



^

## TW SDK QUICK START

## WE SHARE THE FUN

TouristWay developer tools give programmers and web designers extraordinary opportunities to exploit the component-based framework to express their expertise and creativity with unconstrained flexibility. TouristWay API and SDK offer plenty of way to tune and customize navigation and interface, to integrate external application and even extend standard features with third party tools.

### **TABLE OF CONTENTS**

- 1.0 Overview
- 1.1 Cut-to-the-chase version
- 2.0 How it works
- 2.1 Pages
- 2.2 Components



## **I.0 OVERVIEW**

TouristWay is extremely open to customizations and adjustments thanks to its component-based architecture.

Each component is designed to perform a specific set of tasks: there are components for searching the catalogue, for opening product details, for filling a reservation and so on for almost anything might turn useful in your online reservation system.

The look and behavior of each page is controlled by components that can be mixed to create unlimited combinations: for example you can place on your homepage 3 "showcase" component configured to display different special offers and then place a "library" component configured to list all the available destinations.

In particular, behavior of components can be adjusted by editing their configuration (each component has its own set of attributes) while graphic appearance can be manipulated by editing the template that each component uses.

Components get executed by TouristWay and their output fills a specific location on the page.

The container which is meant to keep all component's output together is called VIEWPORT. You can imagine viewport as a normal HTML structure with placeholders to be filled by the component output.

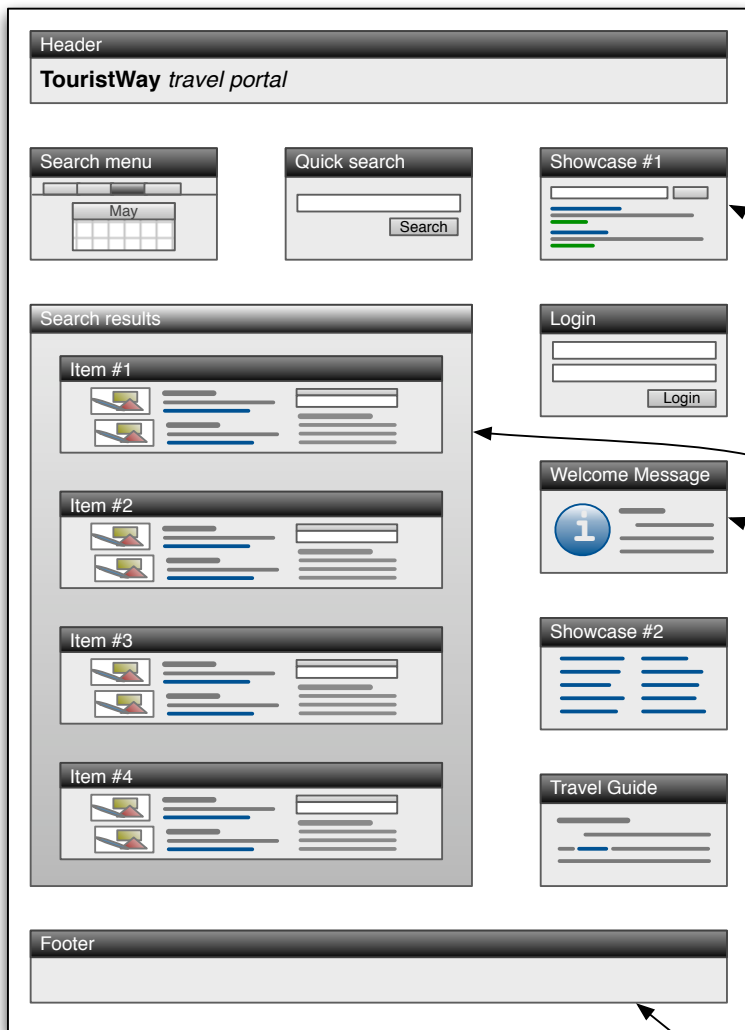
## **I.1 CUT-TO-THE-CHASE VERSION**

Here is how TouristWay foundation elements actually look like:

1. Templates folder contains all PAGES that TouristWay recalls to perform specific operations (i.e. display homepage, return search results, etc.)
2. PAGE contains the XML CONFIGURATION that list the active components for that page and their configuration.
3. PAGE file contains also the HTML code for the VIEWPORT (though there's a notable exception to this statement we will introduce later on).
4. VIEWPORT is a normal HTML structure (think of a static page) with placeholders to accommodate component's output
5. COMPONENTs are listed in the XML CONFIGURATION (see point 2), each one has its own template. Component's template is used to render the component output, thereby merging the data produced by component's logics with the appropriate graphic interface.

