**Software Requirements Document for MTT Configuration File Editor**

**Part 2: Functional Requirements**

**Prepared for Master of Software Engineering Capstone Project**

**Advisor: Joshua Hursey**

**Student : Han Chen**

**Date: May 1, 2015**

**1. About this document**

This document describes the functional requirements for the software for MTT Client Configuration File Editor. The product overview and assumptions for the product are given in [1].

**2. Functional Requirements**

Each functional requirement is given in the following format:

*Index:*

*Name:*

*Purpose:*

*Input parameters:*

*Action:*

*Output parameters:*

*Exceptions:*

*Remarks:*

*Cross-references:*

*Index* refers to a unique index assigned to this functional requirement. It will be used for cross-referencing this requirement from other requirements and also from other documents such as the design document.

*Name* is a descriptive name given to this functional requirement. This name does not need to be unique.

*Purpose* is a short description (in a line or two) of the function. It is used to quickly understand the functionality and is also used to search the required function when all the functionalities are browsed through.

*Input parameters* refer to a set of parameters that the given function accepts as input. These parameters are required in order to design and implement the current function. No type information will be included for parameters in this document.

*Output parameters* refer to a set of parameters that are output/exhibit by the current function, when implemented. No type information will be included for parameters in this document.

*Action* refers to a set of tasks or activities that must be performed in order to satisfy this requirement. These tasks/activities are listed in no particular order. The design will take care of sequencing the tasks.

*Exceptions* refer to a set of conditions, each of which indicates a situation in which the function will stop. Notice that this column only lists a set of exceptions that might occur but does not suggest any action that must be taken when the exceptions occur. These actions will be included in the design document.

*Remarks* includes a set of comments that explain more about the functionality. It also describes hints to the designer and implementer that are suggested by the requirements analyst.

*Cross-references* refer to a set of other functional requirements that are related to the current function.

**2.1 Functional Requirements for Parser**

*Index*: P.1

*Name*: OpenFile

*Purpose*: To open a file

*Input* parameters: File path.

*Action*: Trying to open a file.

*Output parameters*: Opening result.

*Exceptions*: Not a valid INI file or File doesn’t exist.

*Remarks*: This function may be used whenever user wants to open a file by inputting file path as parameter. The designer may choose to have another function called “FileTypeValidation” which may be linked to this function. The designer may choose to have another function called “SyntaxCheck” which may be linked to this function.

*Cross-references*: P.2 P.3

Index: P.2

*Name*: FileTypeValidation

*Purpose*: To check whether the file is valid type or not

*Input* parameters: File path.

*Action*: Check file extension.

*Output parameters*: True or False.

*Exceptions*: None.

*Remarks*: Functionality “OpenFile” may use this function whenever user wants to open a file.

*Cross-references*: None.

*Index*: P.3

*Name*: SyntaxCheck

*Purpose*: To check the syntax of file.

*Input* parameters: File stream.

*Action*: Trying to check file syntax by series of automatic operations. Including “FileStreamSplit”, “CommentSyntaxCheck”, “SectionSyntaxCheck”, “KeyValuePairSyntaxCheck”, “FuncletsSyntaxCheck”, “CrossReferenceCheck”, and so on.

*Output parameters*: Checking result.

*Exceptions*: None.

*Remarks*: Functionality “OpenFile” will use this function whenever user wants to open a file.

*Cross-references*: P.4. P.5. P.6 P.7. P.8. P.9. P.10

*Index*: P.4

*Name*: FileStreamSplit

*Purpose*: To Split file stream to different phases by symbols

*Input* parameters: File stream

*Action*: System will split input file stream by different symbols based on different scenes. This process will prepare the split contexts to be used in further checking.

*Output parameters*: Error message including location and reason.

*Exceptions*: None.

*Remarks*: Functionality “SyntaxCheck” may use this function whenever user wants to open a file. Basically this process will split file stream to a couple of categories such as comments and sections. Sometimes they also include isolated key-value pairs which is outside of sections.

*Cross-references*: None.

*Index*: P.5

*Name*: CommentSyntaxCheck

*Purpose*: To report the comment syntax error of file

*Input* parameters: Error location, error reason.

*Action*: System will figure out the correct syntax of comment and then ignore them in further checking.

*Output parameters*: Error message including location and reason.

*Exceptions*: None.

*Remarks*: Functionality “SyntaxCheck” may use this function whenever user wants to open a file. Usually INI file may content more than 50% of comments in file stream, which will bring a great amount of extra-cost to parsing process. Therefore ignore comments first will be necessarily useful at beginning of parsing.

*Cross-references*: None.

*Index*: P.6

*Name*: SectionSyntaxCheck

*Purpose*: To report the syntax error of file

*Input* parameters: Error location, error reason.

*Action*: Check section syntax based on default section names and so on.

*Output parameters*: Error message including location and reason.

*Exceptions*: None.

*Remarks*: Functionality “SyntaxCheck” may use this function whenever user wants to open a file. There are some of default sections being essential phases that are required to occur in configuration file. Therefore we have to ensure they appear correctly.

*Cross-references*: None.

