Resource Container Documentation

Release 0.2

Door43

Oct 23, 2018

Contents

This site is the official documentation for Door43's Resource Container specification. Resource Containers (RCs) are the building blocks of content in the Door43 eco-system.

If you learn better by seeing, skip straight to the *Resource Container Examples* page.

Detailed Specification:

CHAPTER 1

Resource Container Structure

Resource containers (RCs) exist as directories. They may be optionally compressed or packaged so long as the compressed file follows standard naming conventions for the file extension. For example:

- a zipped RC would end in `.zip`,
- a tarred RC would end in `.tar`,
- a tarred+bzip2 RC would end in `.tar.bz2`

A git repository is also a valid way to store RCs.

Note: When naming an RC directory or repository the best practice is to use a combination of the resource and project identifiers e.g. `en-ulb-gen`. If the RC contains more than one project just use the resource identifier with an optional *Identifier* formatted qualifier e.g. `en-ulb-nt` where `nt` is the custom qualifier denoting the New Testament.

1.1 Directory Structure

RCs have a folder structure like the following:

```
my_resource_container/
    |-.git/
    |-.apps/
    |-LICENSE.md
    |-manifest.yaml
    |-media.yaml
    |-content/
```

- `.git`: only exists when the RC is stored as a git repository.
- `.apps`: is where applications can store custom meta data about the RC. See *App Meta* for more information.
- `LICENSE.md`: contains the appropriate license information for the RC.

- `manifest.yaml`: is the RC *Manifest File*.
- `media.yaml`: contains definitions of *Media* that has been generated from the content of the RC and where to find them.
- `content`: contains the project files. The name of this directory is subject to the *Manifest File*. It is also possible for there to be multiple *Projects Directories* or be excluded altogether.

1.2 Project Directory

The project directory is where all of the translation information exists for a single project within an RC. Files in this directory may contain either meta data or readable text.

```
content/
    |-config.yaml
    |-toc.yaml
    |-front/
    |-back/
```

The folders and files illustrated above are reserved. Although not required, when used they must fulfill their roles as defined:

- `config.yaml` contains meta data for the project at varying levels of granularity as specified in the *Container Types*.
- `toc.yaml` contains the *Table of Contents*.
- `front` directory contains the front matter.
- `back` directory contains the back matter.

Furthermore, there are reserved chunk files which may exist in any folder including the reserved *front* and *back* folders:

Note: we use `md` for the file extension in this example. You should use a file extension that is appropriate for content in your *Container Type*.

```
content/
    ...
    |-front/
    |    |-title.md
    |    |-sub-title.md
    |    |-intro.md
    |    |-reference.md
    |    |-summary.md
    ...
```

Once again, these files are not required but must fulfill their roles as defined:

when in a chapter

- `title.md` - contains the chapter title
- `sub-title.md` - contains the sub title of the chapter
- `intro.md` - contains the introduction to the chapter
- `reference.md` - contains a reference displayed at the end of a chapter
- `summary.md` - contains a summary displayed at the end of a chapter

when in "front"

- `title.md` - contains the resource title
- `sub-title.md` - contains the sub title of the resource
- `intro.md` - contains the introduction to the resource
- `reference.md` - contains a reference displayed at the end of the front matter
- `summary.md` - contains a summary displayed at the end of the front matter

1.2.1 Condensed vs Expanded Form

At times content can be structured slightly differently for added convenience. If the *container type* supports it an RC may use one form or the other or both simultaneously.

Expanded

All *Container Types* support the expanded form. This form is most suitable for active translations because collaborators are less likely to interfere with other's files. And it decouples formatting from structure. For example, here's a *Project Directory* that has a chapter 1 folder containing several chunks:

```
content/
    ...
    |-01/
    |     |-title.md
    |     |-01.md
    |     ...
    |     |-reference.md
    |-02/
    ...
```

Condensed

Most *Container Types* support a condensed form in which content in each chapter folder is stored in a single file. this form is most suitable for content being delivered as source text:

```
content/
    ...
    |-01.md
    |-02.md
    ...
```

Where the file `01.md` may contain the title, sub-title, intro, chunks, etc. for chapter 1.

1.2.2 Naming Chapters and Chunks

When naming folders/files for chapters/chunks we *suggest* you zero pad the chapter or chunk to match the longest chapter or verse in the given book. Doing this allows files to be sorted correctly when viewed in a web application such as git.door43.org.

Note: A chunk is a range of 1 or more verses.

Chapter example: In Psalms there are 150 chapters. This number contains 3 digits so we zero pad all chapters in Psalms to 3 digits. e.g. chapter 1 turns into `001`, chapter 10 turns into `010`, chapter 100 turns into `100` etc.

