

# Designing URI scheme handlers

URI schemes can be used in many different use cases, and this document solely concentrates on their usage with URI Scheme launcher within Windows phone (8).

## Introduction



You can design your own URI Scheme and register it in an application. Another application can use the URI scheme protocol to launch the app and pass it parameters (provided the app is installed).

The actual launch behaviour when using the URI scheme is:

- If there is just one handler application present, then it will be launched
- If there are multiple handlers, then a selection dialog is presented to the user for selecting the app to be launched
- If there are no handlers present, then user will be forwarded to the market place which opens with search for handler for the used URI Scheme protocol.

Note that there are several reserved protocols which you cannot overwrite, these are specified at [Reserved file and URI associations for Windows Phone 8](#) page.

Other than the reserved protocols any app can register any scheme. Therefore you should design the protocol with great care, and make it "accidental reuse" less likely (deliberate reuse cannot be prevented). Your application must also handle any incorrect usage without crashing.

A good base description for the URI scheme is shown in the [URI scheme wiki page](#), same base idea for the structure for the URI scheme can be found from Microsoft pages [Selecting a URI Scheme and Addressing Format](#).

```
<scheme name> : <hierarchical part> [ ? <query> ] [ # <fragment> ]
```

All and all, both of these are discussing the general usage, and as said in this article, we concentrate on single use case and operating environment. Basically the idea is to:

- Have a unique protocol name so there are no accidental duplicates
- Design a logical structure for the URI from which the usage logic can be derived and parsed easily.



Tip: Don't forget to document your app URI scheme and include it in our [URI Association Schemes List](#).

## Proposed structure

The base structure proposal is as follows:

```
<companyName>-<nameOfApp>://<action>/<noun>/?{queryStringParameters}
```

And if there is plan to have several versions we could also add the version into the structure

```
<companyName>-<nameOfApp>://<versionMajor.versionMinor>/<action>/<noun>/?  
{queryStringParameters}
```

The protocol identification part has two id's. First being company name (or abbreviation) and then application name (or abbreviation) separated by hyphen. The idea of this is to make it as much as possible globally unique, thus removing most chances of accidental overwrite.

Then when using the protocol in code, it is logically structured in a way that the usage is clear when reading it from left to right. Same idea should be used when parsing, the parse results could be handled with simple switch-case architecture, where the default case would always provide error message so the user could be notified that something went wrong, instead of causing any other unwanted behaviour.

One very nice example of this kind of URI scheme design is the [Nokia MixRadio URI scheme](#), basically the structure is as follows

```
nokia-music:// <action>/<noun>/?{queryStringParameters}
```

The available actions for the protocol are for example Play, Show and Search, and nouns include gigs, artist mixes etc.

## Designing parameters

---

Also the parameter designing needs to be taken care, also you should remember to well document the formats for each variable, as well as if there is any boundary values. And as said previously you should also remember to handle the parsing in a way that any invalid value would cause error note (or silent failing, if it makes more sense) instead of crashing the app or other unwanted behaviour.

You should take special care with values presented differently in different locales, for example with `DateTime` you could specify that it needs to be submitted as 'Sortable date/time pattern' formatted date-time string. and give example that it can be achieved by using code

```
string encodedTime = dateTimeVariable.ToString("s");
```

With strings you should specify the encoding, for example you could specify that `Uri.EscapeDataString()` needs to be used for encoding the string.

Also do remember that decimal separator differs between different regions, generally it can be dot or comma. And to always get is same way, you could specify that it is always required to be dot. And to get is as dot in all locales, you could propose using following code :

```
string doubleStr =  
doubleNum.ToString(System.Globalization.CultureInfo.InvariantCulture);
```

## Example application

---

There is example URI scheme protocol defined by Microsoft for driving and walking guidance application. This protocol is shown in [How to request driving or walking directions for Windows Phone 8](#) page. Note that this protocol is not designed according to proposal discussed in this article, but is used since Microsoft provides example for utilizing the protocol

The handler example is then explained in [How to respond to requests for directions for Windows Phone 8](#) page.