Documenting SuiteScripts

1. Introduction

Writing documentation is hard. Writing good documentation is even harder. And for most developers there's a constant pressure to concentrate on all other more important things besides the documentation. There's never enough time for writing docs.

The goal of documenting our SuiteScripts is to make the process of writing documentation as painless as possible, in tandum of documenting our work for further and continued development.

The required format for structuring our documentation is **Markdown**:

```
/**
 * Returns description of the time of the day.
 *
 * Different heuristics are used to come up with the **most** appropriate
 * wording for the current user.
 * @author Jeff Pipas <jpipas@entertainment.com>
 * @return {String} Possible return values are:
 *
 * - midday
 * - late night
 * - early morning
 * - just before lunch
 * - tea time
 */
function getDayTime() {
```

[Markdown]: http://daringfireball.net/projects/markdown/syntax

2. Classes

Each file should start out with a doc-comment. In this doc-comment you define the file, what its doing, who created it, and any other important information that is realted to the file.

```
/**
* @class EPI_SUE.Opportunity
* @mixin EPI_SUE.Opportunity.Recommit
* @mixin EPI_SUE.Opportunity.RequestedArrivalDate
* This file contains the user event functions for Opportunities.
* It has several library files attached (noted above)
```

```
*
* @author Jonathan Boivin <jonathan.boivin@erpguru.com>
* @author Jeff Pipas <jpipas@entertainment.com>
* @mod ###November 12, 2011 - Jeff Pipas
* I needed to change the flow of the beforeLoad function so that blah blah blah.
* This meant refactoring the
* {@link EPI_SUE.Opportunity.Recommit EPI_SUE_Opp_Recommit.js}
* file to account for this change.
* @mod ###September 3, 2011 - Jonathan Boivin
* Initial use and creation of this script.
*/
```

Note that there are several `@tags` in this example. The `@class` defines the structure at which scripts are organized. The idea here is to put each of the files into a standardized namespace. A class namespace is made up of the script type (User Event, Client, Suitelet, etc), followed by the record the script is applied too (Customer, Opportunity, SalesOrder), then the actual class is added as the main purpose or functionality the script is accomplishing.

Deconstructed a `@class` declaration is a concatination of the following:

- Begin with one of the following namespaces: `EPI_SUE.`, `EPI_CUE.`, `EPI_SSU.`, `EPI_LIB.`, `EPI_SSC.`, `EPI_RST.`
- Add the record the script applies too: `Opportunity.`, `Customer.`, `SalesOrder` - these should be CamelCased record types
- Followed by the functionalty the script is providing - for example: `Recommit`, `CloseProcess`

Further classification can be used to differentiate between the different "sales channels". For example a Fundraising Close Process Script could be declared as follows:

`@class EPI_SUE.SalesOrder.CloseProcess.Fundraising`

while its counterpart Direct Sales, could be declared as:

`@class EPI_SUE.SalesOrder.CloseProcess.DirectSales`

Also note you can also use `@link` or `@mixin` so the related files or "Library" files as defined in the NetSuite deployment, are also easilly accessible and are referenced with one another.

3. Properties (Global Vars)

Properties are a lot like method parameters. They can have default values and sub-properties. You should define all Global Var's in your scripts as properties of the file.

```
/**
 * @property {String} size Size of the item.
 */
var FIELD_CUSTOM_FORM_FIELD1 = 'custbody_my_form_field1';
```

4. Type Definitions

Throughout this guide you've seen type definitions like {`Number`}. These aren't just arbitrary strings enclosed in curly braces - there's a specific syntax for specifying types and you must follow it. Here's a short overview of supported syntax:

```
- ‘{String[]}‘ - array of strings.
- ‘{String[][]}‘ - 2D array of strings.
- ‘{Number/String/Boolean}‘ - either number, string, or boolean.
- ‘{Boolean...}‘ - variable number of boolean arguments.
```

We also check that you don't reference any unknown types. For example if you specify {`Foo`} and you don't have class Foo included in your scripts, a broken link will be created. Warnings aren't thrown for JavaScript builtin types (`Object`, `String`, `Number`, `Boolean`, `RegExp`, `Function`, `Array`, `Arguments`, `Date`, `Error`) and few DOM types (`HTMLElement`, `XMLElement`, `NodeList`, `TextNode`, `CSSStyleSheet`, `CSSStyleRule`, `Event`). To document a variable that can be of any type whatsoever, use the `Mixed` type. But try to keep its use to the minimum, always prefer the {`Foo/Bar/Baz`} syntax to list possible types.

Valid types include `Number`, `String`, `Boolean`, `RegExp` and `Function`. Everything else will be labeled as `Object`.** NetSuite's `nlObject*` type should be treated as type `Object`. Also note that we infer a Type when no implicit definition is given.

5. Methods

When a doc-comment is followed by a function it's auto-detected as method.

I won't bother explaining the normal `@param` and `@return` syntax and will instead diverge into a more complex example:

```
/**
 * Calculates the score for each item in the array.
 *
 * @param {Array} array The input array.
 *
 * @param {Function} fn The callback function.
 * For every item in array the function will be called with:
 * @param {Mixed} fn.item The item itself.
 * @param {Number} fn.index Index of the item.
 * @param {Number} fn.return Should return a numeric score between 0 and 1.
 *
 * @param {Object} scope Value for ‘this‘.
 *
 * @return {Object[]} Array of objects with fields:
 * @return {Mixed} return.item The original item
 * @return {Number} return.score The score
 */
 function score(array, fn, scope){
```

Here you see a syntax for describing parameters and return values of Function or Object type. For nlObjectForm, nlObjectRecord, nlObject* objects you should use the standard Object type. For callback functions you can describe the parameters and return value. And of course you can do this all recursively however complex parameters or return values you need.

6. Events

The `@event` tag allows us to further define actual mapped functions to the events triggered by NetSuite.

```
/**
* This is the recommit process for Fundraising
*
* @event
* @inheritdoc EPI_CUE.Opportunity.pageInit
*/
function pageInit_Recommit(Object type){
```

The example above introduces `@inheritdoc`. By specifying a class and method/event, the documentation for this event will also be augmented/filled in with its inherited method/event documentation. This helps to fully understand the relationship between all of these methods/events

7. Cross-references

Along the same lines as `@inheritdoc`, inside comments you can link to classes and class members using the {`@link`} tag.

```
{@link Class#method {@link Class#method link text}}
```

You can use this `@tag` to effortlessly link related function calls to one another, for easy navigation through the document. {`@link`} can be used anywhere in the comment-doc to create a link to any other defined Class#method/event.

8. Glossary - Tags and Syntax

@class

This `@tag` namespaces a script. It defines the categorization and heirarchy for the script.

@author

The author of the script. Put your first and last name, followed by your email enclosed in <> brackacks. Example:

`@author Jeff Pipas <jpipas@entertainment.com>`

Multiple `@author` tags can be placed in the document header. `@author` is also avaliable at a function/method level and should follow the same syntax.

@mixin

@singleton

@param

@property

A `@property` is used to define a global variable that is used in a script. It is normal used for constant variables that do not change.

@event

This `@tag` should be used when referencing User Events or Client Events within NetSuite. An event is any method definition that is directly linked to a forms deployment, examples include: `beforeLoad`, `afterSubmit`, `fieldChange`, `validateLineItem`

@jira

If the method, event, or script is part of a JIRA task - simply add the issue number after this tag

```
@jira NS-381
```

The parser will autmoatically pick up on this and create a link directly in the documentation to the issue within JIRA

@mod

@return

@link

@inheritdoc