Verse example: Psalm 119 contains 176 verses. This number contains 3 digits so we zero pad all verses in chapter 176 to 3 digits. e.g. verse 1 turns into `001`, verse 10 turns into `010`, verse 100 turns into `100` etc.

Another verse example: Psalm 1 contains 6 verses so we zero pad all verses in chapter 1 to 1 digit. e.g. verse 1 turns into `1`, verse 2 turns into `2` etc.

Taking from the examples above we would write the chapter/chunk file stucture for Psalm 119:1 as `119/001.md`, while the file structure for Psalm 1:1 would be `001/1.md`.

1.2.3 Content Sort Order

When utilizing content in an RC the order is very important. The content sorting rules are defined as:

Chapters

1. front matter directory
2. numeric chapter directories sorted numerically in ascending order
3. non-numeric chapter directories sorted alphabetically
4. back matter directory

Chunks

1. title
2. sub-title
3. intro
4. numeric chunks sorted numerically in ascending order
5. non-numeric chunks sorted alphabetically
6. reference
7. summary

1.3 Config

The `config.yaml` file contains information specific to each *RC type*. If a particular *RC type* does not need this file it may be excluded.

1.4 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.yaml`, within the *Project Directory*.

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: []
```

- `title` a header in the table of contents
- `sub-title` a sub heading in the table of contents
- `sections` allow you to nest more blocks.
- `link` is the chapter *Identifier* that should be linked to. Alternatively, you may provide a fully qualified link as defined in *Linking*.

Here is an example `toc.yaml` from translationAcademy. Generally speaking the title and sub-title fields in this file should be the same as what is found in the subsequently named chunks. However, the TOC is allowed to deviate as necessary.

```
---
title: "Table of Contents"
sections:
  - title: "1. Getting Started"
    sections:
      - title: "Introduction to the Process Manual"
        link: process-manual

  - title: "2. Setting Up a Translation Team"
    sections:
      - title: "Setting Up A Translation Team"
        link: setup-team

  - title: "3. Translating"
    sections:
      - title: "Training Before Translation Begins"
        link: pretranslation-training
      - title: "Choosing a Translation Platform"
        link: platforms
      - title: "Setting Up translationStudio"
        link: setup-ts

  - title: "4. Checking"
    sections:
      - title: "Training Before Checking Begins"
        link: prechecking-training
      - title: "How to Check"
        link: required-checking

  - title: "5. Publishing"
    sections:
      - title: "Introduction to Publishing"
        link: intro-publishing
      - title: "Source Text Process"
        link: source-text-process

  - title: "6. Distributing"
    sections:
      - title: "Introduction to Distribution"
        link: intro-share
      - title: "How to Share Content"
        link: share-content
```

CHAPTER 2

Manifest File

Note: Keys must always be represented even when the value is optional. If the key is not needed/available the value may be empty.

Resource Containers (RCs) have a `manifest.yaml` file that describes its content and structure. Most of the information adheres to the Dublin Core Meta Data Initiative and can be found nested within the `dublin_core` key.

```
---
dublin_core:
    conformsto: 'rc0.2'
    contributor:
        - 'A Contributor'
        - 'Another Contributor'
    creator: 'Someone Or Organization'
    description: 'One or two sentence description of the resource.'
    format: 'text/usfm'
    identifier: 'ulb'
    issued: '2015-12-17'
    language:
        identifier: 'en'
        title: 'English'
        direction: 'ltr'
    modified: '2015-12-22T12:01:30-05:00'
    publisher: 'Name of Publisher'
    relation:
        - 'en/udb'
        - 'en/tn'
        - 'en/tq'
        - 'en/tw'
    rights: 'CC BY-SA 4.0'
    source:
        -
            identifier: 'asv'
```

(continues on next page)

(continued from previous page)

```
            language: 'en'
            version: '1901'
    subject: 'Bible'
    title: 'Unlocked Literal Bible'
    type: 'book'
    version: '3'

checking:
    checking_entity:
        - 'Organization or Church network'
        - 'Organization or Church network'
    checking_level: '3'

