Interactive Storyboarding Tool

storydoc

User Guide

Revision History

| Version	Date	Author	Description
0.0.1	12/23/2019	Masayuki Otoshi	Document Created

Table of Contents

1 Overview

This document is for system analysts and agile developers who are starting to realize user stories and create acceptance test scenarios by using storydoc tool.

1.1 What is storydoc?

The storydoc is an interactive storyboarding tool for analyzing user stories and defining detailed specifications. Scenarios captured from user stories are modeled in HTMLs using storydoc tags. A conceptual screen and possible actions on the page are defined in each HTML, and navigations among pages are represented by linking HTMLs.
Once you created the model, you can walk through the conceptual screens with your browser and see how the application will work. Also storydoc tool generates a storydoc document including User Story scenarios, Use Case diagram, Navigation map (State diagram) and Acceptance test scenarios. Those help you understand the specifications more and share with developers.
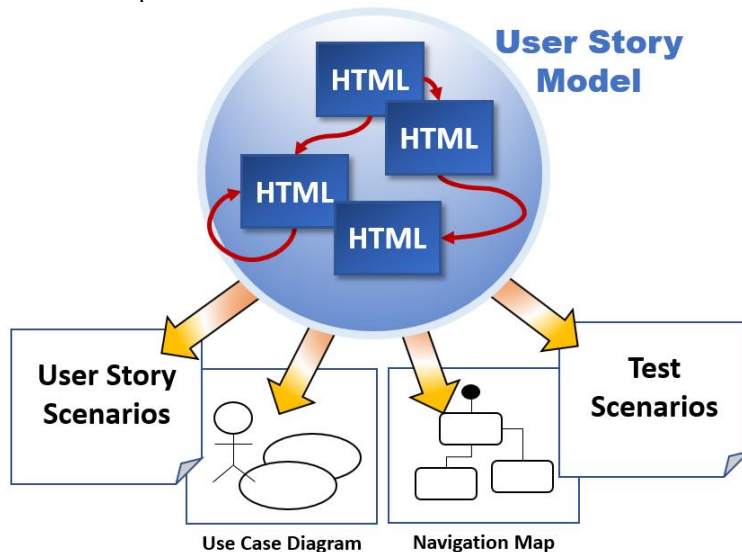
2 Installation

2.1 Prerequisites

Ensure that you have installed the following software on your machine:

- **Java SE Version 8** or above
  https://www.oracle.com/technetwork/java/javase/
- **Chrome browser**
  https://www.google.com/chrome/
- (Optional) **PlantUML Jar Version 1.2019.13** or above
  http://plantuml.com/
  You can also get the Jar file from storydoc site as a dependency.

2.2 Download storydoc

Download master ZIP file from the storydoc site below:

https://github.com/story-doc/js/archive/master.zip

Unzip it. We assumed that the required Jar files (storydoc.jar and plantuml.jar) are located in a same folder:

```
js-master/
├── doc/
├── samples/
├── plantuml.jar
├── README.md
├── storydoc.jar
└── storydoc.js
```

2.3 Storyboard sample story

That is all for the installation. As the next step, open sample HTMLs and verify if you can storyboard properly.

1. Go to samples/SearchStoryboard folder and open index.html with your browser.
2. You will be able to storyboard Product Search User Story. You will get the same experience as the storyboard from the URL below:

https://story-doc.github.io/SearchStoryboard/index.html

2.4 Generate storydoc HTML

Next, execute storydoc command and verify if the command works properly on your environment.

1. Go back to js-master folder.
2. Run the command below from a command line:
   (Make sure that plantuml.jar exists in the same folder with the storydoc.jar)

```
$ java -jar storydoc.jar samples/SearchStoryboard/index.html
```

A storydoc.html will be generated in the samples/SearchStoryboard folder. The generated HTML should be same as the one to be displayed from the URL below:

https://story-doc.github.io/SearchStoryboard/storydoc.html

3 Getting Started

This chapter introduces how to realize a user story with using the storydoc tool.

3.1 Product Search Story

We here use a sample User Story, Product Search that user searches for a product by a model name on customer's website. When a user enters a model name and submits the form, the system will display the information of the product matched with the model name entered.
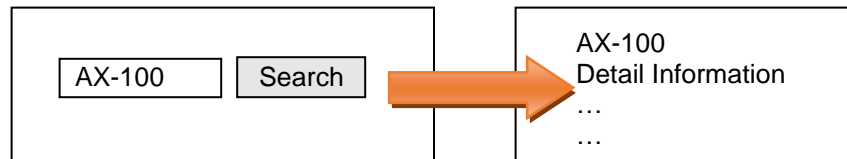
In this chapter, we receive a user story and acceptance criteria from a customer and realize the story through analysis and design process. The goal of this process is to create Acceptance Test Scenarios. But intermediate documents (Storyboard, Use Case Diagram, Navigation Map) are also generated in the process of analyzing and designing.
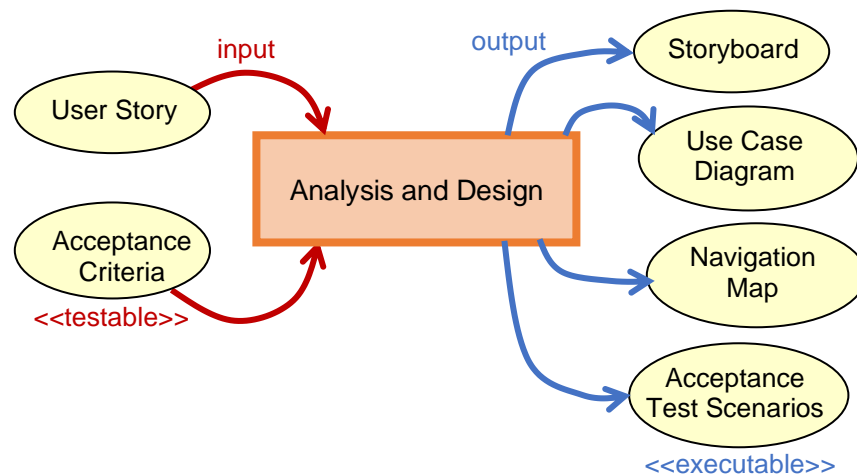
3.2 Define User Story

User Story is supposed to be defined by customer. They trawl requirements through conversations with stakeholders and write down the core requirements as user story and acceptance criteria. Normally they are described using the following template:

User Story

As a *<ROLE>*, **I want** *<GOAL>* **so that** *<REASON>*.

Acceptance Criteria

Given *<PRECONDITION>*
When *<ACTION>*
Then *<EXPECTED OUTCOME>*

In this example, we supposed to get the following user story and acceptance criteria from customer:

User Story

As a website user,
I want to search products by a model name
so that I can find the information on the product.

Acceptance Criteria

Given I open the website
When I search by model name
Then the system shows the product information

Since user story and acceptance criteria are generated as a result of requirements gathering, detail specifications (user interface, etc) are not defined yet at this moment. By leaving ambiguity on purpose, they can focus on core requirements and consider what they really need. Thus, HTML element names and actions tightly related to HTML elements are not described in the acceptance criteria. For example, "Click Search button" will give a concrete idea to implement a HTML button with a label "Search" on it. But there are some other ways to submit the form. In this phase, the customer should focus on what they need. How to implement should be discussed in later phase.

3.3   Find Screens and Actions

Once the customer has clarified what they really need and defined the user story and acceptance criteria, system analyst (or developer) can start to analyze the story.
The first step to do is finding screens and user (or system) actions in the story. In this example, from the line 2 of the acceptance criteria, we see that the user enters a model name. So, we can guess there is a text field to enter the model name on a search form. And the user action will make page change to product information page (Product Detail page).
Through the above story analysis, we found two screens, Search Form and Product detail pages. And there is a user action on the Search Form and it opens the product detail page.
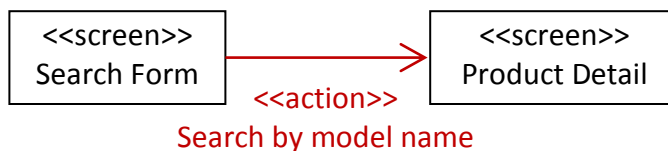
3.4   Initial Storyboard