As the viewport is likely to repeat itself by big groups of pages (i.e. all secondary pages have similar layout, often just the main content change), it's possible to share the same viewport among pages by using what TouristWay defines a CANVAS.

This is the notable exception to the point 4: if XML configuration includes a CANVAS element then this will be merged with the current page to create a reusable viewport.



**Viewport**  
 contain the html required to draw this layout, the boxes in the diagram will appear as placeholders in the viewport. Placeholder will be filled by components output

**Component**  
 class=showcase  
 id=myshowcase1

**Attributes**  
 catalogue=packages  
 destination=current  
 usetemplate=specials.html

**Component**  
 class=search  
 id=hotels

**Attributes**  
 catalogue=accommodation  
 destination=current  
 dtcheckin=form.dtcheckin  
 dtcheckout=form.dtcheckout  
 usetemplate=hotelslist.html

**Component**  
 class=widget  
 id=welcomemessage

**Attributes**  
 usetemplate=w\_%-account.type-%.html

**Component**  
 class=widget  
 id=footer

**Attributes**  
 usetemplate=footer.html



## **2.0 HOW IT WORKS**

By calling any TouristWay url you request to run an XML script which contains a collection of component to be executed. TouristWay search for the script in the template folders, the name of the script is the same of the page you call:

```
http://www.mysite.com/twpub/index.cgi
```

will load:

```
/templates/default/RAW/index.html
```

From here TouristWay will make its magic based on the configuration of

- pages
- components
- templates

## **2.1 PAGES**

Once the page is loaded TouristWay will look into it expecting:

- the XML script
- the VIEWPORT (page layout)

**XML SCRIPT:** controls what is going to be available in the output, what type of users will be granted for access to the page and what kind of media format we want to output (XML,HTML).

```
<touristway>
  <settings>
    <security>EVERYBODY</security>
    <media>HTML</media>
  </settings>
  <components>
    <component id="hotels" class="search">
      <catalogue>accommodation</catalogue>
      <dtcheckin>%-global.dtcheckin-%</dtcheckin>
      <dtcheckout>%-global.dtcheckout-%</dtcheckout>
      <destination>%-form.loc-%</destination>
      <rating>%-form.cat-%</rating>
      <sortby>ratingdes desc</sortby>
      <itemsperpage>6</itemsperpage>
      <gotopage>%-form.pag-%</gotopage>
      <usetemplate>
        %-system.lngetemplatepath-%hlib/srchresult-a.html
      </usetemplate>

      <renderto>component_search</renderto>
      <retrieve>bizcard,fares,availability</retrieve>
      <datasetalias>
        <alias id="results">
          components.search.results
        </alias>
      </datasetalias>
    </component>
  </components>
</touristway>
```

**VIEWPORT (PAGE LAYOUT):** whatever is not wrapped in the <touristway> element is used to define the layout of the page output. Layout is nothing but plain HTML code plus some TouristWay placeholders that will accommodate the dynamic informations coming from the catalogue and from TouristWay logics.

Let me start remarking that any element or design choice in the page is completely up to you. In a matter of fact you can place whatever content in any way you like just by working on this HTML and on component definitions (see next chapter).

```
<touristway>
  <settings>
    <security>EVERYBODY</security>
    <media>HTML</media>
  </settings>
  <components>
    [... etc. etc. etc. ...]
  </components>
</touristway>

<!-- END OF XML SCRIPT - BEGINNING OF PAGE LAYOUT -->

<html>
<head>
  <title>%-global.projectname-%</title>
</head>

<body>
<div id="wrap">
  %-component_header-%
  <div id="menu">
    <ul>
      <li><a href="%-global.scriptname.home-%">Home</a></li>
    </ul>
  </div>

  <div id="content">
    <div id="home-sidelist">
      %-component_sidelist-%
    </div>

    %-component_homesearchform-%

    <div id="home-teaser">
      %-component_teaser-%
    </div>
    <div id="home-flyers">
      %-component_flyers1-%
    </div>
    <div id="home-flyers">
      %-component_flyers2-%
    </div>
    <div class="clear">&nbsp;</div>
  </div>

  %-component_footer-%
</div>
</body>
</html>
```

## 2.2 COMPONENTS

TouristWay does not assign any specific behavior to the page and this is what makes it extensible and most flexible: by loading the module “searcha.cgi” you don’t run a set of predefined tasks as usually happens, “searcha.cgi” represents just a hook to the XML script “searcha.html” which will contain high-level instructions about the required behavior of the page. You probably realized that adding a new different search result page is as easy as creating a new file.