*Index*: P.7

*Name*: KeyValuePairSyntaxCheck

*Purpose*: To report the syntax error of file

*Input* parameters: Error location, error reason.

*Action*: Checking key-value pairs based on required format and syntax.

*Output parameters*: Error message including location and reason.

*Exceptions*: None.

*Remarks*: Functionality “SyntaxCheck” may use this function whenever user wants to open a file. The most contents of each sections are key-value pairs. Therefore ensuring their syntax correctness is important.

*Cross-references*: None.

*Index*: P.8

*Name*: FuncletsSyntaxCheck

*Purpose*: To report the syntax error of file

*Input* parameters: Error location, error reason.

*Action*: Checking funlects names and syntax. Collecting location and reason of errors when they occur.

*Output parameters*: Error message including location and reason.

*Exceptions*: None.

*Remarks*: Functionality “SyntaxCheck” may use this function whenever user wants to open a file. Funclets are one of features of MTT configuration file. Their names are reserved key words and we need to check their syntax as well.

*Cross-references*: None.

*Index*: P.9

*Name*: CrossReferenceCheck

*Purpose*: To report the syntax error of file

*Input* parameters: Error location, error reason.

*Action*: Checking errors of cross reference among different sections.

*Output parameters*: Error message including location and reason.

*Exceptions*: None.

*Remarks*: Functionality “SyntaxCheck” may use this function whenever user wants to open a file. Ensuring that sections have to call existing and correct other sections.

*Cross-references*: None.

*Index*: P.10

*Name*: ErrorReport

*Purpose*: To report the syntax error of file

*Input* parameters: Error location, error reason.

*Action*: Listening to the result of “SyntaxCheck”, collect location and reason of errors when they occur.

*Output parameters*: Error message including location and reason.

*Exceptions*: None.

*Remarks*: Functionality “SyntaxCheck” may use this function whenever user wants to open a file.

*Cross-references*: None.

**2.2 Functional Requirements for Estimator**

*Index*: E.1

*Name*: EstimateTotalRuntime

*Purpose*: To report the total estimated runtime of file

*Input* parameters: None.

*Action*: Collect necessary values in each section, use those values to query estimated runtime records from database, sum the result, and report it to user.

*Output parameters*: Total estimated runtime.

*Exceptions*: None.

*Remarks*: Only can be triggered when there are no syntax errors.

*Cross-references*: E.2. E.3.

*Index*: E.2

*Name*: EstimateRunTimeBySection

*Purpose*: To report the estimated runtime by section name

*Input* parameters: Section name.

*Action*: Collect necessary values in corresponding section, use those values to query estimated hours records from database, sum the result, and report it to user.

*Output parameters*: Estimated runtime of that section.

*Exceptions*: None.

*Remarks*: Only can be triggered when there are no syntax errors.

*Cross-references*: None.

*Index*: E.3

*Name*: EstimateRunTimeByTestSuite

*Purpose*: To report the estimated runtime by section name

*Input* parameters: Test suite name.

*Action*: Tracking linear test suite including Test Get, Test Build, and Test Run, and then estimating its runtime.

*Output parameters*: Estimated runtime.

*Exceptions*: None.

*Remarks*: Only can be triggered when there are no syntax errors.

*Cross-references*: None.

**2.3 Functional Requirements for Tuner**

*Index*: T.1

*Name*: Navigation

*Purpose*: To give user choices of selecting proper operations

*Input* parameters: Command names.

*Action*: Doing corresponding operations based on what user what to tune.

*Output parameters*: Estimated runtime after changes.

*Exceptions*: Invalid input.

*Remarks*: Only can be triggered when there are no syntax errors. This functionality needs input validation. This functionalities is going through the tuning process. It will enrich users’ operation choices therefore to give this system more friendly interfaces.

*Cross-references*: E.1. E.2. E.3. T.2. T.3.

*Index*: T.2

*Name*: TuneParameter

*Purpose*: To tune parameter therefore changing the estimated runtime

*Input* parameters: Section name, parameter, and updated value.

*Action*: Locate the value by input section name and parameter, update the value, and show new estimated result to user.

*Output parameters*: Estimated runtime after changes.

*Exceptions*: Invalid input.

*Remarks*: Only can be triggered when there are no syntax errors. This functionality needs input validation. It will show new estimated result after user confirmed changes.

*Cross-references*: P.3. T.3. E.2. E.3.

*Index*: T.3

*Name*: ReportComparingResult

*Purpose*: To give user a comparing result of estimated runtime between before tuning and after tuning.

*Input* parameters: None.

*Action*: Displaying comparing result based on old file and user input. After report, the “Navigation” will be available for user to do further operations.

*Output parameters*: Estimated runtime after changes.

*Exceptions*: Invalid input.

*Remarks*: Only can be triggered when there are no syntax errors. This functionality needs input validation. It will show new estimated result after user confirmed changes. This functionalities will give user opportunity to regret their tuning by choosing more options such as “undo” and “redo”.

*Cross-references*: None.

**References**

1. Han Chen, “Software Requirements Document for MTT Client Configuration File Editor – Part 1: Product Overview and Assumptions”, May 2015.