TRIFORK.

QTI Engine: Custom Script Interaction

History

Version	Date	Date	Author
1.0	2013-01-30	Initial stable version	Eric Bednarz

Abstract

Custom Script interactions are enabled by using the `object` element type with a `type` attribute value that corresponds to a registered, non-obsolete scripting media type as defined by **RFC 4329** (cf. **attributes**), and a `data` attribute that specifies a manifest file in JSON format.

The interaction is rendered to an `iframe` element whose `contentDocument`'s

Contents

- `compatMode` is `CSS1Compat` (aka 'standards-compliant')
- `head` element contains the following elements, in order:
  - a copy of the `base` element of the iframe's `ownerDocument`
  - an empty `title` element
  - copies of all linked style sheets of the iframe's `ownerDocument`
  - one `link` element for each style sheet specified in the **JSON manifest** file
  - a `script` element that assigns the public custom javascript interaction **API** to a **global identifier** in the iframe's `contentWindow`
  - one `script` element for each script file specified in the **JSON manifest** file
- `body` element is empty

All other `HTML` is generated by the delivered script, considering the `load` and `DOMContentLoaded` events as you would expect in any normal document.

XML

Element type

- `object`

Attributes

- `type` - (`application/ecmascript` | `application/javascript`)
- `data` - the relative URI reference of the **JSON manifest** resource
- `width` - width of the generated `iframe` element
- `height` - height of the generated `iframe` element

All attributes are required.

Child elements (optional)

- `param`

If the required number of response fields is greater than 1, it can be specified with an optional `param` element with the follwing attributes:

- `name` = "`responseLength`"
- `value` - *Integer*

JSON manifest

To prevent conflicts with JSON files in the package that are part of the interaction implementation, the JSON manifest file name **must** be equal to or end with `manifest.json` because the URLs in the manifest file are rewritten by the importer.

The JSON manifest consists of an object with one to three properties:

- *Array* `script`
- [*Array* `style`]
- [*Array* `media`]

The value of each property is an Array of relative URI references. The `script` and `style` Arrays must be ordered according to their respective cascade and dependency requirements.

NB: URI references in the JSON manifest are identical to those in the package manifest (i.e. not relative to the location of the JSON manifest file).

Script API

Global identifier

Object `CES`

Properties

- *Function* `setResponse`(*String* `data`)
  Sets the candidate response 'as is' (the format of the response and its state is the responsibility of the implementation).
- *Function* `getResponse`()
  Gets the previously saved candidate response (if any).
- *Function* `getMedia`()
  Gets an array of resolved manifest media URLs.
- *Function* `setStageHeight`([*Number* `verticalMargin`])
  Convenience function to adjust the height of the generated iframe to the current height of the custom interaction's content height; if that content's visual size exceeds its layout box (e.g. by using `box-shadow`), an optional vertical margin can be specified.

Example

XML

```
<customInteraction responseIdentifier="RESPONSE">
    <object
        type="application/javascript"
        data="resources/manifest.json"
        width="600" height="600"
    />
```

```
            <param name="responseLength" value="2">
        </object
    </customInteraction>
```

JSON Manifest

```
{
    "script": [
        "resources/library.js",
        "resources/widget.js",
    ],
    "style": [
        "resources/widget.css"
    ],
    "media": [
        "resources/widget.png"
    ]
}
```

Generated HTML

```
<!DOCTYPE html>
<html>
<head>
<base href="/base/path/of/parent/document/">
<title></title>
<link href="parent/document/stylesheet" rel="stylesheet">
<link href="resolved/path/to/resources/widget.css" rel="stylesheet">
<script>
var CES = window.parent.customInteraction["RESPONSE"];
</script>
<script src="resolved/path/to/resources/library.js">
<script src="resolved/path/to/resources/widget.js">
</head>
<body></body>
</html>
```

Script

```
// use a resolved media object
var media = CES.getMedia();
var image = document.createElement('IMG');
image.src = media[0];
// image.getAttribute('src')
// => resolved/path/to/resources/widget.png
// ...
// save response and state...
function serialize() {
    // return serialized response and state as string, e.g. JSON, XML
}
CES.setResponse(serialize());

// restore response and state...
function unserialize(string) {
    // return unserialized response and state from string
}
var response = unserialize(CES.getResponse());
```

Limitations and risks

Important: if you need to save media references as part of the state, save their `media` array indexes, not the resolved values. The resolved value can vary, depending on the application context your custom interaction is loaded in.

Since the custom interaction is rendered in an `iframe`, several of its `ownerDocument`'s tools cannot interact with it, notably:

- magnifier
- text marker
- spell checker
- text to speech

While the `iframe` provides a separate environment that should prevent most accidental scripting conflicts (e.g. global variables, augmented native or host objects), it should not be considered a sandbox in the security sense.

2012–2013 | **ces@trifork.nl**