After you found screens and actions, you can make a minimal storyboard. The storyboard requires an entry HTML for User Story description. We usually name the file as index.html.
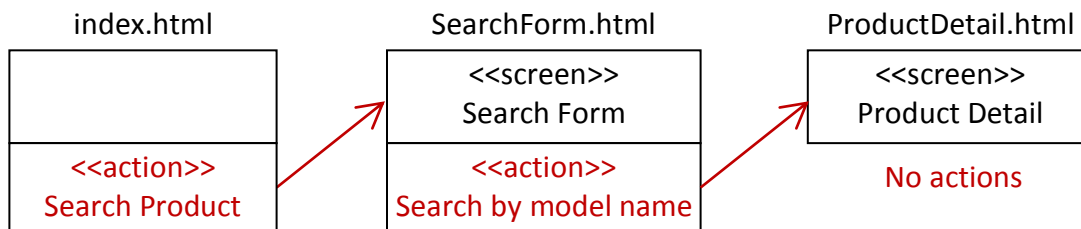
The index.html has a <action> tag linked to the first page of this story. The first page also has a <action> and it is linked to the final page. These HTMLs can be created by using storydoc <action> tags as shown below:

index.html

```
<script src="https://cdn.jsdelivr.net/gh/story-doc/js@master/storydoc.js"></script>

<action name="Product Search"
     link="SearchForm.html">
As a <role>website user</role>,
I want to search products by a model name
so that I can find the information on the product.
</action>
```

SearchForm.html

```
<script src="https://cdn.jsdelivr.net/gh/story-doc/js@master/storydoc.js"></script>

<action name="I search by model name"
     link="ProductDetail.html"/>
```

ProductDetail.html

```
<script src="https://cdn.jsdelivr.net/gh/story-doc/js@master/storydoc.js"></script>
```

As shown above, <action> is a minimal requirement for storyboarding. But that is not enough for test scenario. I here show the acceptance criteria defined by customer:

```
Given I open the website
When I search by model name
Then the system shows the product information
```

The 2nd line (When clause) is defined in SearchForm.html as the <action> but 1st and 3rd lines are missing. Those Given and Then clauses can be added by <test> tag.
The 1st line occurs before the 2nd line (SearchForm.html), so we can add a <test> tag in index.html for describing the 1st line.

index.html

```
<script src="https://cdn.jsdelivr.net/gh/story-doc/js@master/storydoc.js"></script>

<action name="Product Search"
     link="SearchForm.html">
As a <role>website user</role>,
I want to search products by a model name
so that I can find the information on the product.
<test>I open the website</test>
</action>
```

And the 3rd line occurs after the 2nd line (SearchForm.html), so we can add a <test> tag in ProductDetail.html for describing the 3rd line.

ProductDetail.html

```
<script src="https://cdn.jsdelivr.net/gh/story-doc/js@master/storydoc.js"></script>
<test>the system shows the product information</test>
```
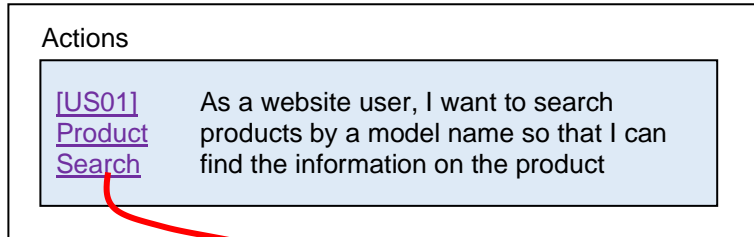
Now, in the storyboard, we see all information defined in user story and acceptance criteria. Let's walk through the HTMLs from the entry page, index.html. Click the URL below, and you will see the entry page with your default browser.

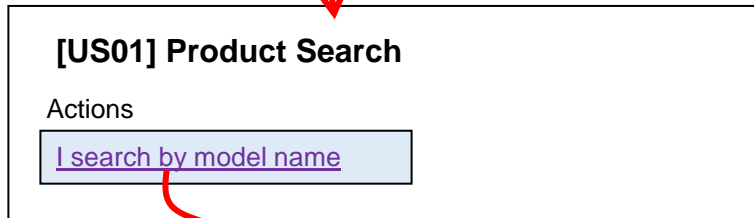https://story-doc.github.io/SearchStory/index.html

The entry page shows a list of stories in Actions box. Since we have only one story, only one <action> link is there. When you click the link, storyboarding begins. It shows the first page, SarchForm which has a link saying that "I search by model name". This represents a user action that enters a keyword and submits the form. When you click the link, it shows the next page, which is the final page, ProductDetail. The page says no actions that represents the end of story.

index.html

Actions

[US01] Product Search   As a website user, I want to search products by a model name so that I can find the information on the product

SearchForm.html

[US01] Product Search

Actions

I search by model name

ProductDetail.html

[US01] I search by model name

Actions

No actions (end of story)
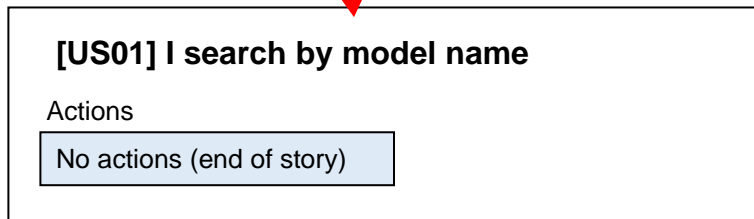
The above storyboard is hard to understand what is going on, because we have no screens yet. That also means that it is difficult to understand requirements described by sentences. We will add screens on it in later section, but let's generate documents from the storyboard, anyway. Go to the story folder where the index.html exists. And ensure that required jar files (sorydoc.jar and plantuml.jar) are in your CLASSPATH. Then run the command below from a command line:

```
C:> java -jar storydoc.jar index.html
```

It will generate a HTML named "storydoc.html" and a text "storytest.txt" in the same folder. You can browse the storydoc document from the URL below:

https://story-doc.github.io/SearchStory/storydoc.html

The document contains:

- **UseCase diagram** that represents relations among stories and user roles.
- **Main scenario** that represents steps of user and system actions.
- **Navigation map** (State diagram) that represents action flows among screens.

- • **Test scenario** that represents test steps (same as acceptance criteria at this moment)

Once you created HTMLs as a story model, storydoc generates those artifacts automatically from the model. Thus, you do not have to update those documentations individually whenever you or customer refine requirements or detailed specifications.

3.5   Add Screens

You may feel it is difficult to understand the initial storyboard due to missing screens. Adding screens will help readers understand what is happening in each step. To do so, you can add a <screen> tag and describe a conceptual screen with using simple HTML tags.
The entry page, index.html, has no screen, but we can add a <screen> tag for the title of the storydoc document.

index.html
```
<script src="https://cdn.jsdelivr.net/gh/story-doc/js@master/storydoc.js"></script>

<screen>
  <h1>Product search by model name</h1>
</screen>

<action name="Product Search"
      link="SearchForm.html">
As a <role>website user</role>,
I want to search products by a model name
so that I can find the information on the product.
<test>I open the website</test>
</action>
```

We can guess there are, at least, an input field and a Search button on the SearchForm page, the conceptual screen can be described with a <input> and a <button> tags as shown below:

SearchForm.html
```
<script src="https://cdn.jsdelivr.net/gh/story-doc/js@master/storydoc.js"></script>

<screen>
  <input> 
  <button>Search</button>
</screen>

<action name="I search by model name"
      link="ProductDetail.html"/>
```

The final page, ProductDetail, shows the information on the product. But we do not care what information is displayed, so you can simply specify "Product Detail" in the <screen> tag.

ProductDetail.html
```
<script src="https://cdn.jsdelivr.net/gh/story-doc/js@master/storydoc.js"></script>

<screen>
  Peoduct Detail
</screen>

<test>the system shows the product information</test>
```

Then, you can create the storydoc document again. Here is the link to the revised storydoc document including conceptual screens.

https://story-doc.github.io/SearchInitialStoryboard/index.html

index.html

Product search by model name

Actions

[US01]
Product
Search

As a website user, I want to search products by a model name so that I can find the information on the product