projects:
    -
        categories:
            - 'bible-ot'
        identifier: 'gen'
        path: './content'
        sort: 1
        title: 'Genesis'
        versification: 'kjv'
```

2.1 Definitions

- `dublin_core`
  - `conformsto`: the version of this specification used by the RC.
  - `contributor`: an array of names or aliases of people that have contributed to the resource.
  - `creator`: the entity in charge of creating the resource.
  - `description`: a brief description of what the resource is.
  - `format`: the file format of content within the RC, e.g. `text/usfm`, `text/markdown` etc.
  - `identifier`: a *Identifier* formatted string uniquely identifying the resource.
  - `issued`: the *Date* of publication. Normally this should stay in sync with `version` increments.
  - `language`: the language of the resource.
  - `modified`: the date the resource was last modified. Occassionally, this may be more recent than `issued` if there are metadata changes that do not include changes to the content.
  - `publisher`: the name of the entity that published the resource.
  - `relation`: a array of *Short Links* to related resources.
  - `rights`: the license under which the resource is distributed.
  - `title`: the title of the resource
  - `type`: the RC *container type*.
  - `version`: the published iteration of the resource.
- `projects`: an array of projects inside the RC.
  - `categories`: an array of category *identifiers* used for hierarchical ordering.

  - `identifier`: an *Identifier* formatted string uniquely identifying the project.
  - `path`: the relative path to the project within the RC. Depending on the RC type this may be a directory or a file.
  - `sort`: the sorting order of this project when viewed in relation to other projects in this RC.
  - `title`: the title of the project.
  - `versification`: the system used for placing verse markers and consequently chunk markers.

2.2 Generating From USFM

See *USFM to Manifest* for instructions on populating the manifest.yaml from the headers in a usfm file.

CHAPTER 3

Media

Resource Containers (RCs) have a `media.yaml` file that describes media generated from the content within the RC. Media is most often generated for the purpose of distributing a consumable form of the RC's content.

3.1 Media Block

RC's may be compiled into many different forms. Therefore, you may include many different media blocks each representing a single format. A minimal media block requires the following keys:

- `identifier` an *Identifier* representing the media type
- `version` the version of the media file. This does not necessarily correspond to the RC version.
- `contributor` a list of people who have contributed in generating this media type. e.g. printers, sound engineers, etc.
- `url` the web address where the media file can be downloaded.

Example:

```
identifier: 'pdf'
version: '1'
contributor: []
url: 'http://cdn.door43.org/en/ta/v1/pdf/en_ta_v1.pdf'
```

3.1.1 Variable Expansion

The value of the `url` key may also include variables to make the path dynamic. However, the variables are limited to keys within the media block and only scalar (non-list) values are allowed. e.g. you can use the `version` key but not `contributor` since `contributor` is not a scalar value.

Additionally, you may map the `version` key to the version found in the *Manifest File* by settings it's value to `{latest}`.

Example:

```
identifier: 'pdf'
version: '{latest}'
contributor: []
url: 'http://cdn.door43.org/en/ta/v1/{identifier}/en_ta_v{version}.pdf'
```

3.1.2 Media Quality

For certain media types it is important to note the quality such as audio and video media. For such types an additional key may be added to the media block.

- `quality` indicates the quality (such as bit rate or frame rate) of the media.

Example:

```
identifier: 'mp3'
version: '1'
quality: '64kbps'
contributor: []
url: 'https://cdn.door43.org/en/obs/v4/media/mp3/v1/{quality}/obs.zip'
```

3.1.3 Chapter Media

In order to support the concept of chapters. We introduce a new key to the media block:

- `chapter_url` this is similar to the `url` key except it contains a pseudo variable named `chapter` that can be expanded across all chapters in a resource.

Example:

```
identifier: 'mp3'
version: '1'
quality: '64kbps'
contributor: []
url: 'https://cdn.door43.org/en/obs/v4/media/mp3/v1/{quality}/obs.zip'
chapter_url: 'https://cdn.door43.org/en/obs/v4/media/mp3/v1/{quality}/obs_{chapter}.
↪mp3'
```

3.2 Media Scope

There are two different scopes for media blocks: resource, and projects. The resource scope allows you to define a media type that encompasses the entire RC while the projects scope encompasses individual projects within the RC. All resource scoped media blocks should be nested within a top level `resource` key. All projects scoped media blocks should be nested within a top level `projects` key.

Example:

```
---
resource:
  version: '1'
  media: []
projects: []
```

The projects scope is a list while the resource scope is simply a dictionary (because there's only one resource in the RC).

Each project scope must include the following project keys:

- `identifier` the project identifier as found within the *Manifest File*.
- `version` the version of the RC that is represented in the subsequent media.

The resource scope only needs the `version` key.

If you expect to keep your media up to date with the latest version of the RC, as found within the *Manifest File*, you may use the variable expansion `{latest}` with the `version` key above. *hint: if you use {latest} in a media block you should probably also use it in the parent scope*.

Note: The two project keys are currently **not** available for variable expansion within child media blocks.

3.3 Putting It All Together

