Resource Container Documentation

Release 0.1

Door43

November 14, 2016

Contents

This site is the official documentation for Door43's resource container specification. Resource Containers are the building blocks of content in our Door43 eco-system.

Contents:

CHAPTER 1

Resource Container Structure

1.1 Container Format

A resource container may exist in two forms:

- a compressed archive with the extension .tsrc as in: some_resource_container.tsrc
- a directory as in: some_resource_container/

In both cases the internal structure is exactly the same. Compressing a resource container and changing the file extension to .tsrc simply makes the resource container portable.

1.2 Slug

The resource container slug is primarily used to create unique file names for storing resource containers on the disk. In the case of creating a translation this is also the name of the online repository where the resource container is uploaded on Door43. These slugs are also used when linking from one resource container to another. See Resource Container Links for more information about linking.

The slug is also a convenient way to identify a resource container without opening it to inspect the package.json. This can be helpful when searching for resource containers through an API that only returns the resource container slugs.

Slugs are structured as indicated below. Each component of the slug is delimited by an underscore _.

```
[language slug]_[project slug]_[resource slug]
```

Each component of the slug may include alphanumeric characters and dashes -. See Resource Types for a list of valid resource types that may be used in the slug.

NOTE: slug components may not contain any underscores _ since this is the delimiter.

An example of using the slug to identify a resource container is illustrated below. By viewing the file name we are able to quickly identify that this resource container contains the ULB version of the book Genesis translated in English:

```
en_gen_ulb.tsrc
```

As mentioned above, the slug is not the strict authority regarding the nature of the resource container. Therefore whenever a resource container is utilized the package.json should always be consulted and has the final word. This is important since a user could change the name of the file between exporting it and sharing it with someone else.

In order to correctly generate the slug of a resource container you must consult the package.json which contains all of the necessary information.

1.3 Directory Structure

Resource containers may be zip archives with the .tsrc (translationStudio) extension, or directories.

```
my_resource_container.tsrc/
    |-.git/
    |-LICENSE.md
    |-package.json
    |-content/
```

- the .git directory is optional and is usually only seen in active translations.
- LICENSE.md contains the appropriate license information for the resource container.
- package.json contains meta data about the resource container.
- the content directory contains any other files needed by the resource type.

1.4 Content

The folder structure of the content directory in resource containers are mostly the same. Differences between resource types may include the absense of some files or the inclusion of others.

Note: that where a .txt extension is shown below, the proper extension should be used according to the content_mime_type indicated in the package.json. For example .usfm or .md.

```
content/
    |-config.yml
    |-toc.yml
    |-01/
    |    |-title.txt
    |    |-sub-title.txt
    |    |-intro.txt
    |    |-reference.txt
    |    |-summary.txt
    |    |-01.txt
    |    |-02.txt
    |    ...
    ...
    |-front/
    |-back/
    ...
```

The directories shown above indicate chapters except for the two reserved folders *front* and *back* which contain, if applicable, the front matter and back matter of the container.

The files within each chapter represents the chunks of the chapter. Within each folder are additional reserved files:

- title
- sub-title
- intro
- reference
- summary

1.5 Config

The *config.yml* file contains information specific to the resource type. However, there is a reserved field *media* which allows different media to be assoicated with the resource container.

```
---
  media:
    image:
        mime_type: "image/jpeg"
        size: 37620940
        url: "https://cdn.door43.org/en/obs/v3/jpg/360px.zip"
    image_large:
        mime_type: "image/jpeg"
        size: 807010466
        url: "https://cdn.door43.org/en/obs/v3/jpg/2160px.zip"
    single_image:
        mime_type: "image/jpeg"
        size: 80701
        url: "https://cdn.door43.org/en/obs/v3/jpg/01_01.jpg"
```

In the above example there are three different media types:

- image
- image_large
- single_image

These media types are utilized via Resource Container Links .

The *url* can point to either a single media file or a zip archive which contains many pieces of media. The downloaded media files themselves can be named whatever you want so long as they adhere to the naming conventions for slugs.