SearchForm.html

[US01] Product Search

Search

Actions

I search by model name

ProductDetail.html

[US01] I search by model name

Product Detail

Actions

No actions (end of story)
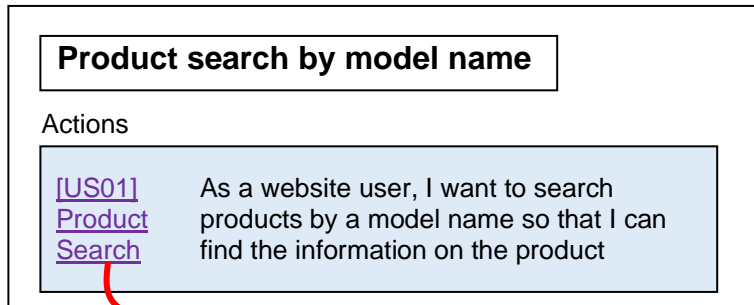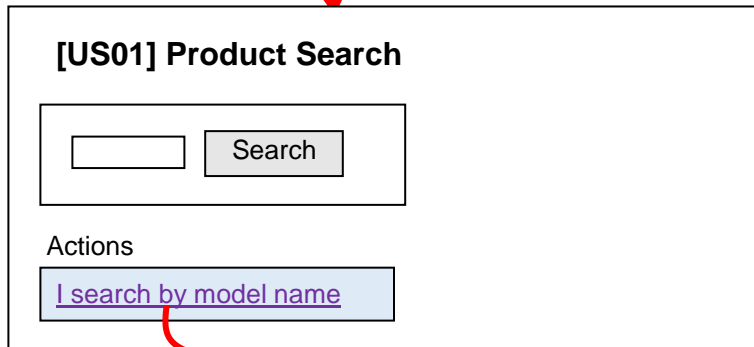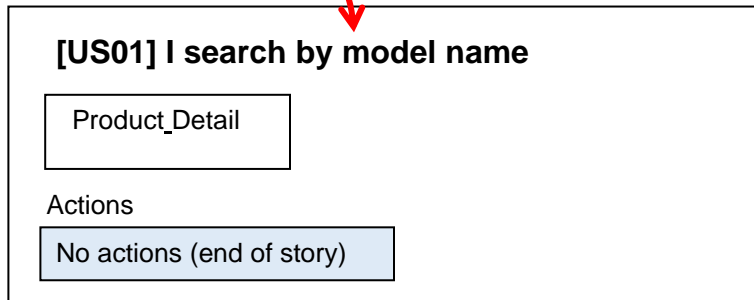
NOTE:
We need to emphasize that you should not draw a concrete screen image in the <screen> like UI prototype. The <screen> is for helping to understand how the action is made on the page. Hence, you need to focus on HTML elements related to the action and ignore other elements.

Once you updated the HTMLs with conceptual screens, run the storydoc command again to update the storycoc document:

```
C:> java -jar storydoc.jar index.html
```

It is also available to see the updated storydoc HTML from the URL below:

https://story-doc.github.io/SearchInitialStoryboard/storydoc.html

3.6   Find Alternative scenarios

So far, we have focused on the main scenario defined by customer in the story. But we could find more scenarios when we started exploring the story in detail. Since customer needed to focus on core requirements, only typical course taken by most of users is considered. However, in analysis phase, we need to find other courses that could happen by possible operations on the page, as much as possible in order to clearly define the specifications.
In this example, the main scenario defines the case that the system finds only one product with the given model name. But what if it finds multiple products or what if the model name does not match any products? As a system analyst or developer, we need to define how the application works in those cases before starting the coding.
As an example of adding an alternative scenario, we here add a scenario that the entered model name matches multiple products.

Alternative scenario:

When I enter a partial model name
Then the system shows a list of products
When I select a product
Then the system shows the product information

In this scenario, user will see a new screen, SearchResults, and they need to choose a product from a list of products matched.
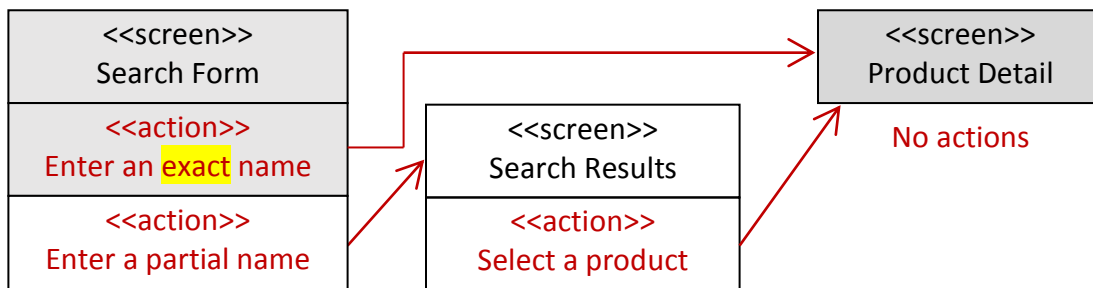
As you can see the above screen flow, we need to add and change the following items:

- Change existing action name to say "Enter an exact model name" from "Search by model name" in SearchForm.html (to distinguish from a new action).
- Add a new action "Enter a partial model name" in SearchForm.html
- Add a new screen "SearchResults.html" and add an action "Select a product".

The code below shows the changes in RED:

SearchForm.html

```
<script src="https://cdn.jsdelivr.net/gh/story-doc/js@master/storydoc.js"></script>

<screen>
  <input> 
  <button>Search</button>
```

```
</screen>

<action name="I enter an exact model name"
      link="ProductDetail.html"/>

<action name="I enter a partial model name"
      link="SearchResults.html"/>
```

SearchResults.html

```
<script src="https://cdn.jsdelivr.net/gh/story-doc/js@master/storydoc.js"></script>

<screen>
  <u>Product 1</u><br>
  <u>Product 2</u><br>
  <u>Product 3</u><br>
</screen>

<action name="I select a product"
      link="ProductDetail.html"/>
```

You can open the index.html and walk though storyboard. Also generate storydoc HTML document again and you will see that the alternative scenario is newly added in there.

3.7   Find Exception scenarios

As we found an alternative scenario in previous section, we could find an exception scenario. An exception scenario represents a course in error and you can define by using <exception> tag. An exception is technically a sort of alternative scenario, so you can define it with a <action> tag if you want to.
The difference is that the exception is categorized into Exception Scenarios in a storydoc HTML and test scenarios.
.As an example of exceptions, we add a new scenario below:

Exception scenario:

```
When I search with an empty keyword
Then the system shows an error message
```

The scenario represents that the user submits the search form without entering a model name, and the system displays an error. On the error page, there is a button to go back to the SearchForm so that the user can try again.
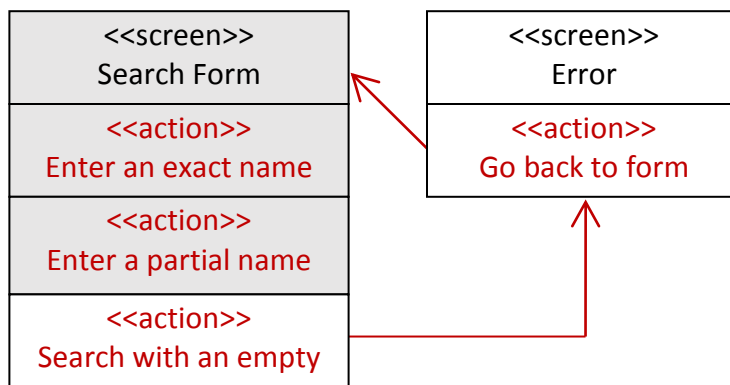
As you can see the above screen flow, we need to add the following items:

- Add a new action "Search with an empty keyword" in SearchForm.html
- Create a new HTML "Error.html"
- Add a new action "Go back to the search from" in Error.html

The changes can be seen in the following code in RED:

SearchForm.html

```
<script src="https://cdn.jsdelivr.net/gh/story-doc/js@master/storydoc.js"></script>

<screen>
  <input> 
  <button>Search</button>
</screen>

<action name="I enter an exact model name"
      link="ProductDetail.html"/>

<action name="I enter a partial model name"
      link="SearchResults.html"/>

<exception name="I search with an empty keyword"
      link="Error.html"/>
```

Error.html

```
<script src="https://cdn.jsdelivr.net/gh/story-doc/js@master/storydoc.js"></script>

<screen>
  Invalid model name.
  Please try again.<br>
  <button>OK</button>
</screen>

<action name="I go back to the search form"
      link="SearchForm.html"/>
```

Alternative and exception scenarios were added in the story, so we have now three scenarios including Main scenario. Let's check to see those in storydoc HTML.
Run the storydoc command again and open the storydoc HTML generated:

```
C:> java -jar storydoc.jar index.html
```

The storydoc HTML is available to see from this URL:

```
https://story-doc.github.io/SearchStoryboard/storydoc.html
```