```
---
resource:
  version: '{latest}'
  media:
    -
      identifier: 'pdf'
      version: '{latest}'
      contributor: []
      url: 'https://cdn.door43.org/en/ulb/v{version}/media/{identifier}/ulb.pdf'

projects:
  -
    identifier: 'gen'
    version: '{latest}'
    media:
      -
        identifier: 'mp4'
        version: '{latest}'
        contributor:
          - 'Narrator: Steve Lossing'
          - 'Checker: Brad Harrington'
          - 'Engineer: Brad Harrington'
        quality:
          - '360p'
          - '720p'
        url: 'https://cdn.door43.org/en/ulb/v{version}/media/{identifier}/{quality}/
↪gen_chapter_videos.zip'
        chapter_url: 'https://cdn.door43.org/en/ulb/v{version}/media/{identifier}/
↪{quality}/gen_{chapter}.mp4'
      -
        identifier: 'pdf'
        version: '{latest}'
        contributor: []
        url: 'https://cdn.door43.org/en/ulb/v{version}/media/{identifier}/01-GEN.pdf'
```

3.4 Media Storage

The responsibility for storing media is left to those creating it. Media will not be accepted for storage in DCS in order to reduce load on the server and maintain speed and reliability.

When preparing media for distribution we suggest uploading files to a content delivery network (CDN).

CHAPTER 4

Container Types

Resource Containers (RCs) can be used to represent different forms of data. These different forms are represented by the following types.

The types below are noted by `Type Name (Type Slug)` e.g. `Book (book)`

Note: Most types support a condensed from. See *Condensed vs Expanded Form* for details.

4.1 Book (book)

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

Books of the Bible should use *usfm* for the file format. Open Bible Stories should use *markdown* for the file format.

```
content/
    ...
    |-01/
    |    |-title.md
    |    |-sub-title.md
    |    |-intro.md
    |    |-01.md
    |    |-02.md
    |    ...
    |    |-reference.md
    |    |-summary.md
    ...
```

4.2 Help (help)

Note: This type does not support the *condensed form*.

A helpful resource to supplement chunks in a book e.g. notes or questions, and is structured in the same was as a *Book (book)*. All help RCs must use the markdown format.

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

```
# In the beginning God created

This introductory statement gives a summary of the rest of the chapter. AT: "This is␣
↪about how God made...in the beginning." Some languages translate it as "A very long␣
↪time ago God created." Translate it in a way that that shows that this actually␣
↪happened and is not just a folk story.

# 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.

4.3 Dictionary (dict)

A standalone dictionary of terms. All dictionary RCs must use the markdown format.

The dictionary terms are used as the chapter *Identifier* and is most often organized in the *condensed form*.

```
content/
    ...
    |-aaron.md
    |-abel.md
    ...
```

Note: If desired, lengthy dictionary terms may use the *expanded form* and be split into multiple chunks.

The `01.txt` file contains the description of the term. The term title must always be at the top of the file as a h1 heading (a single #). *Links* may be used to reference other words, or content in other containers.

```
# Aaron #

## Word Data: ##

* Strongs: H0175
* Part of speech: Proper Noun

## Facts: ##

Aaron was Moses' older brother. God chose Aaron to be the first high priest for the␣
↪people of Israel.

* Aaron helped Moses speak to Pharaoh about letting the Israelites go free.
* While the Israelites were traveling through the desert, Aaron sinned by making an␣
↪idol for the people to worship.
```

(continues on next page)

(continued from previous page)

```
* God also appointed Aaron and his descendants to be the [priests](../kt/priest) for␣
↪the people of Israel.

(Translation suggestions: [How to Translate Names](rc://en/ta/man/translate/translate-
↪names))

(See also: [Priest](../kt/priest.md), [Moses](../other/moses.md), [Israel](../other/
↪israel.md))

## Bible References: ##

* [1 Chronicles 23:12-14](rc://en/tn/help/1ch/23/12)
* [Acts 07:38-40](rc://en/tn/help/act/07/38)
* [Exodus 28:1-3](rc://en/tn/help/exo/28/01)
* [Luke 01:5-7](rc://en/tn/help/luk/01/05)
* [Numbers 16:44-46](rc://en/tn/help/num/16/44)

## Examples from the Bible stories: ##

* __[09:15](rc://en/tn/help/obs/09/15)__ God warned Moses and __Aaron__  that Pharaoh␣
↪would be stubborn.
* __[10:05](rc://en/tn/help/obs/10/05)__ Pharaoh called Moses and __Aaron__  and told␣
↪them that if they stopped the plague, the Israelites could leave Egypt.
* __[13:09](rc://en/tn/help/obs/13/09)__ God chose Moses' brother, __Aaron__, and␣
↪Aaron's descendants to be his priests.
* __[13:11](rc://en/tn/help/obs/13/11)__ So they (the Israelites) brought gold to __
↪Aaron__  and asked him to form it into an idol for them!
* __[14:07](rc://en/tn/help/obs/14/07)__ They (the Israelites) became angry with␣
↪Moses and __Aaron__  and said, "Oh, why did you bring us to this horrible place?"
```