If media is served as a zip archive the archive should contain appropriately named media files which may optionally be organized within folders also appropriately named.

```
my_media.zip/
    |-01/
    |   |-01.jpg
    |   |-02.jpg
    |
    |-02/
    |   |-01.jpg
    |   |-02.jpg
    ...
```

If you want to provide hierarchy to media files in a zip archive without using folders you may use an underscore _ to delimit the slugs.

```
my_obs_media.zip/
    |-01_01.jpg
    |-01_02.jpg
    ...
```

Implementation Notes: When downloaded, the media should be stored in a central location where each media type is stored under a folder named according to it's type. e.g. /media/image_large. The examples above deal only with images, however it is possible to reference other media formats including video or audio content. For more information about how to use media types see Resource Container Links .

1.6 Table of Contents

Chapter directories and chunk files are often named with padded integers. A side effect of this is the natural file order often represents the appropriate order. However, you may also indicate the order of chapters and frames by providing a table of contents toc.yml within the content directory. If no such file exists then the integer order followed by the natural order of the files will be used.

The table of contents is built with small blocks as shown below. All of the fields in the blocks are optional:

```
---
  title: "My Title"
  sub-title: "My sub-title"
  link: "my-link"
  sections: []
```

The sections field allows you to nest more blocks. The link fields may accept the chunk that should be linked to. Alternatively you may provide a fully qualified link as defined in Resource Container Links.

Here's an example toc.yml from translationAcademy. Generally speaking the title and sub-titles fields in this file should be the same as what is found in the subsquently named chunks. However, the TOC is allowed to deviate as nessesary.

```
---
  title: "translationAcademy Table of Contents"
  sub-title: ""
  link: ""
  sections:
    -
      title: "Introduction to translationAcademy"
      sub-title: "This page answers the question: What is in the Introduction?"
      link: ""
      sections:
        -
          title: "Introduction"
          sub-title: ""
          link: ""
          sections:
            -
              title: ""
              sub-title: ""
              link: "ta-intro"
              sections: []
            -
              title: ""
              sub-title: ""
              link: "uw-intro"
              sections: []
    -
      title: "Table Of Contents - Process Manual Vol 1"
      sub-title: "This page answers the question: What is in the process manual volume 1?"
      sections:
        -
          title: "Process Manual Volume 1"
          sub-title: ""
          link: ""
          sections:
            -
              title: "1. Getting Started"
              sub-title: ""
```

```
            link: ""
            sections:
              -
                title: ""
                sub-title: ""
                link: "process-manual"
                sections: []
              -
                title: ""
                sub-title: ""
                link: "getting-started"
                sections: []
          -
            title: "2. Setting Up a Translation Team"
            sub-title: ""
            link: ""
            sections:
              -
                title: ""
                sub-title: ""
                link: "setup-team"
                sections: []
  -
    title: "Table Of Contents - Translation Manual Volume 1"
    sub-title: "This page answers the question: What is in Volume 1 of the translation manual?"
    sections: []
```

Alternatively you can choose to use a simplified table of contents as shown below.

```
---
-
    chapter: '01'
    chunks:
        - title
        - '01'
        - '02'
        - '03'
        - '04'
        - '05'
        - '06'
        - '07'
        - '08'
        - '09'
        - '10'
        - '11'
        - '12'
        - '13'
        - '14'
        - '15'
        - '16'
        - reference
-
    chapter: '02'
    chunks:
        - title
        - '01'
        - '02'
        - '03'
        - '04'
```

```
        - '05'
        - '06'
        - '07'
        - '08'
        - '09'
        - '10'
        - '11'
        - '12'
        - reference
```

The simple version will rely on the available content (titles, references, etc.) to generate the table of contents. i.e. Readable titles will be retrieved from the content itself.

