

# Keywords in CMRL

CMRL Version 1.0





# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>The Problem</b>	<b>3</b>
<b>3</b>	<b>The Solution</b>	<b>4</b>
<b>4</b>	<b>Keywords Persist</b>	<b>5</b>
<b>5</b>	<b>Summary</b>	<b>7</b>

## 1 Introduction

Keywords in CMRL are used to “correct” user input to a list of standard tokens that are somehow “expected” under a particular match. Keywords make it easy to handle the misspellings, abbreviations, and other ambiguities that crop up with text-message input.

This document describes keywords in CMRL. The document assumes that you have some familiarity with DOTGO and CMRL (obtained, e.g., by reading [A Brief Introduction to DOTGO](#)).

## 2 The Problem

Suppose you were to implement an engine under the internet domain name “example” and the match “travel” related to travel between the major east-coast cities Boston, New York, Philadelphia, and Washington, D.C., and suppose that the engine were to take a pair of these cities as arguments. The engine might be referenced by the following CMRL fragment:

```
<match pattern="travel">  
  <engine href="http://example.com/cgi-bin/travel.cgi"/>  
</match>
```

A possible query might be

example travel New York Washington, D.C.

The problem is that there are many other queries that express exactly the same intention, including for example

example travel new york washington, d.c.

example travel new yorke washington, d.c.

example travel ny wash

example travel nyc dc

In other words, the problem is that the query might contain mis-spellings and abbreviations, which must be handled to make the engine useful in practice.

## 3 The Solution

You could handle all possible mis-spellings and abbreviations in the engine, but there is an easier and better way. You probably already know that the system seeks to obtain matches using phonetics and abbreviations as well as by direct comparison. Well, the same feature is also available to “correct” other user input to a list of standard tokens. The feature is implemented using the `<keywords>` and `<keyword>` tags within the `<block>` tag. For example, the keywords “Boston,” “New York,” “Philadelphia,” and “Washington, D.C.” could be associated with the engine described above by the following CMRL fragment

```

<match pattern="travel">
  <block>
    <keywords>
      <keyword>Boston</keyword>
      <keyword>New York</keyword>
      <keyword>Philadelphia</keyword>
      <keyword>Washington, D.C.</keyword>
    </keywords>
    <engine href="http://example.com/cgi-bin/travel.cgi"/>
  </block>
</match>

```

With this construction, the engine would be passed “New York” and “Washington, D.C.” for all of the queries discussed in section 2. The query

```
example travel los angeles seattle
```

would not be corrected, and the engine would be passed “los,” “angeles,” and “seattle.” The query

```
example travel los angeles nyc
```

would be partially corrected, and the engine would be passed “los,” “angeles,” and “New York.”

So what’s going on? The `<block>` tag is a CMRL terminating node that allows various “helper” tags to be associated with another terminating node. The `<block>` tag must contain exactly one terminating node (which in this case is the `<engine>` tag) and may contain various other tags, including zero or one `<keywords>` tags (and zero or more `<set>` tags, which are beyond the scope of this document). The `<keywords>` tag may contain zero or more `<keyword>` tags, which list standard tokens that are somehow “expected” under a particular match.

How are the “corrected” tokens communicated to the engine? The system posts the corrected argument as the parameter “sys\_corrected\_argument.” The system posts the known (because they are successfully matched to a keyword) corrected tokens as the parameters “sys\_corrected\_argument\_known[*n*]” (where *n* is a zero-based index) and the number of such tokens as the parameter “sys\_num\_corrected\_argument\_known.” The system posts the unknown (because they are not successfully matched to a keyword) corrected tokens as the parameters “sys\_corrected\_argument\_unknown[*n*]” and the number of such tokens as the parameter “sys\_num\_corrected\_argument\_unknown.” And the system posts the (known and unknown) corrected tokens (together in the original order) as the parameters “sys\_corrected\_argument[*n*]” and the number of such tokens as the parameter “sys\_num\_corrected\_argument.”

## 4 Keywords Persist

It is important to understand that *keywords persist*. For example, say you started with following CMRL fragment

```

<match pattern="colors">
  <block>
    <keywords>
      <keyword>red</keyword>
      <keyword>green</keyword>
      <keyword>blue</keyword>
    </keywords>
    <engine href="http://example.com/cgi-bin/colors.cgi"/>
  </block>
</match>

```

but then changed your mind and replaced this CMRL fragment with the following CMRL fragment

```

<match pattern="colors">
  <engine href="http://example.com/cgi-bin/colors.cgi"/>
</match>

```

As long as the first CMRL fragment was accessed at least once, *the keywords “red,” “green,” and “blue” set in the first CMRL fragment would persist.*

Why is this? This feature allows keywords to be set in either a CMRL document or an engine. For example, another way to set the keywords in the first CMRL fragment above would be to have the engine referenced in that fragment return the following CMRL fragment

```

<block>
  <keywords>
    <keyword>red</keyword>
    <keyword>green</keyword>
    <keyword>blue</keyword>
  </keywords>
  <message>
    <content>Thanks for selecting one of red, green, or blue</content>
  </message>
</block>

```

In this case, the keywords “red,” “green,” and “blue” would be set the first time the engine was called and would persist until changed or cleared. Keywords are associated with (or stored with respect to) a terminating node of a CMRL document.

How is this useful? Suppose you have an engine that requires a lot of keywords—say the names of all cities in the United States. Presumably, you would not want to list such a large number of keywords in some huge CMRL file; similarly you would not want your engine to return such a large number of keywords every time it was called. The solution is to have your engine return the keywords only when specifically directed, say in response to a “reset” request. For example, you could put the following CMRL fragment into your CMRL document

```

<match pattern="reset">
  <engine href="http://example.com/cgi-bin/cities.cgi?mode=reset"/>
</match>

```

and have your engine return the following CMRL fragment in response to a reset request

```
<block>
  <keywords>
    <keyword>Aaronsburg</keyword>
    <keyword>Abbeville</keyword>
    <keyword>Abbotsford</keyword>
    .
    .
    .
    <keyword>Zwolle</keyword>
  </keywords>
  <message>
    <content>Keywords have been reset</content>
  </message>
</block>
```

You can keep the keywords up to date by calling the engine with a reset request any time the keywords returned by the engine are modified.

To clear all keywords, just use an empty <keywords> tag, i.e. a <keywords> tag that contains no <keyword> tags.

## 5 Summary

By correcting user input to a list of standard tokens, keywords make it easy to handle the misspellings, abbreviations, and other ambiguities that crop up with text-message input. The next step is to try it out for yourself.