In the following, the “local file system” is assumed to be a clone of a GitHub repository.

# Workflows

The following workflows are desired for working with Dokimion.

## Create a new test case in the local file system and send it to Dokimion

1. The Updater tool now uses Markdown as the format for the test cases on the file system. It is much more readable by humans than JSON (the native format for Dokimion) or XML. We defined the conventions to be used so that it will be easily parseable.
2. We will have a template that the test case author can use.
3. The steps in Updater are:
4. Log into Dokimion
5. Select the GitHub repo.
6. Select the project in the list on the left.
7. Click “Compare Test Cases in Project”. This will populate the test case list with the results of the comparison.
8. Click “Dokimion Missing” in the Test Case Filter.
9. The test case list should now show the new test case.
10. Click the checkbox in the Select column of the test case list.
11. Click the “File System to Dokimion” button at the bottom of Updater.

## Edit an existing test case in the local file system and send it to Dokimion

1. No special considerations. When doing the comparison, the Updater tool will not know if what is Dokimion or on the file system is newer, so it will show the status as “unknown”. The user can “show test case differences” to verify that the changes are as expected.
2. The steps in Updater are:
3. Log into Dokimion
4. Select the GitHub repo.
5. Select the project in the list on the left.
6. Click “Compare Test Cases in Project”. This will populate the test case list with the results of the comparison.
7. Click “Dokimion Missing” in the Test Case Filter.
8. The test case list should now show the new test case.
9. Click the checkbox in the Select column of the test case list.
10. Click “Show Test Case Differences” to verify the changes are as expected.
11. Click the “File System to Dokimion” button at the bottom of Updater.

## Backup a project on a server to the local file system

The backup files can be used for:

* Restoring individual test cases if they get corrupted or deleted.
* Restoring a whole project.
* Copying a project to a different server.

Backing up a project involves these steps in Updater:

1. Log into Dokimion
2. Select the location in the file system to save the project. This does not need to be a GitHub repo.
3. Select the project in the list on the left.
4. Click “Compare Test Cases in Project”. This will populate the test case list.
5. Click “Save Project Info” near the center of Updater. This will save project-level information, including the attributes defined for the project.
6. Click “Select All” to select all the test cases.
7. Click “Dokimion to File System” to save all the test cases to the file system.

## Restore a whole project from the local file system

1. This is tricky since Dokimion only marks a project as deleted instead of deleting it from its database. Creating a project with the same name/id undeletes the old project *with all its test cases*.
2. Restoring a project involves these steps in Updater:
3. Log into Dokimion
4. Click “Restore Project From Info Files”.
5. Select the location in the file system that has the info files that were previously saved when backing up the project.
6. Click “Restore”.
7. Select the project in the list on the left.
8. Click “Compare Test Cases in Project”. This will populate the test case list.
9. If some test were deleted from the project before backing it up there will be some missing test case numbers. Updater will need to create these to keep the numbers the same before and after the restore. Click “Delete Empty Test Cases” to delete these created, but unused, test cases.
10. Decide what to do with the test cases that don’t match.

## Restore a (corrupted? deleted?) test case from the local file system

Restoring a corrupted tests case from the local file system is the same as editing a test case on the local file system and sending it to Dokimion.

Restoring a deleted test case requires a change to the Dokimion API (see below). Otherwise, the new test case will have a different test case number than the old one had. Currently, an error is reported when attempting to do this.

## Update the local file system with changes on a server

This is similar to “Backup a project” above. It is OK to update test cases that are already the same. Or you can filter for “Not Equal” and “Select All” and then press “Dokimion to File System”.

## Copy a whole project from one server to another

This could be used, for example, for copying a project from the production server to a staging server to provide content to test features and bug fixes.

This is done in two steps. First “Backup the project” from the one server and the “Restore the project” to the other server. Be careful to select the correct servers when logging into the server. The files would not need to be put into GitHub, but can be anywhere in the file system.

# Changes to Dokimion API

We need to be able to undelete a test case. Currently, if we try to PUT the “deleted” field to “false”, the API returns a “Not found” error. I propose allowing PUTs to deleted test cases, but not GETs. Also, the PUT to a deleted test case should not perform the “last modified time” check.

Attachments are currently broken. You often cannot view an attachment with the UI. You also cannot delete an attachment via the UI. You cannot get an attachment via the API.

# Pie-in-Sky Goals

Some thoughts:

* Do testcase development/edits in GitHub.
* Press one button to send to Dokimion.
* Use GitHub Action?
* Separate file for metadata?
* Don’s definition of “test case” is: Title, description, attributes, steps
* Be creative!

If we put a process in place that requires a GitHub “pull request” to check anything into the test case repo, then we can trigger an Action when a pull to the main branch occurs. We would need to create a command line app that would then update Dokimion with all test cases that are different in GitHub.

# Possible Tool Improvements

Would it be helpful to filter test cases using project attributes?

Would it be helpful to edit test cases in the Updater tool?

I would like to optimize the “Compare Test Cases” process to make it quicker.

Alphabetize the project list.