The so-called “instructions” we just mentioned is nothing but a set of XML elements, each will recall a TouristWay component that will execute a specific task according to the attributes specified in the XML.

Let’s have a deeper dive into a typical component:

- component id and class
- component attributes
- component template
- TouristWay markup language

**ID AND CLASS:** each component belong to a class which identify the type of task the component is designed to accomplish.

Some of the main classes are:

- search
- showcase
- closeup
- pickbed
- pickseat
- booking

You can then consider to have more than one component for each class in the same page: for example you may want 3 showcases to appear in your page. For this reason to uniquely identify each component TouristWay requires you define both class and ID, the latter can be any name that make sense to you.

```
<touristway>
  <settings>
    <security>EVERYBODY</security>
    <media>HTML</media>
  </settings>
  <components>
    <component id="somehotels" class="search">
      [... etc. etc. etc. ...]
    </component>
    <component id="specialoffers" class="search">
      [... etc. etc. etc. ...]
    </component>
  </components>
</touristway>
```

**ATTRIBUTES:** components need parameter in order to properly do their job, therefore you are supposed to provide them at least the minimal informations required to execute the function you are asking for.

An example of the most typical parameters used in catalogue browsing:

- dtcheckin
- dtcheckout
- destination
- family
- rating
- catalogue

The value of any attribute can be explicit or parametric. It's explicit when you "sculpt" in the XML the parameter you want to pass:

```
<catalogue>accommodation</catalogue>
```

It's parametric when the value is a TouristWay variable, most likely coming from a form filled by the user:

```
<destination>%-form.location-%</destination>
```

Many attributes are not compulsory and act as a filter only if they are provided. In the example above if the form does not contain a key 'location' or its value is empty then the component will ignore it (in this case by not filtering by location) exactly as it was:

```
<destination></destination>
```

Here's a fairly easy example of search attributes set:

```
<component id="hotels" class="search">
  <catalogue>accommodation</catalogue>
  <dtcheckin>%-global.dtcheckin-%</dtcheckin>
  <dtcheckout>%-global.dtcheckout-%</dtcheckout>
  <destination>%-form.loc-%</destination>
  <rating>%-form.cat-%</rating>
  <itemsperpage>6</itemsperpage>
  <gotopage>%-form.pag-%</gotopage>
  <usetemplate>
    %-system.lngtemplatepath-%hlib/srchresult-a.html
  </usetemplate>
  <renderto>component_search</renderto>
  <retrieve>bizcard,fares,availability</retrieve>
  <datasetalias>
    <alias id="results">
      components.search.results
    </alias>
  </datasetalias>
</component>
```

**TEMPLATES:** once TouristWay component has done its job and retrieved the information according to your instructions and attribute values it's crucial to properly display them and make the result as pretty as possible by respecting your travel portal graphic design.

Attributes <renderto> and <usetemplate> makes the task of "skinning" your component output extremely easy yet powerful.

TouristWay will load the file specified at <usetemplate> and will make the component data model available anywhere in the file so that, just by placing around appropriate TouristWay markers you will end up exactly with the design you wanted, any single tag is under your control. Template will be most likely an HTML file however TouristWay uses templates also for plain text emails or special formats as CSV, Javascript code, etc.