In the document, you will see the following navigation map. The diagram shows how the three scenarios (main, alternative and exception) transit among screens.
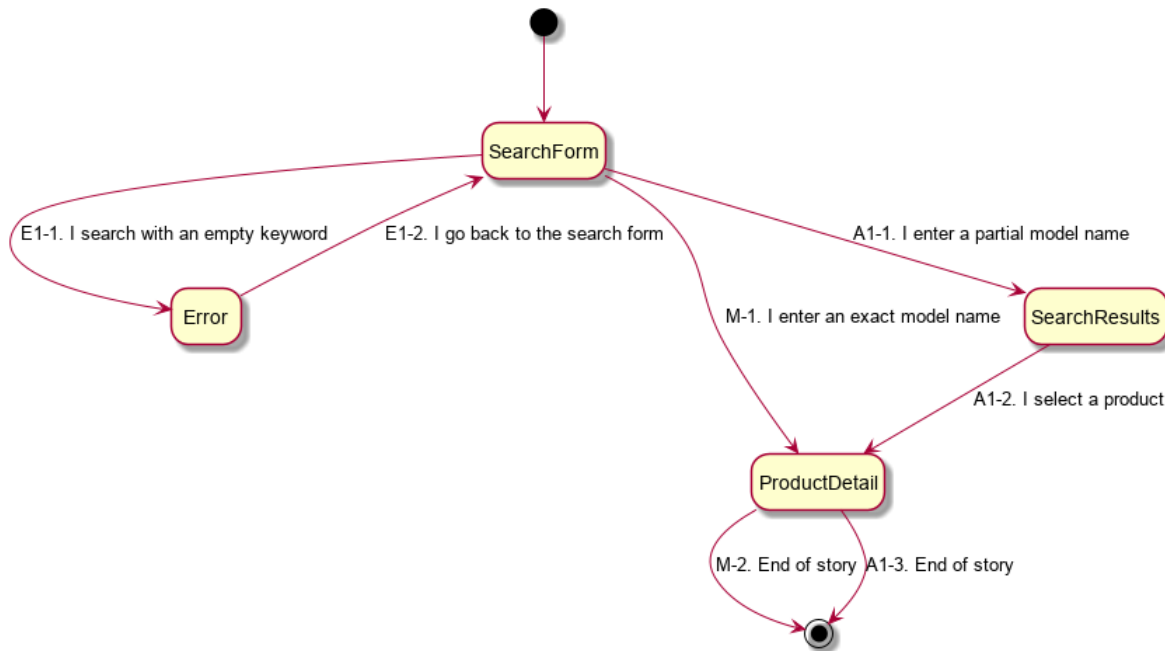
3.8 Create Acceptance Test Scenarios

In previous sections, we added actions for Alternative and Exception scenarios. Those will be "When" clauses in acceptance test scenarios but "Then" clauses are still missing.

If you run storydoc command and generate storydoc document, you will see the following test scenarios in it:

```
Story: [US01] Product Search

Main Scenario:
     Given I open the website
  M-1. When I enter an exact model name
     Then the system shows the product information

Alternative Scenario 1:
     Given I open the website
  A1-1. When I enter a partial model name
  A1-2. And I select a product
     Then the system shows the product information

Exception Scenario 1:
     Given I open the website
  E1-1. When I search with an empty keyword
  E1-2. And I go back to the search form
 #E1-3. Go back to M-1
```

We are expecting that A1-1 action makes a redirect to Search Results page but assertion is not defined. We can add an assertion to verify a list of products are displayed. Likewise, E1-1 action should display an error message for the empty keyword, but no assertion is defined.

In this section, we add <test> tags to assert the results of the actions. Below shows the assertions expected to be added for A1-1 and E1-1 in RED:

```
Story: [US01] Product Search

Main Scenario:
      Given I open the website
  M-1. When I enter an exact model name
      Then the system shows the product information

Alternative Scenario 1:
      Given I open the website
  A1-1. When I enter a partial model name
      Then the system shows a list of products
  A1-2. When I select a product
      Then the system shows the product information

Exception Scenario 1:
      Given I open the website
  E1-1. When I search with an empty keyword
      Then the system shows an error message
  E1-2. When I go back to the search form
 #E1-3. Go back to M-1
```

At first, let's add an assertion for A1-1. As you can see in the above scenarios, we want to add the assertion after the A1-1 action. To do so, we have two options:

- Add a <test> tag in the <action> tag of the A1-1 (SearchForm.html).

```
<action name="I enter a partial model name"
      link="SearchResults.html">
  <test>the system shows a list of products</test>
</action>
```

- Add a <test> tag in the next page (SearchResults.html) of the A1-1 action.

```
<screen>
  <u>Product 1</u><br>
  <u>Product 2</u><br>
  <u>Product 3</u><br>
</screen>

<test>the system shows a list of products</test>
```

Both ways result to generate exact same test scenario in Alternative Scenario 1. The difference is that the first option adds the assertion after the specific action (in this case, A1-1) only, while the second option adds the assertion for all actions visit the page.
In this example, the assertion can be applied on all cases that visits the SearchResults page.
So, the second option would be better. Below shows the SearchResult.html with the assertion:

SearchResults.html

```
<script src="https://cdn.jsdelivr.net/gh/story-doc/js@master/storydoc.js"></script>

<screen>
  <u>Product 1</u><br>
  <u>Product 2</u><br>
  <u>Product 3</u><br>
</screen>

<test>the system shows a list of products</test>
```

```
<action name="I select a product"
    link="ProductDetail.html"/>
```

Likewise, we can add one more assertion in Error.html. The error page is displayed when user entered an invalid model name. As an assertion for the action, we can add the <test> tag in Error.html:

Error.html

```
<script src="storydoc.js"></script>

<screen>
  Invalid model name.
  Please try again.<br>
  <button>OK</button>
</screen>

<test>the system shows an error message</test>

<action name="I go back to the search form"
    link="SearchForm.html"/>
```

Run storyboard tool to see the acceptance test scenarios with additional assertions.

```
C:> java -jar storydoc.jar index.html
```

The command will generate the storytest.txt. You can see the text from the URL below:

https://raw.githubusercontent.com/story-doc/js/master/samples/SearchStoryboard/storytest.txt

The test scenarios should be executable, unlike acceptance criteria (testable) created based on core requirements. So each step must be defined based on user operations. That is why we need to analyze stories by interactive storyboarding and clearly define specifications in detail.

4   Usages of Storydoc tags

This chapter describes how to use Storydoc tags with showing examples.

4.1   <action>

The <action> is a tag to describe a user or system action in HTML. For example, user logs in the system, the system displays search results.

4.1.1   link

By taking an action on a page and the action navigates the user to another page. The navigation is defined by using link attribute of <action> tag. Suppose we are on a Search Form page and search a keyword. Then the system displays the Search Results page. The navigation from the Search Form to Results page can be defined by the <action> tag below:

SearchForm.html

```
<screen>Search Form</screen>
<action name="I search for a keyword" link="SearchResults.html"/>
```

SearchResults.html

```
<screen>Search Results</screen>
```


In the Search Results page, the user can select a result to see the detail information. To represent the action on the results page, we can add a new <action> and a new HTML.

SearchResults.html (Modified)

```
<screen>Search Results</screen>
<action name="I select an item" link="ItemDetail.html"/>
```

ItemDetail.html (New)

```
<screen>Item Detail</screen>
```
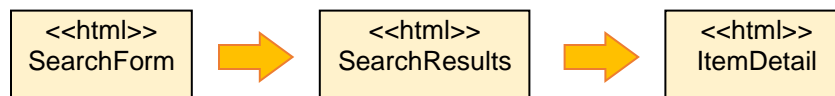

4.1.2   Alternative course

By defining another <action> on the same page, an alternative course can be added. In the previous sub section, we defined a main scenario to search and select an item, which most of users will take. But some of the users may take another action on the page. Suppose there is a link to open an advanced settings page. You can add another <action> as shown below:

SearchForm.html

```
<screen>Search Form</screen>
```

```
<action name="I search for a keyword" link="SearchResults.html"/>
<action name="I open advanced settings" link="Settings.html"/>
```
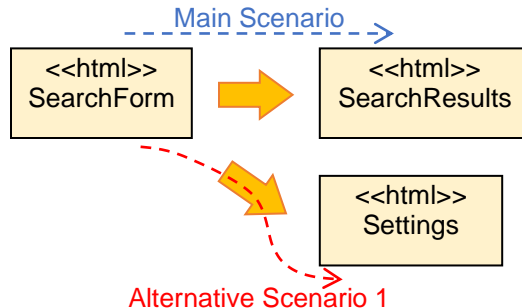
The first <action> will be an action of Main Scenario and the second <action> will be an Alternative Scenario 1.

4.1.3 Error course

Similar to the alternative course, <exception> tag can be used to add an error course. If the Search Form shows an error page for an invalid keyword, you can add a <exception> tag as shown below:

SearchForm.html

```
<screen>Search Form</screen>
<action name="I search for a keyword" link="SearchResults.html"/>
<action name="I open advanced settings" link="Settings.html"/>
<exception name="I enter an invalid keyword" link="Error.html"/>
```

The first and second <action> tags will be actions of Main Scenario and an Alternative Scenario 1, respectively. And the <exception> will be an action of Exception Scenario 1.