The `config.yaml` file contains extra details about the term that may be helpful for some automation tools.

```
---
  aaron:
    false_positives: []
    occurrences:
      - 'rc://en/ulb/book/1ch/23/12'
      - 'rc://en/ulb/book/1ch/07/38'
      - 'rc://en/ulb/book/1ch/28/01'
      - 'rc://en/ulb/book/1ch/01/05'
      - 'rc://en/ulb/book/1ch/16/44'
      - 'rc://en/obs/book/obs/09/15'
      - 'rc://en/obs/book/obs/10/05'
      - 'rc://en/obs/book/obs/13/09'
      - 'rc://en/obs/book/obs/13/11'
      - 'rc://en/obs/book/obs/14/07'
```

Generally, `false_positives` and `occurrences` are mutually exclusive. That is, you should probably only have one or the other.

If `false_positives` exists, it is a list of places that should be excluded. For example, if a typical regex search for “Aaron” would turn up instances that should not be shown to the user, they should be listed here.

Alternatively, if `occurrences` exist, then it specifies the entire list of occurrences of this word in the given resource. If this key exists then a regex search should not be performed by the software.

4.4 Manual (man)

A user manual. All manuals 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.

Note: If desired the module can be split into additional chunks.

The `config.yaml` file indicates recommended and dependent modules:

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

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

4.5 Bundle (bundle)

A bundle is simply a flat directory (no sub-folders) with a single file for each project. e.g. there is no *Project Directory*. This type is particularly suited for USFM when providing “USFM Bundles”.

When defining a project in the *Manifest File* be sure the path is pointing to a file and not a directory.

```
---
  projects:
    -
      identifier: 'gen'
      title: 'Genesis'
      versification: 'kjv'
      sort: 1
      path: './01-GEN.usfm'
      categories:
      - 'bible-ot'
```

RC file structure:

```
my_rc/
    ...
```

(continues on next page)

(continued from previous page)

```
    |-01-GEN.usfm
    |-manifest.yaml
```

Note: When your application supports "USFM Bundles" it can identify the them in two ways

- attempt to read the *Manifest File* to determine type as `bundle` and the format as `text/usfm`.
- look for any `*.usfm` files in the root directory if the *Manifest File* does not exist.

In this way the application will satisfy both the `Bundle` RC type described above and generic "USFM Bundles" as is common in the industry.

CHAPTER 5

Linking

A Resource Container (RC) link allows one RC to reference content from another RC. Also, web links may be used to reference content online.

Note: Applications using RCs may want to search the RC data for external links and cache the online content so the RC can be used while offline without missing content such as images.

How links are written depends on the file format. For example, a link within a markdown file would be displayed with the title in brackets and the url in parenthesis:

```
[Link Title](https://example.com)
```

In markdown we support an additional link form that provides a link without a title. The title in these cases will be automatically generated from the context:

```
[[https://example.com]]
```

RC links follow the same form as web links in that they have a *scheme* and *uri*.

```
scheme://uri
```

5.1 Scheme

RC links are identified by the `rc` schema in the same way that websites are identified by `http`. The scheme tells the software how to process the uri.

```
rc://uri
```

5.2 URI

The uri in an RC link is composed of the following components.

- **language** - an IETF compatible language tag indicating the language of the resource
- **resource** - an *Identifier* for the resource
- **type** - an *Identifier* for the *Container Type*
- **project** - an *Identifier* for the project

Any additional information you include must be added after those mentioned above.

```
rc://language/resource/type/project/extra/information
```

5.2.1 Wildcards

Some times it can be helpful to create a generic link. Such as when referencing an entire resource like the English Unlocked Literal Bible.

To facilitate this RC links support a wildcard `*` that can be used in place of any component in the uri.

Note: If the wildcard occurs at the end of the link you can exclude it entirely.

```
rc://en/ulb/book/*
# or
rc://en/ulb/book
```

You can also do things like link to a book in any language

```
rc://*/ulb/book/gen
```

5.2.2 Resolution

An RC link is resolved like a file path. The first few components address which RC to use. And any remaining components address the specific content inside the RC.

This is illustrated below:

```
# link
rc://en/ulb/bundle/exo

# bundle RC on file system
en_ulb_bundle/
    ...
    |-01-GEN.usfm
    |-02-EXO.usfm <=== the manifest will indicate that exo points here
    ...
```

From this point we can lengthen the link to include a chapter *Identifier*.