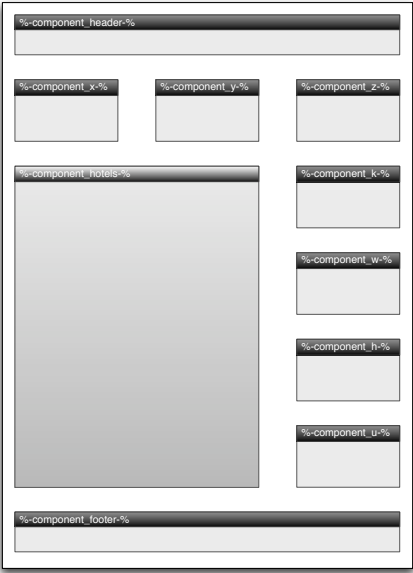
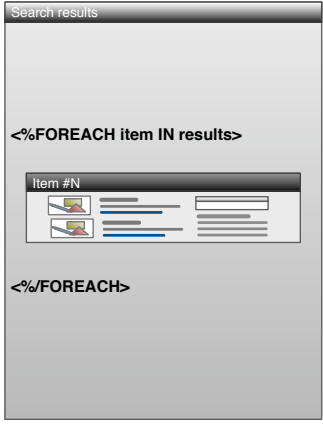
After merging your template and component data finally the output will be placed into the viewport exactly where you left the component-output placeholder. Component-output placeholder looks like:

`%-myplaceholdername-%`



or usually something more expressive like:

`%-component_mycomponentid-%`

Check the following diagram for a visual recap:

<p><b>1</b></p>		<p>Viewport gets loaded as described at 2.1</p> <p>Viewport is nothing but a skeleton.</p> <p>Note that often is useful to share the same viewport among several pages, in this case TouristWay offers the option to use an external file called CANVAS.</p> <p>Canvas can be defined in the &lt;settings&gt; element and provide a convenient way to import a viewport. Usually this option is widely popular for secondary pages as they usually comes with a similar layout.</p>
<p><b>2</b></p>		<p>Then one by one component gets executed. Their results (only data so far) get stored in TouristWay data model.</p> <p>&lt;usetemplate&gt; attribute is then used to retrieve component skin (template) where data get merged with graphic to create whatever look you planned to render.</p>



<p><b>3</b></p>		<p>Finally component output get transferred to the viewport.</p> <p>TouristWay will assign the component output to the variable named after the attribute <code>&lt;renderto&gt;</code>.</p> <p>This variable is supposed to match the placeholder name in the viewport.</p> <p><code>&lt;renderto&gt;myout&lt;/renderto&gt;</code> goes to <code>%-myout-%</code></p>
<p><b>4</b></p>		<p>Process starts again with the following component and continues until all components are processed.</p>

**MARKUP LANGUAGE:** so far we have been talking generally about “variables”, here’s a brief reference of the language TouristWay provides to place data into the templates.

**VARIABLES:** TouristWay data model is basically a big tree where components and the core procedures write results of their tasks. Tree can be traversed to fetch any value from the data model:

`%-account.type-%`

Formatting specifiers are available to perform most typical adjustments on values (truncate string, case, default values, etc.)

**CYCLES:** many informations are lists which requires to be looped to extract and display all their values. This common task is accomplished by `<%FOREACH>` statement:

```
<FOREACH product IN components.hotels.results>
  %-product.name-%
</%FOREACH>
```

Cycles can be nested to loop into inner lists or trees

```
<%FOREACH product IN components.hotels.results>
  %-product.name-%
  <%FOREACH feature IN product.facilities>
    %-feature.name-%
  </%FOREACH>
</%FOREACH>
```

**SEGMENTS:** react to content and variable characteristics by showing or hiding page elements is crucial to make your site look smart and user-friendly. TouristWay

provides the <%SHOWIF> statement which removes its content when a custom condition is not verified.

```
<%SHOWIF %-product.photos.gallery--format=scalar-%>
  <H1>Picture gallery</H1>
  <%FOREACH photo IN product.gallery>
    <IMG SRC="%-photo.content-%">
  </%FOREACH>
</%SHOWIF>
```

SHOWIF works with many type of conditions, the example above check the number of item in the list (if >0 then condition is true). More complicated conditional statements are welcome:

```
<%SHOWIF '%-account.type-%' eq 'affiliate'>
  Welcome travel agent!
</%SHOWIF>
```

```
<%SHOWIF
  ((%-product.photos.gallery--format=scalar-%) &&
  ('%-product.photos.gallery[0].family-%' eq 'OUTDR'))
>
  Take a Look at our outdoor facilities.
  [... etc. etc. etc. ...]
</%SHOWIF>
```