4.1.4 Use Case Diagram

The <action> tag in the first HTML is also used to generate a Use Case diagram. The first HTML is special and for describing a list of user stories, while the rest of HTMLs are for describing screens with actions. For that reason, a <action> tag in the first HTML represents a User Story, and you can also define relations among User Stories in the <action>.
Suppose customer defined the User Story below:

```
As a website user
I want to search products by a model name
so that I can find the information of the product.
```

You can define the story using the <action> below:

```
<action name="Product Search">
As a <role>website user</role>,
I want to search products by a model name
so that I can find the information on the product.
</action>
```

Storydoc tool generates the following Use Case Diagram from the <action>:
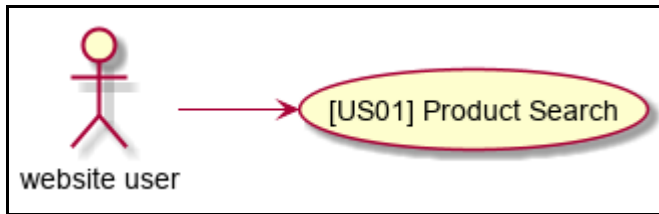
The <action> also accepts **include**, **extend** and **generalization** attributes. If the User Story includes other story, for example, US02, specify the User Story ID (US02) into the include attribute.

```
<action id="US01" name="Product Search" include="US02"/>
<action id="US02" name="Send Email"/>
```
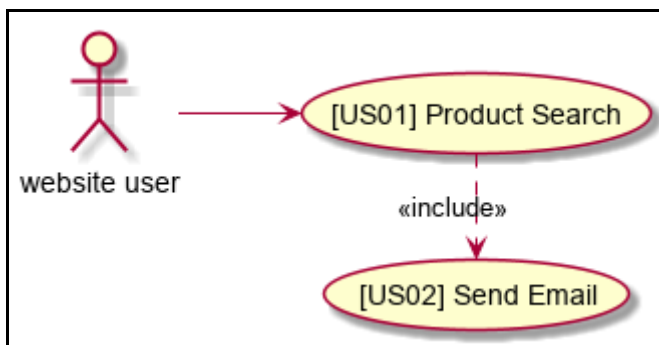

If the User Story has an alternative story, for example, "US03", specify the User Story ID (US03) to extend attribute.

```
<action id="US01" name="Product Search" extend="US03"/>
<action id="US03" name="Site Search"/>
```
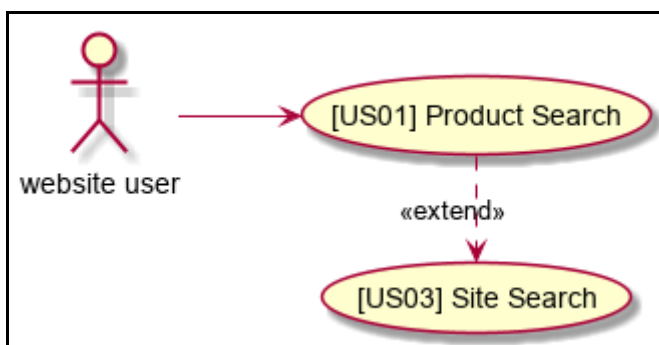

If the User Story is specialized from another story, for example, "US05", specify the User Story ID (US05) to generalization attribute.

```
<action id="US01" name="Product Search" generalization="US04"/>
<action id="US04" name="General Search"/>
```
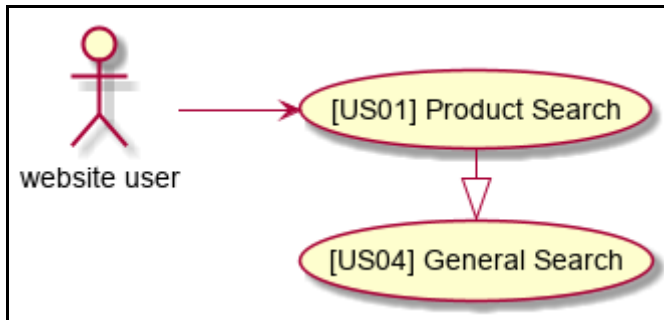
The include, extend and generalization attributes accept multiple User Story IDs. If you want to, for example, include US02, US03 and US04 from US01, specify the IDs in comma-separated format as shown below:

```
<action id="US01" name="Product Search" include="US02, US03, US04"/>
<action id="US02" name="Send Email"/>
<action id="US03" name="Site Search"/>
<action id="US04" name="General Search"/>
```

4.2 <screen>

The <screen> is a tag to draw a conceptual screen image with using HTML tags and images. For example, the <screen> below represents a Search Form.

SearchForm.html

```
<screen>
  <input> <button>Search</button>
</screen>
```

It will generate a simple form containing a text field and a Search button.

It may look poor compared to screen prototype decorated with CSS. But the conceptual screen must be as simple as possible. Even though there are some other elements displayed on the page, we must focus on elements which makes actions we want to capture. Header, footer and other unrelated elements must be ignored. We incrementally refine requirements and designs. Hence, the conceptual screens are also updated whenever refined.

4.3   <note>

The <note> is a tag to add a short description on that HTML. It is displayed under <screen>, so the node is usually mentioned about the screen.

SearchForm.html

```
<screen>
  <input> <button>Search</button>
</screen>
<note>The text box displays a placeholder to say "Please enter a keyword"</note>
```

The node will be displayed the screen when storyboarding:
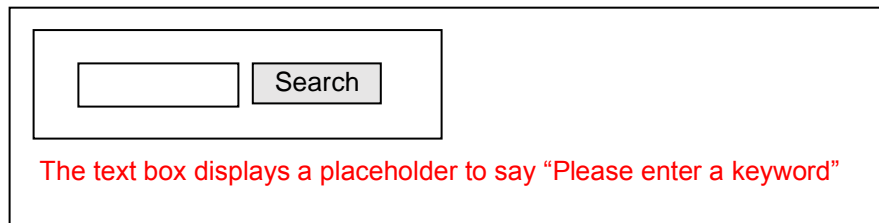
4.4   <test>

After you added screens and actions in HTMLs, you will see test scenarios in storydoc document, but they are not completed. You need to add assertions and test data etc. This chapter shows how to add <test> tags to add assertion steps and test data examples in test scenarios.

The <test> is a tag to describe assertions and test data, and the description is not appeared while storyboarding. It only added in test scenarios.

4.4.1   Root Level

There are two places where <test> tag can be described. One of the places is at root level in HTML file. It is the place for all actions that visit the HTML. For example, there are two pages and each page has a <test> tag as shown below:

Page1.html

```
<screen>Page 1</screen>
<test>the system shows Page 1</test>
<action name="I open Page 2" link="Page2.html"/>
```

Page2.html

```
<screen>Page 2</screen>
<test>the system shows Page 2</test>
```

From the above HTMLs, the test scenario below is generated by storydoc tool.

```
Given the system shows Page 1
When I open Page 2
Then the system shows Page 2
```

4.4.2   Action Level

The other place where you can use <test> tag is in a <action> tag. As a part of body in <action>, you can describe <test> tags. The test description is added after the action description (with When clause) as shown below:

Page1.html

```
<screen>Page 1</screen>
<test>the system shows Page 1</test>
<action name="I open Page 2" link="Page2.html">
  <test>the system closes Page 1</test>
</action>
```

From the above HTML, the test scenario below is generated by storydoc tool.

```
Given the system shows Page 1
When I open Page 2
Then the system closes Page 1
And the system shows Page 2
```

If you want to insert the test description before the action, set **insert** attribute to **before**. The description will be added before the action description "I open Page 2".

Page1.html

```
<screen>Page 1</screen>
<test>the system shows Page 1</test>
<action name="I open Page 2" link="Page2.html">
  <test insert="before">Page 2 link is visible</test>
  <test>the system closes Page 1</test>
</action>
```

From the above HTMLs, the test scenario below is generated by storydoc tool.

```
Given the system shows Page 1
And Page 2 link is visible
When I open Page 2
Then the system closes Page 1
And the system shows Page 2
```

4.4.3   Override Action name

By default, the name attribute value of <action> is used as a When clause in test scenario. But you may want to use different description from the action name. If you have a test specific description for the action, specify test attribute in the <action>.

Page1.html

```
<screen>Page 1</screen>
<action name="Click Page 2 link"
        test="I open Page 2"
        link="Page2.html"/>
```

From the above HTML, the test scenario below is generated by storydoc tool. The name "Click Page 2 link" is shown on storyboarding and steps in storydoc document. The test attribute will be used for test scenario document.

> When I open Page 2
> Then the system shows Page 2

4.4.4   Examples

If you have concrete input data and expected values for the test, use Examples.