Note: If the RC is a *Bundle (bundle)* the client application is responsible for understanding how to resolve to the chapter or any other location in the content.

```
# link
rc://en/obs/book/obs/01

# book RC on file system
en_obs_book_obs/
    ...
    |-content/
    |   |-01/ <=== link points here
    |   ...
    ...
```

Going a step further we can link to a specific chunk

```
# link
rc://en/obs/book/obs/01/01

# file system
en_obs_book_obs/
    ...
    |-content/
        |-01/
            |-01.md <=== link points here
```

In some of the examples above the link was pointing to a directory. In those cases the link should resolve to the first available file in order of the sorting priority described in *Naming Chapters and Chunks*.

Note: Depending on the client application, several files may be combined together when displayed to the user. For example: when linking to a chapter in a book of the Bible it would make more sense to show at least the title and summary, if not the rest of the chapter, rather than just the title.

5.2.3 Examples

book

- `[Genesis 1:2](rc://en/ulb/book/gen/01/02)`
- `[Open Bible Stories 1:2](rc://en/obs/book/obs/01/02)`

help

- `[[rc://en/tq/help/gen/01/02]]` - links to translationQuestions for Genesis 1:2
- `[[rc://en/tn/help/gen/01/02]]` - links to translationNotes for Genesis 1:2

dict

- `[Canaan](rc://en/tw/dict/bible/other/canaan)`

man

- `[Translate Unknowns](rc://en/ta/man/translate/translate-unknown)`

bundle

- `[Genesis](rc://en/ulb/bundle/gen/01/01)`

Note: Linking to a *Bundle (bundle)* will resolve down to the project level. The application will need to support parsing the bundle format (if references are supported) in order to continue resolving the link.

Formats that support references are:

- usfm
- osis

Note: When using RCs with multiple projects the application will need to inspect the *Manifest File* to determine which *Project Directory* to read while resolving a link.

5.3 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.

5.3.1 Internal Links

When linking to a different section within the same RC you may leave off the *Scheme* and simply provide a UNIX styled relative file path. File extensions are optional.

For example, let's say we have the following RC:

```
en-ta/
    ...
    |-intro/
    |       |-ta-intro/
    |       |         |-title.md
    |       |         |-sub-title.md
    |       |         |-01.md    <====== link from here
    |       |
    |       ...
    |-checking/
    |       |-acceptable/        <====== to here
    |       |         |-title.md
    |       |         |-sub-title.md
    |       |         |-01.md
    |       ...
    ...
```

With the addition of internal links we can reference the "Acceptable Style" article from within the "Introduction to translationAcademy" in any of the following ways:

```
[Acceptable Style](rc://en/ta/man/checking/acceptable)
[Acceptable Style](../../checking/acceptable)
```

A better use case for relative paths would be in tW using the *condensed form*.

```
en-tw/
    ...
    |-bible/
    |      |-other/
    |      |      |-aaron.md
    |      |      |-moses.md
    |      |      ...
    |      ...
    ...
```

Note: In the above example the *Manifest File* will refer to the `/bible` directory as the project path.

From within aaron.md we can link to moses in any of the following ways:

```
[Moses](moses)
[Moses](moses.md)
[Moses](./moses)
[Moses](./moses.md)
[Moses](../other/moses)
[Moses](../other/moses.md)
[Moses](rc://en/tw/dict/bible/other/moses)
[Moses](rc://en/tw/dict/bible/other/moses.md)
```

Note: For compatibility with displaying in online services such as github or DCS we suggest either including the file extension when practical or using a fully qualified link complete with the `rc://` schema.

5.3.2 Short Links

A short link is used to reference a resource but not a project. Short links are composed of just the language and resource.

- `en/tn`

Short links are used within the *Manifest File* when referring to related resources.

5.3.3 Bible References

Bible references in any RC may be automatically converted into resolvable links according to the linking rules for **book** resource types. Of course, if the biblical 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 if in the text either of the following conditions is satisfied:

- a case *insensitive* match of the entire project title. e.g. `Genesis` is found in the text.
- a start case (first letter is uppercase) match of the project *Identifier* e.g. `Gen`.

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

```
Genesis 1:1
genesis 1:1
Gen 1:1
Gen 1:1-3
```

The chapter and verse numbers should be converted to properly formatted *identifiers*.