# Updater UI

Updater has a table showing the test cases and some buttons to take action on the selected test cases. The test case table looks like:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Title In Dokimion** | **ID** | **Status** | **Select** | **Title In File System** |
| TT-1477(Retain paragraph breaks when posting to Paratext) | 10 | = | □ | TT-1477(Retain paragraph breaks when posting to Paratext) |
| TT-1585 (remove isDeveloper flag so code on dev and qa will use new ui) | 11 | > | √ | TT-1585 (remove isDeveloper flag so code on dev and qa will use new ui) |
| 1636 - (Remove the "Delete User" according from the New profile screen) | 15 | » | √ |  |
|  | 19 | « | □ | Members Button - INVITATIONS TAB: |

Status:

* = means the test case is identical in both Dokimion and the file system.
* > means the test case in Dokimion is newer than in the file system (eg. ID 11).
* < means the test case in the file system is newer than in Dokimion.
* » means the test case is only in Dokimion (eg, ID 15).
* « means the test case is only in the file system (eg. ID 19).
* <> means that the timestamps are the same, or one is missing, but the test case content is different.

The “Select” column is where you select which test cases to act on.

Actions:

* Select all test cases being shown.
* Clear all selections.
* Update selected test cases from file system to Dokimion.
* Update selected test cases from Dokimion to file system.
* Show differences between the Dokimion and the file system for selected test cases.
* (Actions will never act on hidden test cases.)

Filters:

* Show all test cases
* Show test cases where Dokimion is newer
* Show test cases where the file system is newer
* Show test cases which Dokimion is missing
* Show test cases which the file system is missing
* Show test cases which we don’t know which is newer
* Show test cases which differ for any reason between Dokimion and the file system

# Using Markdown

See <https://www.markdownguide.org/> for information about Markdown. I propose defining some conventions about how to use it.

I like using Markdown because is a textual format which allows GitHub differences to be meaningful, unlike something like a Word document, which has a binary file format. Markdown is supported by many editors, such as Visual Studio, Visual Studio Code, and Notepad++.

## Header Level Rules

Header levels contain specific keywords that the tool uses.

Markdown requires a blank line before and after a header.

If there is no content for a section, that section’s header may be omitted.

A line starting with one # followed by a space and some text is a Header Level 1. The Header Level 1 headers do not need to be in a specific order, but the following order is used when saving a test case from Dokimion. Header Level 1 items:

* ID
* Name
* Description
* Preconditions
* Steps
* Attributes
* Metadata
* Attachments

The ID and Name have the id and name on the same line. All the others are on a line by itself, followed by a block of text or a list of items.

A line starting with two ## followed by a space and the text is a Header Level 2. Header Level 2 items:

* Step (used under Steps, one for each Dokimion step)
* Attachment (used under Attachments, one for each attachment)

A line starting with three ### followed by a space and the text is a Header Level 3. Header Level 3 items:

* Action
* Expectation

Action and Expectation are under Step.

## Blocks of Text

A block of text is permitted after the following headers:

* Description
* Preconditions
* Action
* Expectation

The block of text ends at the next header or the end of the file.

Blocks of text need to be HTML formatted text. Unfortunately, since Dokimion in HTML-based, we cannot use Markdown in the file system and upload it to Dokimion.

(Perhaps in the future, if we **only** do transfers from the file system to Dokimion and never from Dokimion to the file system, we could do the Markdown to HTML translation before doing the upload to Dokimion. This would prohibit the “back up the server” use case.)

You need to tell the Markdown parser that this is a block of HTML by putting ```html on a line by itself before the block and ``` on a line by itself after the block.

Maybe because I’ve dealt with HTML for so long, it doesn’t seem to be too hard to manually generate it. For example, a numbered list looks like:

<ol>

<li> Open a browser to ...</li>

<li> Log in with user credentials </li>

</ol>

If you have multiple paragraphs, separate them with <p>. Otherwise Dokimion will show one big paragraph.

I suggest you do **not** use something like Word and “Save as HTML” because Word adds lots of junk (to retain the exact formatting) which just clutters up the HTML, making it almost unreadable by humans.

## Other Headers

The Metadata header will be followed by an unordered Markdown list of item names followed by a colon followed by the item value, such as:

* Locked: true

The Attributes header will be followed by an unordered Markdown list of attribute names followed by a colon followed by the attribute values, such as:

* TestLodge Section: Deprecated

The Attachment header will be followed by an unordered Markdown list of member names followed by a colon followed by the member value, such as:

* ID: xx-yy-zzz
* Title: Foo.txt
* Created time: June 1, 2024 16:02:28 GMT
* Created by: Joe
* Data size: -1

Note: Attachments are currently broken in Dokimion. You can upload them, but not view or download them.

# Example Markdown

This is an example of what the Markdown source could look like.

# ID 1

# Name Number1

# Description

```html

This is the short description.

```

# Steps

## Step

### Action

```html

This is what you do:

<ol>

<li> Open a browser to ...</li>

<li> Log in with user credentials </li>

<li> Verify the use is logged in.</li>

</ol>

Make sure you do all this.

<p>

Paragraph two.

```

### Expectation

```html

Expect all verifications to pass.

```

## Step

### Action

```html

<ol>

<li> Do this.</li>

<li> Do that.</li>

<li> Do the other thing.</li>

</ol>

```

### Expectation

```html

We expect this step to complete.

```

# Attributes

\* Platform: Windows

# Metadata

\* automated: False

\* broken: False

\* createdBy:

\* createdTime: 0

\* deleted: False

\* lastModifiedBy: admin

\* lastModifiedTime: 1743791496739

\* launchBroken: False

\* locked: False