-9a27-43c9-be97-4f5832474015/Partners_logos_silver.aspx?latestfordocid=75&**hash=eee41e027bd45142fd1f44d173d61950f34d6e98f4e35018fda70542322adc77&chset=013bca78-6bf2-42ac-8959-b8bbbeb0a8e8**

The part in bold is the hash parameter added to the URL. If an attacker modifies the parameters and tries to submit such a URL, the system rejects it, because the query string parameters and hash do not match.

The hash function is calculated from the query string parameters using SHA-2. The hashing is used in various parts of the system:

- Dialog boxes
- Macro signatures
- In links for downloading files (e.g., getamazonfile.aspx, getazurefile.aspx or sometimes even with getfile.aspx) – to ensure that users download only the file specified in the original URL and nothing else.

Salt

Protection of query string parameters using only the hash function would not be enough because attackers are able to reverse the hash function using rainbow tables. For this reason, the system adds a **salt** to the URL before hashing it.

Hash calculation:

URL parameters + salt -> all this is hashed using SHA-2 and added to the original URL.

The salt is a secret string of characters, which users do not have access to. In Kentico, the default salt added to the URL is a **randomly generated GUID** specified by the *CMSHashStringSalt* key in the web.config file:

```
<add key="CMSHashStringSalt" value="e68b9ad6-a461-4707-8e3e-ece73f03dd02" />
```

You may change this value if you need to. If you delete this key completely, the system uses the connection string stored in the web.config file as the salt.

If you have already stored some persistent information (links for downloading files, image links, macros) in the system and you change the salt calculation, then these links and **information may become broken**. You will have to re-save the content to create hashes with the new salt.

User specific hash

Sometimes it is useful to add user-specific information to the hash. Either the user session ID or IP address (if there is no session ID) are used for this purpose. If attackers manage to eavesdrop a URL and try to exploit it to gain sensitive data, they would not succeed, as their session ID would not match the hash.

The user specific hash is used mainly for non-persistent information, such as displaying dialog boxes.

Note that if you use user specific hashing and save some content with it, users other than the one who saved the content will not be able to use it.

Custom hash validation

We recommend using hashing features when developing custom dialog boxes. The following methods are available in the Kentico API:

Hash calculation:

- **QueryHelper.BuildQueryWithHash** - builds the entire query string with any number of parameters, automatically calculates and adds a hash parameter (with all parameters as the input).

  Example - BuildQueryWithHash

  ```
  using CMS.Helpers;

  ...

  // Sample URL parameter values
  string urlParamValue1 = "value1";
  string urlParamValue2 = "value2";

  // Sample base URL
  string url = "base";

  // Creates a query string containing the sample parameters and the hash
  string queryString = QueryHelper.BuildQueryWithHash("parameter1", urlParamValue1,
  "parameter2", urlParamValue2);

  // Adds the query string parameters to the URL, including the hash
  url += queryString;
  ```

- **ValidationHelper.GetHashString** - creates a SHA2 hash of a specified value. Can be used to create hashes for any values, not just query strings. The optional *HashSettings* parameter allows you to specify whether the hash is user-specific, and also add a custom salt.

  Example - GetHashString

  ```
  using CMS.Helpers;

  ...

  // Sample string value
  string customValue = "value";

  // Creates a hash for the custom value
  string hash = ValidationHelper.GetHashString(urlParameterValue);
  ```

Hash validation:

- **QueryHelper.ValidateHash** - works directly with the query string and can exclude individual parameters from hash validation.
- **ValidationHelper.ValidateHash** - general hash validation method, can be used for macro signatures.

You can validate hashes on pages individually:

```
using CMS.Helpers;

...

// Validates a hash value
if (QueryHelper.ValidateHash("hash"))
{
        // The validation was successful, perform any required actions
}
```

Or you can validate hashes automatically by including the *[HashValidation(Name="hash2")]* attribute on a page:

```
using CMS.UIControls;
using CMS.Helpers;
using CMS.Base;

[HashValidation(HashValidationSalts.REDIRECT_PAGE)]
public partial class CMSMessages_Redirect : MessagePage
{
        /// <summary>
        /// OnInit event.
        /// </summary>
        protected override void OnPreInit(EventArgs e)
        {
                string url = QueryHelper.GetText("url", String.Empty);
                CheckHashValidationAttribute = !(url.StartsWithAny(StringComparison.
InvariantCulture, "~", "/"));
                base.OnPreInit(e);
        }
```