Page1.html

> \<screen\>\<input\>\<button\>Submit\</button\>\</screen\>
> \<action name="I enter a \<Number\>" link="Page2.html"/\>

Page2.html

> \<screen\>Doubled number\</screen\>
> \<test\>the system shows a \<Doubled number\>\</test\>
> \<test\>
> Examples:
> | Number | Doubled number |
> | 1 | 2 |
> | 20 | 40 |
> | -3 | -6 |
> \</test\>

From the above HTMLs, the test scenario below is generated by storydoc tool.

> When I enter a \<Number\>
> Then the system shows a \<Doubled number\>
>   Examples:
> | Number | Doubled number |
> | 1 | 2 |
> | 20 | 40 |
> | -3 | -6 |

The above test scenario represents to be executed three times with the value pairs:

(\<Number\>, \<Doubled number\>) = (1, 2), (20, 40), (-3, -6)

4.4.5   Given-When-Then

Given, When, Then and And keywords are automatically added in each step in the test scenarios.

Given is added on \<test\> tags appearing before the first \<action\>, and represents preconditions or preparation steps.
When is added on \<action\> tags, and represents a trigger to make a user or system action.
Then is added on \<test\> tags appearing after the first \<action\>, and represents a assertion that verify results of an action or test data examples.

If you want to use different clause name from the default name, specify **gwt** attribute in \<test\> or \<action\> tag.

23

storydoc
User Guide

Page1.html

```
<screen>Page 1</screen>
<action name="I open Page 2" link="Page2.html"/>
```

Page2.html

```
<screen>Page 2</screen>
<action name="I cannot browse Page 2"/>
```

From the above HTMLs, the test scenario below is generated by storydoc tool.

```
When I open Page 2
And I cannot browse Page 2
```

But you want to say "Then" for the second <action> instead of "And". To change the clause, specify gwt attribute with a clause name (Given, When or Then).

Page2.html

```
<screen>Page 2</screen>
<action gwt="Then" name="I cannot browse Page 2"/>
```

Now you will see the result below:

```
When I open Page 2
Then I cannot browse Page 2
```

5 Storydoc Tag Reference

This chapter lists all the Storydoc tags.

5.1 <action>

The <action> tag creates a hyperlink to jump to another page. The tag represents a possible user (or system) action which can be taken on the page.
If you describe multiple <action> tags in a state HTML, the first <action> represents a step in Main Scenario and the other <action> tags represent Alternative Scenarios. To describe an exceptional action, use <exception> tag.

Example 1: Make a link on a state HTML.

```
<action name="Search for a keyword"
        link="SearchResults.html">
User enters a keyword and submits the form.
</action>
```

This will produce the following result:

Actions

Search for a keyword	User enters a keyword and submits the form.

On the root HTML, the <action> is used to define a user story. Story specific attributes (role, include, extend, generalization) and <role> tag is available to use in it for drawing a Use Case diagram.

Example 2: Define a User Story in the root HTML.

```
<action id="US01"
        name="Log in"
        link="Browser.html"
        include="US02">
As an <role>End User</role>,
I want to log in System
so that I can see protected pages.
</action>
```

This tag will produce the following result:

Actions

[US01] log in	As an End User, I want to log in System so that I can see protected pages.

The tag will also produce the following use case diagram:

Common Attributes for Page and User Story:
The following attributes can be described in both Story and State HTMLs.

Attribute Name	Value	Description	Required
name	Action step description	Specifies a short description of a possible user or system action	Yes
test	Test step description	Specifies a test description to be outputted into Acceptance Criteria document.	No

		If not specified, the name attribute value is used.	
id	User story ID	Specifies user story ID. In User Story HTML, this is recommended to be specified. In State HTML, this is needed when the action jumps to another user story.	No
link	HTML file path	Specifies a file path of the HTML to be shown next.	No
onclick	JavaScript code	Specifies JavaScript code to be executed when the tag is clicked.	No
style	HTML style value	Specifies HTML element style	No

Attributes for User Story only:
The attributes below are effective in User Story (root) HTML only.

Attribute Name	Value	Description	Required
role	User role name	Specifies an user role name such as User, Admin, External System.	No
extend	Comma separated User story IDs	Specifies User story IDs of the alternative stories, if this story have alternative stories.	No
include	Comma separated User story IDs	Specifies User story IDs of the other stories used from this story, if this story uses other stories.	No
generalization	Comma separated User story IDs	Specified User Story ID of the parent story, if this story is specialized based on a generic (parent) story.	No

5.2 <role>

The <role> tag is only effective in <action> tag in User Story (root) HTML. It marks a part of description of the <action> tag as an user role.

Example 1: Define a role 'End User' in the root HTML.

```
<action id="US01"
        name="Log in"
        link="Browser.html">
As an <role>End User</role>,
I want to log in System
so that I can see protected pages.
</action>
```

This tag will produce the following result:

Actions	
[US01] log in	As an End User, I want to log in System so that I can see protected pages.

The tag will also produce the following use case diagram:

The <role> accepts extend attribute to specify the parent user role.

Example 2: Define a parent user role.
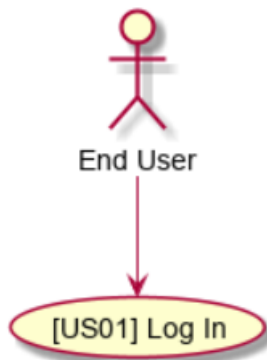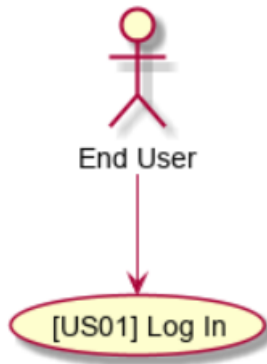
```
<action id="US01"
        name="Log in"
        link="Browser.html">
As an <role extend="Parent User">End User</role>,
I want to log in System
so that I can see protected pages.
</action>
```

The tag will produce the following use case diagram:

Attributes:

Attribute Name	Value	Description	Required
extend	User role name	Specifies a name of parent user role.	No

5.3 <exception>

Like <action> tag, the <exception> tag is also used to describe a possible user (or system) action. And it is displayed in Actions box on the state HTML. Hence, the <exception> tag accepts exact same attributes as the <action>.
The difference is the <exception> is for describing an Exception scenario, and the <action> is for describing a Main or Alternative Scenario.
If the user or system action will cause an error, use <exception> instead of <action>.

Example 1: Define an action and an exception on a state HTML.

```
<action name="Upload a CSV"
     link="UploadProgress.html">
User selects a valid CSV file and System starts uploading.
</action>

<exception name="Invalid file type"
     link="Error.html">
User selects a non-CSV file.<br>System displays an error message.
</exception>
```

This will produce the following result:

Actions	
Upload a CSV	User selects a valid CSV and System starts uploading.
Invalid file type	User selects a non-CSV file. System displays an error message.

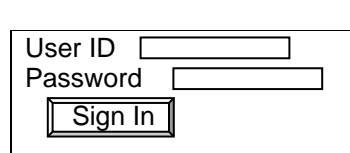**Attributes**: See attributes in <action>.

5.4 <screen>

The <screen> tag is used to describe a conceptual screen in HTML format. It is important to describe HTML elements only which will produce user or system actions on the page.

Example 1: A conceptual screen of Log In page.

```
<screen>
  User ID <input><br>
  Password <input><br>
  <button>Sign In</button>
</screen>
```

This tag will produce the following result:

```
User ID  [_____]
Password     [_____]
   [ Sign In ]
```

Attributes:

Attribute Name	Value	Description	Required
style	HTML style value	Specifies HTML element style	No

5.5

The <parameter> tag is used to define a default value of a URL parameter. A state HTML expects two parameters to be passed from the previous page. One is storydoc_id for User Story ID and the other is storydoc_name for the page title.
As long as the state HTMl is called from other page, the default values are not needed to define. But if you open the HTML directory with a browser and you want to see the exact same result on the page, you need to define the default values using the <parameter> tag.

Example 1: Define default values for URL parameters: storydoc_id and storydoc_name.

```
<parameter name="storydoc_id">US01</parameter>
<parameter name="storydoc_name">View History</parameter>
```

This tag won't produce any results on the state HTML.

Attributes:

Attribute Name	Value	Description	Required
name	A name of URL parameter	Specifies a URL parameter name	Yes

5.6 <note>

The <note> tag is used to describe a note or explanation on the state. The note is displayed on a state HTML page and is also attached under page tag section in storydoc HTML.

Example 1: Describe notes.

```
<note>The rows shall be sorted by created date.</note>
```

<note>The error rows shall be displayed in RED.</note>

This will produce the following result:

The rows shall be sorted by created date. The error rows shall be displayed in RED.