CHAPTER 2

Manifest File

All resource containers have a package.json file which looks something like this:

```
{
  "package_version": 7,
  "modified_at": 20151222120130,
  "content_mime_type": "text/usfm",
  "versification_slug": "some-slug",

  "language": {
    "slug": "en",
    "name": "English",
    "dir": "ltr"
  },

  "project": {
    "slug": "gen",
    "name": "Genesis",
    "description": "",
    "icon": "",
    "sort": 1,
    "categories": [
      "bible-ot"
    ]
  },

  "resource": {
    "slug": "ulb",
    "name": "Unlocked Literal Bible",
    "type": "book",
    "status": {
      "translate_mode": "all",
      "checking_entity": [
        "Wycliffe Associates"
      ],
      "checking_level": "3",
      "version": "3",
      "comments": "",
      "contributors": [
        "Wycliffe Associates"
      ],
      "pub_date": "2015-12-17",
      "license": "CC BY-SA 4.0",
      "checks_performed": [
```

```
        "keyword",
        "metaphor",
        "etc..."
      ],
      "source_translations": [
        {
          "language_slug": "en",
          "resource_slug": "asv",
          "version": "1901"
        }
      ]
    },
  },

  "chunk_status": [
    "00-title",
    "01-01",
    "01-02",
    "01-title",
    "01-reference",
    "02-01"
  ]
}
```

2.1 Definitions

- package_version: indicates the version of the resource container spec that is implemented by the current container.
- modified_at: indicates when this resource container was last modified
- content_mime_type: the format of the text that is being stored inside this resource container. The supported formats are currently: - text/usfm - text/markdown
- versification_slug: the versification system used to define the chunks in this resource.
- resource: the resource that is stored in this resource container - type: the type of resource being represented in this resource container. - translate_mode: indicates if/when this resource can be translated
  - all: it can always be translated.
  - gl: it can only be translated when gateway language mode is activated in the app.
  - none: it can never be translated.
- language: the language of the text that is being stored inside this resource container.
- project: the project to which the content stored in this resource container belong.
- chunk_status: checking status of chunks in this resource container.
- status > translate_mode: The mode in which the resource may be translated.

2.2 Changing Identifying Properties

An identifying property is any property used in the generation of the resource container slug.

Sometimes it is desirable to change certain identifying properties. For example a mistake may have been made when choosing the language or the resource.

Not all identifying properties are allowed to be modified after a resource container has been created. The properties that can be changed are:

- language
- resource

Care should be taken since this will also change the slug of the resource container.

If while changing an identifying property the resource container will conflict with an existing resource container the user should be asked if they would like to merge the two resource containers or cancel the change. See Merging Resource Containers for more information about merging.

In order to fully change an identifying property the following steps must be taken

1. change the value of the property in the package.json.
2. update any usages of the resource container's slug to the new slug.

CHAPTER 3

Resource Types

Content can be displayed in different forms and used in different ways within an application. Therefore several resource types exist to support different needs.

All resource types, unless otherwise specified, may employ any translation mode. Translation modes indicate to an application under what conditions a resource container is translatable.

3.1 Book

Represents any text that is structured like a book. e.g. there are chapters and chunks.

Related resources can be indicated in the config.yml file:

```
---
content:
  01:      // chapter
    01:    // chunk
      words:
        - "//bible/tw/creation"
        - "//bible/tw/god"
        - "//bible/tw/heaven"
        - "//bible/tw/holyspirit"
      questions:
        - "//gen/tq/01:01"
      notes:
        - "//gen/tn/01:01"
      images:
        - "image://gen/ulb/01:01"
```

Implementation Notes: Related resources as shown above may be displayed in the application along the side of the book content in order to provide contextual information.

3.2 Help

A helpful resource to supplement chunks in a book. e.g. notes or questions. Currently all help resources must use the markdown format.

Each chunk contains one or more helps which correlate to the corresponding chunk in a book resource:

```
#In the beginning God created

This introductory statement gives a summary of the rest of the chapter. AT: "This is about how God ma

#In the beginning

This refers to the start of the world and everything in it.
```

When parsed by an app the helps in this chunk are split at the headers. If there is preceding text (without a header) it will be displayed as a single help and a short snippet of the text will be used for the header if applicable.

3.3 Dictionary

A standalone dictionary of terms. Currently all dictionary resources must use the markdown format.

The dictionary terms are used as the chapter slug and the translation of the term is placed inside a single 01.txt file:

```
content/
    |-config.yml
    |-aaron/
    |    |-01.txt
    |
    |-abel/
    ...
    |-unclean/
```

NOTE: lengthy dictionary terms may be split into more than one chunk.

