Smart NFC Card Application

Coding Standards Document

Vast Expanse (Group 7)

COS 301 - 2019

Wian du Plooy	*u17237263*
Duncan Vodden	*u17037400*
Tjaart Booyens	*u17021775*
Savvas Panagiotou	*u17215286*
Jared O'Reilly	*u17051429*

Table of Contents

Introduction

Coding standards are a set of rules that serve as requirements and guidelines for writing programs for a project or within an organisation. This document will outline the various Coding standards and Coding Conventions which will be used throughout the implementation and development of the Smart-NFC-Application project.

Throughout the project, our documentation will occur between comments which follow the following standards:

```
/**

*

*/
```

Where information will be included in the lines with a single asterisk (*).

File Headers

A file header will be located at the top of each file that is created which specifies various information that includes:

ITEM	DESCRIPTION
File Name	The name of the file that is being presented
Project Name	Name of the project for which the file was written for
Organisation Name	Name of the organisation which creates the program
Copyright	Copyright information
List of Classes	A List of the classes declared and implemented in the file
Related Documents	A list of the related documents (URLs included if possible)
Update History	A list of updates, specifying the date, author and the change made
Functional Description	Overall description of the functionality and behaviour of the program
Error Messages	A list of error messages that can be produced by the program specified in the file
Assumptions	A list of conditions that must be satisfied or may affect the operation of the program
Constraints	A list of restrictions on the use of the program including restrictions on the input, environment and various other variables

Description of Classes

A description of each class will be located before the declaration of the class and will include information such as:

ITEM	DESCRIPTION
Purpose of class	A statement of the purpose of the class
Usage Instructions	How the class will be used
Author	Specified by **“@author”**. The programmer who created the class
Version	Specified by **“@version”**. The version number of the class

Before a function is declared the following descriptions will be included for each function

ITEM	DESCRIPTION
Description	Description of what the function will be used for/the function's purpose. The description will not have a label “Description”, instead, the first line will be the description
Param	Specified by **“@param paramName paramDataType paramDescription”** where paramName is the name of the parameter, paramDataType is the data type of the parameter and paramDescription is a description of what the parameter is in relation to the function
Return	Specified by **“@return returnDataType returnDescription”** where returnDataType is the data type of the returned object and returnDescription is a description of the object returned

Naming Conventions

The following Naming conventions should help the understanding of the program as well as aid in the maintenance of the program:

ITEM	CODING CONVENTION
Folders	Folders will follow the rules of camel casing and will start with uppercase characters
Files	Files will follow the rules of camel casing and start with lowercase characters
Classes	Classes will follow the rules of camel casing and will start with uppercase characters
Attributes	Attributes will follow the rules of camel casing and start with lowercase characters
Functions	Functions will follow the rules of camel casing and start with lowercase characters
Constants	Constants will be declared in all uppercase characters
Variables	Variables will follow the rules of camel casing and start with lowercase characters
Subscripts	Variables that require subscripts will be fined in a manner such as: "variableName_i" where "_i" is the subscript.

Formatting Conventions

Formatting conventions specify formatting rules used to arrange program statements.

The following formatting conventions should help in the structure of our program and help us write 'neat' code

ITEM	CODING CONVENTION
Line Break	Have a line break after every function to show separation clearly and in every function have a line break between different code blocks such as before and after loops The first line of the function/loop/condition starts on the line after the function definition/loop/condition
Indentation	Use tabs for indentation and indent separate coding blocks within each other with one more tab
Alignment	After a coding block has started the '{' will be on the same line and the '}' will be aligned with the first character that started the coding block
Spacing	Add a space before and after every opening and closing parenthesis Spaces will be added after every data type and the variable name For assignments and comparisons, a space is added before and after every operator Spaces will be added for comments after the '//'

In-code comment Conventions

In-Code Comments will aid the understanding as well as the maintenance of the program.

These comments will be kept to a minimum to avoid having code that looks cluttered with comments. These comments should occur before a block of code to explain what that specific block of code achieves

Example

The following example will be of a JavaScript Class contained in a file named helloWorld.js

```
/**
*	File Name:	helloWorld.js
*	Project:	Smart-NFC-Application
*	Orginization:	VastExpanse
*	Copyright:	©  Copyright 2019 University of Pretoria
*	Classes:	HelloWorld
*	Related documents:	SRS Document - www.example.com
*
*	Update History:
*	Date		Author		Version		Changes
*	-------------------------------------------------------------------------------------
*	2019/05/18	Duncan		1.0		Original
*	2019/05/19	Tjaart		1.1		Added foo Function
*
*	Functional Description:		This class is to demonstrate the use of our Coding
*					standards that we will be using throughout our COS
*					301 Module
*	Error Messages:		“Error”
*	Assumptions:	None
*	Constraints:	None
*/

/**
*	Purpose:	This class is used demonstration purposes of coding conventions
*	Usage:		This class can be used to output “Hello World” to console by calling
*			function foo
*	@author:	Duncan Vodden
*	@version:	1.1
*/
class HelloWorld{

	/**
	*	The constructor of the class is used to initialise the hello attribute of the class
	*/
	constructor(){
		this.hello = “Hello World”;
	}

	/**
	*	This Function calls the bar function and returns an appropriate response
	*	depending on the result of the bar call
	*	@return string Return “Error” if bar was unsuccessful else return “Success”
	*/
	foo(){
		// This will return the appropriate value depending on the result of bar
		let resultBar = bar(this.hello);
		if (!resultBar){
			return “Error”;
		}
		else {
			return “Success”;
		}
	}
	/**
	*	This function prints out a message to the console and returns true after it has
	*	done so
	*	@param hello string This is a string passed into the function
	*	@return bool Return true after console logged output
	*/
	bar(hello, bye){
		// this
		console.log(hello);
		return true;
	}
}
```