Attributes:

Attribute Name	Value	Description	Required
style	HTML style value	Specifies HTML element style	No

5.7 <test>

The <test> tag is used to describe a stet specific step, such as loading test data, assertion. The description is not displayed on state HTMLs and generated storydoc HTML. It is only added in storytest TEXT file.

Example 1: Load test data.

<test>Load test data.</test>

This won't produce any results in state HTMLs and storydoc HTML.
It will produce the following result in storytest TEXT:

Load test data.

The <test> tag can be described in <action> or <exception> tag. The test description is added after the <action> / <exception> step description by default.

Example 2: Load test data.

<action name="Upload a CSV"   link="UploadProgress.html"> User selects a valid CSV file and System starts uploading. <test>assert the uploaded data is returned when calling the lookup API</test> </action>

It will produce the following result in storytest TEXT:

M-1. Upload a CSV   assert the uploaded data is returned when calling the lookup API

Attributes:
The following attributes can be described in <test> tag added in <action> or <exception> tags.

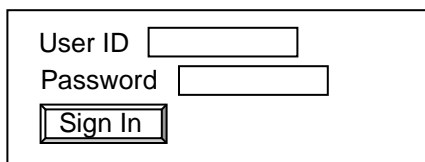| Attribute Name	Value	Description	Required

insert	'before' or 'after'	Specifies an insert-direction (before or after).
		  before: Inserts the test description before this action.
		  after: Inserts the test description after this action. (default)
			No

6   Apply CSS Styles

This chapter introduces how to specify stylesheets on <screen> and storydoc document.

6.1   style attribute

You can apply CSS styles on HTML elements in storyboard and storydoc HTMLs to make the look and feel more readable.

```
<screen>
  <label>User ID</label><input><br>
  <label>Password</label><input><br>
  <button>Sign In</button>
</screen>
```

The above <screen> tag is rendered as shown below:

It looks not bad but you may want to align the text fields. To do so, you can specify width of the labels using style attribute:

```
<screen>
  <label style="width: 85px; display: inline-block;">User ID</label><input><br>
  <label style="width: 85px; display: inline-block;">Password</label><input><br>
  <button>Sign In</button>
</screen>
```
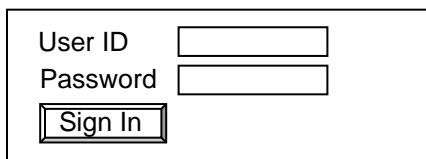
Now you will see the text fields aligned as shown below:

6.2 CSS file

Using style attribute is easy to apply on a specific element. However, as you can see the example in previous section, some styles can be reused on other elements. In that case, you may want to define the styles in an external CSS file.
To define styles in CSS file, you need to know how storydoc tags are converted into HTML tags.

```
<screen>
  <label>User ID</label><input><br>
  <label>Password</label><input><br>
  <button>Sign In</button>
</screen>
```

For example, the above <screen> tag is converted into below when you are storyboarding:

```
<div storydoc="screen" style="border:1px solid black; padding:10px; display:inline-block;">
  <label>User ID</label><input><br>
  <label>Password</label><input><br>
  <button>Sign In</button>
</div>
```

The <screen> is changed to <div> tag and storydoc and style attributes are added. Hence if you want to only apply styles on <label> tags in <screen>, you can define the style below:

custom.css
```
div[storydoc=screen] label
{
  width: 85px;
  display: inline-block;
}
```

In order to apply the styles on your HTML, you need to add a <link> tag with the CSS path.

```
<link rel="stylesheet" href="custom.css"></link>

<screen>
  <label>User ID</label><input><br>
  <label>Password</label><input><br>
  <button>Sign In</button>
</screen>
```

With the <link>, you will see the styles when storyboarding but in stordoc document. To apply the styles in storydoc document, the <link> must be added in the first HTML page as well.

index.html
```
<link rel="stylesheet" href="custom.css"></link>

<screen>
  <h1>Log-in Story</h1>
</screen>
<action name="Log In" link="US01/publicPage.html"/>
```

When you execute storydoc command, it reads all CSS files linked from the top HTML file and insert the styles into storydoc HTML file.

Below shows examples of styles:

Change background color of <screen>

```
div[storydoc=screen] {
  background-color: whitesmoke;
}
```

Change text color and size of <note>

```
span[storydoc=note] {
  color: seagreen;
  font-size: 10pt;
}
```

Change background color of Test Scenarios box in storydoc document

```
pre[storydoc=test] {
  background-color: lightyellow;
}
```

Change text color and font of page title (H1 tag generated by storydoc)

```
h1[storydoc] {
  color: royalblue;
  font-family: Tahoma, Geneva, sans-serif;
}
```

Change text color and font of non-storydoc H1 tag (H1 tag defined by user)

```
h1:not([storydoc]) {
  color: crimson;
  font: italic 200% fantasy;
  font-family: Impact, Charcoal, sans-serif;
}
```

7 Share Screens

In some cases, you will define similar screens in different HTMLs, and you may want to define once and reuse to others. This chapter shows a way to define common screens.

7.1 Storydoc.insertHTML

Storydoc provides you a JavaScript function below:

```
Storydoc.insertHTML( html , selectors , position );
```

Parameters:

Parameter Name	Value	Description	Required
html	A HTML text	A HTML fragment to be inserted	Yes
selectors	CSS Selector	CSS selector query string e.g. ".loginForm" matches with a class name "loginForm".	Yes
position	beforebegin afterbegin beforeend afterend	Position to be inserted. default: **afterbegin**	No

The function finds HTML elements by the given selectors query string, and insert the HTML string into all the HTML elements found. By default, it inserts the HTML after the beginning tag.

At first, you need to create an external JavaScript file and call the function in a function to be invoked when the HTML page is loaded.

custom.js
```
window.onload = function() {

  Storydoc.insertHTML(`
    <label>User ID</label><input><br>
    <label>Password</label><input><br>
    <button>Next</button>
  `, '.createForm');

};
```

As you did for CSS file, the JavaScript file needs to be included in the target HTML and the first HTML. Also you can define the class name on or in the <screen> tag.

```
<script src="custom.js"></script>
<screen class="createForm"/>
```

The above code generates the same screen image as the code below:

```
<screen>
    <label>User ID</label><input><br>
    <label>Password</label><input><br>
    <button>Next</button>
</screen>
```

Since the insertHTML inserts after the beginning tag of the target (in this case <screen>), if you have additional HTML fragment under the common screen, you can simply describe it in the <screen> tag.

```
<script src="custom.js"></script>
<screen class="createForm">
    <button>Cancel</button>
</screen>
```

The above code generates the same code as below:

```
<screen>
    <label>User ID</label><input><br>
    <label>Password</label><input><br>
    <button>Next</button>
    <button>Cancel</button>
</screen>
```

On the contrary, if you have an additional HTML fragment to be inserted before the common HTML, put another HTML tag and specify the class on it.

```
<script src="custom.js"></script>
<screen>
    <u>Go to home</u><br>
    <span class="createForm"></span>
</screen>
```

```
<screen>
    <u>Go to home</u><br>
    <span>
        <label>User ID</label><input><br>
        <label>Password</label><input><br>
        <button>Next</button>
    </span>
</screen>
```

7.2   Common HTML

There is another way to share a screen. This way to be introduced here is to share HTML file, so we can reuse <action> as well as <screen>.
First, you need to create a HTML to be shared. For example, below is a HTML to display an error message with a OK button.

Error.html
```
<script src="storydoc.js"></script>

<screen>
  <p>#{message}</p>
  <button>OK</button>
</screen>

<action name="An error '#{message}' is visible"
        link="#"
```

```
        onclick="javascript:window.history.back(-1);return false;"/>
```

In the <screen> and <action> tags, a placeholder, #{message}, is used to describe an error message. The actual message comes from the URL parameter. In previous HTML, the message parameter is specified as shown below:

```
<exception name="Invalid file type"
        link="Error.html?message=Please select a CSV file."/>
```

When the Error.html is loaded, the #{message} is replaced with the parameter value "Please select a CSV file.".

Error.html?message=Please select a CSV file.

Please select a CSV file.

  OK

Actions

An error 'Please select a CSV file.' is visible

8 Command Line

8.1 Options

Storydoc command accepts the following options:

C:> java -jar storydoc.jar [-doc] [-test] <HTML file path>

Options:

Option Name	Description	Required
-doc	Generates storydoc.html file.	No
-test	Generates storytest.txt file.	No

Both -doc and -test options are not specified, the storydoc tool generates the both files.

9 Tips

9.1 Story Relations (include, extend and generalization)

User Story can make relations with other stories. To make the relation, specify User Story ID in "include", "extend" or "generalization" attribute of <action> tag in Story HTML.
It is easy to confuse extend relationship between stories with generalization. We here show some explanations of usage for each.