The 01.txt file contains the translation of the term where the header is the title of the term and the rest is the description:

```
#Aaron

God chose Aaron to be the first high priest for the people of Israel.
```

The config.yml is used to indicate related terms, aliases, and examples.

```
---
  aaron:
    see_also:
      - "covenant"
      - "testimony"
    aliases:
      - aaronalias # note: not a real alias for this word
    examples:
      - "09-15"
      - "10-05"
```

Examples are tricky because a dict may be referenced by many different projects/resources. Therefore we cannot specify a resource link but instead must simply provide the chapter and chunk that contains the example.

3.4 Manual

A user manual. For now manual resources must use the markdown format.

Manuals are a collection of modules (articles):

```
content/
    ...
    |-translate-unknowns
    |    |-title.txt
    |    |-sub-title.txt
    |    |-01.txt
    ...
    |-writing-decisions/
```

The 01.txt file contains the translation of the module. The title.txt file contains the name of the module. And sub-title.txt contains the question that is answered by this module.

NOTE: if desired the module can be split into multiple chunks. The config.yml indicates recommended and dependent modules:

```
---
  translate-unknowns:
    recommended:
      - "translate-names"
      - "translate-transliterate"
    dependencies:
      - "figs-sentences"
```

Dependencies are id's of modules that should be read before this one. Recommendations are modules that would likely benefit the reader next.

CHAPTER 4

Linking

A resource link provides navigable directions to a Resource Container.

All resource links follow a very simple structure in two different flavors: Anonymous and Titled. The link path is the Resource Container slug where the slug delimiter is replaced by a slash / character.

Links may include arguments as required by the resource type. Arguments must be specified at the end of a link and separated from the link path by a slash as well.

4.1 Anonymous Link

These links have no title and are declared by enclosing the link in double brackets

`[[language/project/resource]]`

4.2 Titled Link

These links have a title and are indicated by enclosing the link title in single brackets and the link in parentheses.

```
[Link Title](language/project/resource)
```

4.3 Shorthand Links

Shorthand links may be used when the resource slug matches it's project slug. For example:

`[[en/obs/obs/01:02]]` could be written as `[[en/obs/01:02]]`.

Shorthand links may only be used when linking to a passage in a book or linking to a resource as a whole. i.e.

- `[[en/obs/01:02]]` Links to OBS 1:2
- `[[en/obs]]` Links to OBS

4.4 External URLS

You may link to online media by simply using a url instead of a resource container id.

- `[[https://www.google.com]]`

- `[Google](https://www.google.com)`

Links where the path begins with `http://` or `https://` are treated as external urls.

4.5 Arguments

Some Resource Containers can accept or require additional arguments in the link. These are described below. For more information about these types please see Resource Containers.

4.5.1 book

These types accept an additional chapter and verse parameter formatted as is common in Bible passages.

Here are some examples:

- `01:02` verse two in chapter one
- `01:02-04` verses two through four in chapter one. This is allowed for convenience. It will resolve as `01:02`.
- `02` chapter two. This will resolve to the first chunk of the chapter e.g. `02:title`. NOTE: the first chunk is not necessarily the first verse. Additional front matter may exist.

Please note that the values in these arguments are not digits but slugs to chapters and chunks. Therefore care must be taken to not use them as digits. For example `1:2` will not resolve because there is no chapter `1` or chunk `2`. Because these are just slugs we may link to other elements such as chapter titles or even project titles.

- `01:title` the title of chapter one
- `title` the title of the book

Links must always be direct therefore you may not indicate multiple ranges of passages.

- `01:02,04` **this is incorrect and will not resolve**

Complete Examples:

- `[[en/obs/obs/01:02]]`
- `[Open Bible Stories 1:2](en/obs/obs/01:02)`
- `[[en/gen/ulb/01:02-06]]`
- `[Genesus 1:2-6](en/gen/ulb/01:02-06)`

4.5.2 help

See arguments for **book**. Differences are described below.

When linking to a help you must always link to a chapter + chunk combination. You cannot link to just a chapter.

