Test Sequence Document Spec

Version	Date	Author	Comment
0.8	11/15/2015	Wei Wang	Intial Draft
0.9	11/22/2015	Wei Wang	Added explanation for limit and special return tags
1.0	12/02/2015	Wei Wang	Adding naming convention for sequence csv file

Structure of of the Test Sequence

Name of the sequence file

The name of the sequence file must be of the format [name]__[version].csv. Name is the sequence name. The version is the sequence version. The name and version will show up in GUI and be reported to PDCA. The separator is double underscore. It's OK to have single underscore in the sequence name but not double underscore.
It's OK if there is a dot in the version. For instance: short_plan__1.0.csv.

Content of the sequence file

Test sequence is a sequential list of test items. Each item is executed in order. Each test item consists of these fields:

Field Name	Mandatory	Constraints
GROUP	Yes	All items of a group must appear consecutively Between two items of the same group there can not an item that does not belong in this group.
TID	Yes	Must be a unique ID in the whole test plan. This will be sent to PDCA as the test name, so it must follow the constraints of the PDCA test name. The most important limit is that it can not be longer than 48 characters.
DESCRIPTION	No	
FUCNTION	Yes	This must be a function supported by the test engine. See below for the list of supported functions
TIMEOUT	No	If no timeout value is specified, the default timeout is 3000mS. If you do specify a value, the unit is mili Seconds.
PARAM1	No	The parameter to pass to the test functions. Most functions do require a parameter. See below for further clarifications.

Field Name	Mandatory	Constraints
PARAM2	No	The parameter to pass to the test function or a returned value from the test function. See below for further clarifications.
UNIT	No	The unit for the value reporting to PDCA, and for information purpose. Otherwise it’s ignored by the software
LOW	No	
HIGH	No	
KEY	No	Along with the VAL field, they define if the current step should be executed. KEY is always a variable whose value is stored in memory by previous steps. If the value of the variable named by KEY matches the value of VAL, this step should be executed. If not, this step should be skipped.
VAL	No	
VALIDATE	No	This is a single line of python code which should return and only return a pass/fail value. This checks if the returned values form the test engine is valid or not.

Details of selected fields:

PARAM1 and PARAM2:

There are special variable names in PARAM1 and PARAM2, either surrounded by curly braces “{{}}” or square brackets “[[]]”. They have different meanings depends on where they are. The table below explains the differences:

Marker \ Location	PARAM1	PARAM2
[[]]	This variable will be replaced with its actual value before sent to the TestEngine. This value should have been initialized by a previous value returned from TestEngine	A value is expected to be returned by the TestEngine. The sequencer will remember this value under the variable name
{{}}	This should only appears in PARAM1 for the parse function. This is marks a parsing keyword in the template files.	This is also an expected value handled by the sequencer and TestEngine just like a variable surrounded with “[[]]”. In addition this value will be sent to PDCA as a data value (as opposed to an attribute).

Marker \ Location	PARAM1	PARAM2
<<>>	This should not appear in PARAM1	This is also an expected value handled by the sequencer and TestEngine just like a variable surrounded with “[[]]”. In addition this value will be sent to PDCA as an attribute (as opposed to a data value).

LOW and HIGH:

- If in the PARAM2 field there is a returned variable (variable name surrounded by “[[]]” or “{{}}”, the sequencer will look for the low and high limits of this test in the sequence.
- If only the low and high limits exist but no returned variable specified, the sequencer will not try to judge if this test is a pass or a fail, because it does not know what value to judge.
- If the *LOW* value is different from the *HIGH* value, the sequencer assumes they are both numeric values and the result should be between them.
- If the *LOW* value is exactly the same as *HIGH* value, the sequencer assumes the return value has to be a exact literal match to pass.
- There are functions that the pass/fail result can be judged by the test engine directly without any external limits. A example of that is the *detect* function. In this case the function will return a literal ‘—PASS—‘ or ‘—FAIL—‘ as the result, and the sequencer will accept these as the pass/fail result of the test before checking for returned variables or limits. See the TestEngineRPC spec for a list of this kind of functions

FUNCTION:

The FUNCTION field should have a valid function name supported by the test engine, except when the name is surrounded by “<<“ and “>>”. This means it’s calling a plugin function. The test developer can develop a python module with the name “fctplugin”, and put it in the system’s python path. Any function exposed by this plug in can then be called in the FUNCTION field.

These functions will be called with three parameters:

- **params:** a list of parameters. The variable will be replaced with actual values like a normal test item. However note that these params are probably strings so the function in the plugin is responsible to convert the string representation to the correct data types.
- **unit**: the function is free to use this argument for any purpose.
- **timeout:** a time out in mili second. The function must respect this timeout

These functions should return a single value. This returned value will be used to judge if the test has passed or failed based on the setting in the LOW/HIGH field.