Include relation
If the story uses other stories, specify the Story IDs (comma-separated) in **include** attribute.
For example, if you specify the following attribute value in US01 <action> tag, the following include relations will be drawn in generated Use Case diagram.

<action id="US01" **include**="US02, US03">…</action>
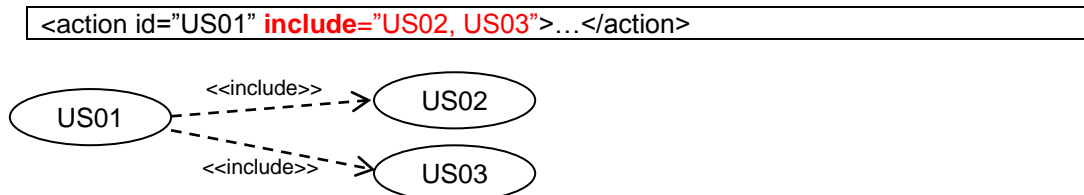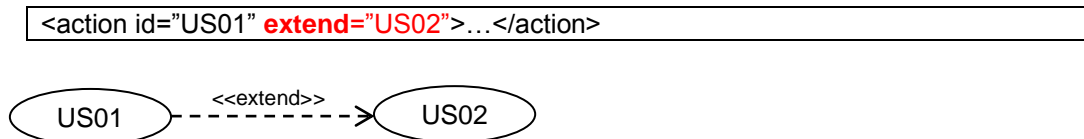
The above relation represents that US01 uses (includes) US02 and US03. In another words, US02 and US03 are part of US01.

Extend relation
If there are alternative User Stories, specify the Story IDs (comma-separated) in **extend** attribute.
For example, if you specify the following attribute value in US01 <action> tag, the following extend relations will be drawn in generated Use Case diagram.

<action id="US01" **extend**="US02">…</action>

The above relation represents that US02 is an alternative use case of the US01. That means the US02 can be executed when the US01 is not available to work.

Generalization relation
If the user story is specialized based on a generic (parent) user story, specify the Story IDs (comma-separated) in **generalization** attribute.
For example, if you specify the following attribute value in US01 <action> tag, the following generalization relations will be drawn in generated Use Case diagram.
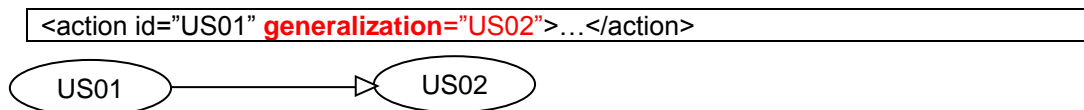
<action id="US01" **generalization**="US02">…</action>

The above relation represents that US01 is a specialized user story inherited from the US02. US02 can be more generic story such as "Register User" and US01 will be a concrete story such as "Register Admin", "Register Operator", "Register Visitor".

9.2   Conceptual Screen

Each HTML could have a <page> tag to represents a Conceptual screen. Conceptual screen does not show a concrete page layout. It only shows HTML elements which will be related to user or system actions, taking place on the page. Hence conceptual screens look quite simple rather than realistic.
For example, a search page can be represented by the following simple HTML code.

```
<screen>
   Keyword <input>
   <button>Search</button>
</screen>
```

The HTML will produce the HTML page below:

Search Form Page
```
Keyword [          ]
 [ Search ]
```

As long as we are discussing what the user wants to do on this page, we just need the two elements (A textbox for a keyword and a Search button to start starting).
In a real website, we could have more elements such as a link to show an advanced search form, a selection box to choose a search category. But those are for another story and we can ignore for now.
Having said that, in some cases, you may want to describe unrelated HTML elements. In the search example, let's say the search form is supposed to be displayed on the search results page also.

Search Results Page
```
Keyword [          ]
 [ Search ]

Results:
Title1 description…
Title2 description…
Title3 description…
…
```

On this page, we want to discuss search results only. Although the search form is displayed on it, we don't have to know the details. It is enough to know that the search form is shown on the same page. To represent this, the page can be revised as shown below:

```
<screen>
   <div style="border:1px dashed;">Search Form</div><br>
   Results:<br>
   Title1 description....<br>
   Title2 description....<br>
   Title3 description....<br>
   ...
```

```
</screen>
```

(Revised) Search Results Page

```
┌──────────────────────────────┐
│   ┌ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┐  │
│   │    Search Form        │  │
│   └ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┘  │
│                              │
│   **Results:**               │
│   Title1 description…        │
│   Title2 description…        │
│   Title3 description…        │
│   …                          │
│                              │
└──────────────────────────────┘
```

Describing unrelated HTML elements will cause duplicated definitions, which makes you duplicate work when the screen layout has been refined. It is important to define related elements only on each page.

9.3 Acceptance Criteria is NOT Test Scenario

This chapter explains the differences between acceptance criteria and test scenarios. We have roughly explained about it in previous chapter, but this is really important for you to understand why you need to spend more time and create storyboard to realize user stories and generate acceptance test scenarios. If customer describes acceptance criteria in scenario format (GWT/BDD), it looks like test scenario already, and we can make test scenarios by adding a few more lines.

```
Given I open the website
When I enter a model name
Then the system shows products
```

However, at this moment, the criteria was created based on core requirements only. The detailed information such user interface, error cases, are not considered. It is because customer is responsible to clarify what they need. In this example, they need to explore and make sure that they will receive values on their business by enabling website users to search products by a model name. They do not care how user and system interact at this moment. Below is the Acceptance Test Scenarios generated for Product Search example.

```
Main Scenario:
      Given I open the website
  M-1. When I enter a part of model name
      Then the system shows a list of products
  M-2. When I select a product
      Then the system shows the product detail

Alternative Scenario 1:
      Given I open the website
  A1-1. When I enter an exact model name
      Then the system shows the product detail

Exception Scenario 1:
      Given I open the website
  E1-1. When I search with an empty keyword
      Then the system shows an error message
  E1-2. When I go back to the search form
 #E1-3. Go back to M-1
```

Compared to the acceptance criteria, there are some additional information in test scenarios.

- A new screen "product detail"
- Alternative Scenario
- Exception Scenario

By analyzing the given user story and acceptance criteria, we found a new screen (Product detail) and additional two scenarios (Alternative and Exception scenarios). There are many ways to analyze but we here used storyboarding technique as shown in previous chapter. It is not an easy step. We need to walk though the story with a browser and find alternative courses and error courses. In the process of the analysis, we can realize what screens we need to add and what actions could happen on each page.

Although acceptance criteria looks like acceptance test scenario, approaches to define those are quite different. When customer defines acceptance criteria, they consider various options and pick the ones they really need for their business. On the other hand, when system analyst or developer defines test scenarios, they consider how they can be converted into programming code which requires detail instructions. So, they need to find all possible courses of the story and clearly define each step.
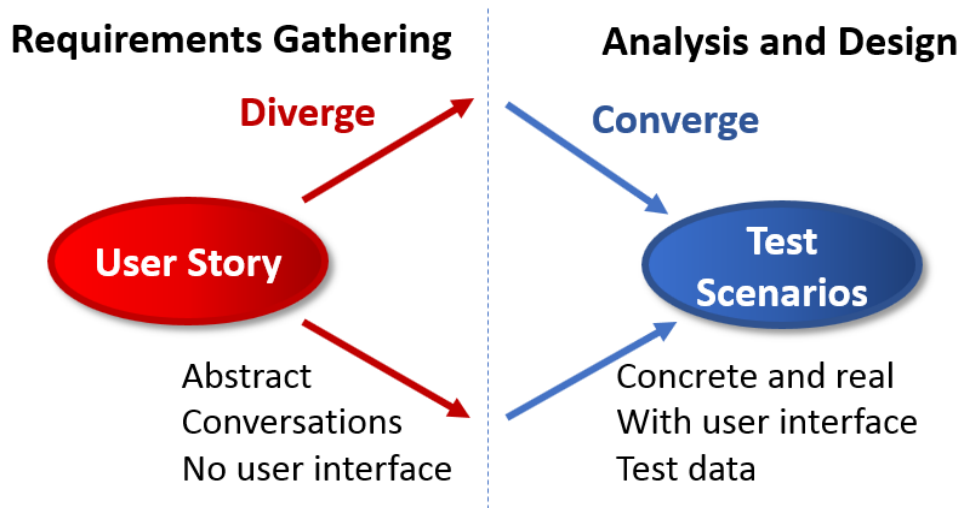
For that reason, Acceptance Criteria does not mention HTML elements so much, while HTML elements appear a lot in test scenarios. For example, acceptance criteria says "Search for keyword", while test scenario could say "Enter a keyword and click Search button".