4.5.3 dict

Accepts a single dictionary term id as an argument. For example:

- `aaron`
- `abel`
- `canaan`

Complete Examples:

- `[[en/bible/tw/canaan]]`
- `[Canaan](en/bible/tw/canaan)`

4.5.4 man

Accepts a single module id as an argument For example:

- `translate-unknowns`

Complete examples:

- `[[en/ta-translate/vol1/translate-unknowns]]`
- `[Translate Unknowns](en/ta-translate/vol1/translate-unknowns)`

4.6 Abbreviations

In certain cases it is appropriate to abbreviate a link. Below are a list of cases where you are allowed to use an abbreviation.

4.6.1 Links within the same resource

When linking to a different part of the same resource you may just provide the arguments.

Example from tA Translate resource:

- `[[translate-unknowns]]`
- `[Translate Unknowns](translate-unknowns)`

Example from tW resource

- `[[canaan]]`
- `[Canaan](canaan)`

4.6.2 Links to any translation of a resource

Some times you may not wish to restrict the linked resource to a particular language. In that case you may exclude the language code from the beginning of the path and place an extra slash `/` in it's place.

Example:

- `[[//ta-translate/vol1/translate-unknowns]]`
- `[Translate Unknowns](//ta-translate/vol1/translate-unknowns)`

4.7 Automatically Linking Bible References

Bible references in any resource container should be automatically converted into resolvable links according to the linking rules for **book** resource types. Of course, if the reference is already a link nothing needs to be done.

Conversion of biblical references are limited to those resources that have been indexed on the users' device. Conversion should be performed based on any one of the following:

- a case *insensitive* match of the entire project title.
- a case *sensitive* match of the project slug where the first character is uppercase e.g. `Gen`.

For each case above there must be a valid `chapter:verse` reference immediately after the matching word separated only by white space. For example:

```
Genesis 1:1
genesis 1:1
Gen 1:1
Gen 1:1-3
gen 1:1 -- not valid
```

If the user clicks on one such generated link where the resource container has not yet been downloaded they should be asked if they would like to download it. After downloading the resource container they should immediately follow the link.

4.7.1 Example

Given the French reference below:

`Genèse 1:1`

If the user has only downloaded the English resource the link will not resolve because the title `Genesis` or `genesis` does not match `Genèse` or `genèse`. Neither does the camel case slug `Gen` match since it does not match the *entire* word.

If the user now downloads the French resource the link will resolve because `Genèse` or `genèse` does indeed match `Genèse` or `genèse`. The result will be:

```
[Genèse 1:1](fr/gen/ulb/01:01)
```

When a match occurs there may be several different resources that could be used in the link such as `ulb` or `udb`. When more than one resource slug is available use the following rules in order until a solution is found:

1. choose the first resource that has a translate_mode of 'all'.
2. choose the first resource that has a translate_mode of 'none.
3. choose the first resource found.

Care must be taken when formatting the chapter and chunk slugs. You must not assume a chapter is padded with a single `0` and likewise for chunks. When preparing the link you should attempt to compare the integer values found in the text with the chapter and verse slugs (this time parsed as integers) in order to identify the correct chapter and chunk.

Because chunks may contain a range of verses some judgment is required to determine if a verse resides within a chunk. For example chunk `01` may contain verses `1-3` whereas chunk `02` contains verses `4-6`.

If no chapter or chunk can be found to satisfy the reference it should not be converted to a link.

4.8 Media Links

Media types are described in Resource Containers. Media defined in such a way is accessible to not only the resource containing them but any resource that links to it.

In order to use a media link you need only pre-pend a link with the media type as indicated below (this assumes a media type `image` exists for both of these resources):

- `[[image:en/obs/obs/01:02]]`
- `[[image:en/gen/ulb/01:02]]`

> NOTE: if a media link is titled the title will be used as the alt text.

You may notice a striking similarity between the media links shown above and their accompanying passage links show below:

- `[[en/obs/obs/01:02]]`
- `[[en/gen/ulb/01:02]]`

Media links are handled in exactly the same way as a normal link except that the arguments are used to reference the correct media file.

The similarity seen above is not required neither will this always be the case since the media files may be named whatever you wish (while adhering to the requirements in Resource Containers). However, it is strongly encouraged where appropriate since it makes the creation of content much easier.