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 *Identifier* `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](rc://fr/ulb/book/gen/01/01)
```

Multiple Matches

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 *Identifier* is available use the following rules in order until a unique match is found:

1. use the same resource as indicated by the application context.
2. use the RC allowed by the translate_mode set in the application.
3. choose the first resource found or let the user choose (e.g. pop up).

Aligning Verses to Chunks

Because chunks may contain a range of verses, a passage reference may not exactly match up to a chunk. Therefore some interpolation may be necessary. For both chapter and verse numbers perform the follow:

> Given a chapter or verse number **key**. And an equivalent sorted list **list** of chapters or verses in the matched resource

- incrementally compare the **key** against items in the **list**.
- if the integer value of the current **list** item is less than the **key**: continue.
- if the integer value of the current **list** item is greater than the **key**: use the previous item in the **list**.
- if the end of the **list** is reached: use the previous item in the **list**.

For example chunk `01` may contain verses `1-3` whereas chunk `02` contains verses `4-6`. Therefore, verse `2` would resolve to chunk `01`.

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

CHAPTER 6

App Meta

Applications can store meta data in a Resource Container (RC) within the `.apps` directory.

Note: The contents of this directory are outside the scope of the RC specification. The examples below are only a suggestion.

6.1 translationStudio

translationStudio keeps track of stages chunks go through during translation.

```
my_rc/.apps/ts/
    |-status.json
```

6.2 translationCore

translationCore keeps track of checking information.

```
my_rc/.apps/tc/
    |-checkdata/
    |    |-LexicalChecker.tc
    |    |-ProposedChanges.tc
    |    |-common.tc
    |
    |-tc-manifest.json
```

6.3 unfoldingWord

unfoldingWord uses dynamically provided localization files for use in the app interface.

```
my_rc/.apps/uw/
    |-app_words.json
```

CHAPTER 7

USFM to Manifest

When converting a USFM file such as `01-GEN.usfm` into an RC follow the rules below when populating the *Manifest File*.

7.1 Map

Some values are known by default since USFM is always used for Bible projects:

- *dublin_core:type* = `book`
- *dublin_core:conformsto* = `rc0.2`
- *dublin_core:format* = `text/usfm`
- *dublin_core:subject* = `Bible translation`
- *projects:categories* = `bible-ot` or `bible-nt` depending on the project identifier

The rest may be parsed from the USFM:

id_<CODE>_(Name of file, Book name, Language, Last edited)

- `code` -> *projects:identifier*
- `Book name` -> *dublin_core:title*
- `Language` -> *dublin_core:language:title*
- `Last edited` -> *dublin_core:modified*

h_text…

- `text` -> *projects:title*

The following items will need to gathered by the person or app doing the conversion:

- *dublin_core:identifier*
- *dublin_core:language:identifier*

- *dublin_core:language:direction*
- *dublin_core:rights*
- *dublin_core:contributor*
- *projects:versification*

CHAPTER 8

Identifier

An identifier is a word or a few words that uniquely describe an object. Identifiers are often used instead of a database id due to their readability and portability.

8.1 Syntax

Identifiers are composed of lowercase alphanumeric characters and hyphens.

```
abcdefghijklmnopqrstuvwxyz1234567890-
```

- The first character in an identifier *must* be a letter.
- The last character in an identifier *must not* be a hyphen.

8.2 Examples

```
gen
pt-br
aey-x-haya
custom-identifier-123
my-1st-identifier
```

CHAPTER 9

Date

Dates follow the encoding scheme defined in the W3CDTF profile of ISO 8601.

9.1 Example

```
1994-11-05
1994-11-05T08:15:30-05:00
```

Corresponds to:

```
November 5, 1994
November 5, 1994, 8:15:30 am, US Eastern Standard Time.
```

CHAPTER 10

Code Libraries

Below are a few libraries designed to work with Resource Containers (RCs). If you would like to have your library displayed in this list please open an issue on GitHub.

10.1 Android

- unfoldingWord-dev/android-resource-container

10.2 Node.js

- unfoldingWord-dev/node-resource-container

10.3 Python

- unfoldingWord-dev/python-resource-container

CHAPTER 11

Resource Container Examples

Here is a list of examples of valid Resource Containers that illustrate the different ways that the specification may be implemented.

11.1 USFM Scripture Bundle Example

Here is what a combined USFM Scripture `bundle` looks like: English Unlocked Literal Bible. If you are familiar with standard USFM zip files you'll notice that this looks the same except that it has the added `manifest.yaml` file that provides extra metadata about the resource.

See also the *Unlocked Bible Interchange Format (UBXF)* specification.

11.2 Book Example

Here is what a simple `book` type looks like: English Open Bible Stories.

11.3 Dictionary Example

Here is what a `dict` type looks like: English translationWords.

11.4 Manual Example

Here is what a `man` type looks like with multiple `projects` in the same repository: English translationAcademy.

Notice how the `projects` array in the manifest.yaml file lists all the projects that are in that repository. Note further that there is a `sort` key in the `projects` array that will tell apps the order in which they should present the projects to the user.

11.5 Help Example

Here is what a `help` type looks like with multiple `projects` in the same repository: English translationQuestions.

Notice how the `projects` array in the manifest.yaml file lists all the projects that are in that repository.

CHAPTER 12

Unlocked Bible Interchange Format (UBXF)

12.1 Overview

The Unlocked Bible Interchange Format (UBXF) is a special type of Resource Container designed to provide a consistent format for separate apps to exchange Bible translations and metadata with one another.

12.2 Characteristics

The UBXF is a specific type of RC *Bundle (bundle)* that utilizes the USFM 3 specification to allow extra metadata to be encoded directly in the translation. All the characteristics of a RC *Bundle (bundle)* apply, with these added requirements:

- UBXF must provide its data in USFM 3.
- The `format` field of the `manifest.yaml` file must be set to `text/usfm3`.
- Word metadata should be encoded in the translation using USFM 3 word attributes.
- Alignment metadata should be encoded using USFM 3 milestones. See *Alignment Data Example* below.

12.3 Licensing

A UBXF is a derivitive work of the source and target translations in the project. It is strongly recommended that you only use texts with one of these licenses:

- Public Domain
- CC0
- CC BY
- CC BY-SA

Use the `source` array in the `manifest.yaml` file to indicate the upstream resources.

12.4 Recommended Source Texts

For New Testament alignment, the Unlocked Greek New Testament (UGNT) or the Bunning Heuristic Prototype (BHP) are the recommended sources.

For Old Testament alignment, the Unlocked Hebrew Bible (UHB) or the Open Scriptures Hebrew Bible (OSHB) are the recommended sources.

12.5 Alignment Data Example

Here is an example of alignment data encoded in UFSM 3 milestones. Note the following features:

- Punctuation occurs outside the `\w` markers.
- Alignment data must use the `\zaln` milestone markers. The `\z` prefix comes from the use of the USFM documentation of the z namespace.
- Use a short code name for the actual Greek or Hebrew text that has been aligned. The example below uses the `x-ugnt` attribute to indicate that the UGNT text is the source. Note that the code should match what is listed in the `source` array in the `manifest.yaml` file.
- The `occurrence` and `occurrences` attributes can help software identify individual occurrences of identical words within a verse.

```
\v 1 \zaln-s | x-strong="G39720" x-occurrence="1" x-occurrences="1" x-ugnt="Παλο"\*
\w Paul|x-occurrence="1" x-occurrences="1"\w*
\zaln-e\*,
\zaln-s | x-strong="G14010" x-occurrence="1" x-occurrences="1" x-ugnt="δολο"\*
\w a|x-occurrence="1" x-occurrences="1"\w*
\w servant|x-occurrence="1" x-occurrences="1"\w*
\zaln-e\*
\zaln-s | x-strong="G23160" x-occurrence="1" x-occurrences="2" x-ugnt="Θεο"\*
\w of|x-occurrence="1" x-occurrences="4"\w*
\w God|x-occurrence="1" x-occurrences="2"\w*
\zaln-e\*
\zaln-s | x-strong="G06520" x-occurrence="1" x-occurrences="1" x-ugnt="πστολο"\*
\w and|x-occurrence="1" x-occurrences="2"\w*
\w an|x-occurrence="1" x-occurrences="4"\w*
\w apostle|x-occurrence="1" x-occurrences="1"\w*
\zaln-e\*
\zaln-s | x-strong="G11610" x-occurrence="1" x-occurrences="1" x-ugnt="δ"\*
\w of|x-occurrence="2" x-occurrences="4"\w*
\zaln-e\*
\zaln-s | x-strong="G24240" x-occurrence="1" x-occurrences="1" x-ugnt="ησο"\*
\w Jesus|x-occurrence="1" x-occurrences="1"\w*
\zaln-e\*
\zaln-s | x-strong="G55470" x-occurrence="1" x-occurrences="1" x-ugnt="Χριστο"\*
\w Christ|x-occurrence="1" x-occurrences="1"\w*
\zaln-e\*,
\zaln-s | x-strong="G25960" x-occurrence="1" x-occurrences="1" x-ugnt="κατ"\*
\w for|x-occurrence="1" x-occurrences="1"\w*
\zaln-e\*
\zaln-s | x-strong="G41020" x-occurrence="1" x-occurrences="1" x-ugnt="πστιν"\*
```

(continues on next page)

(continued from previous page)

```
\w the|x-occurrence="1" x-occurrences="3"\w*
\w faith|x-occurrence="1" x-occurrences="1"\w*
\zaln-e\*
\zaln-s | x-strong="G23160" x-occurrence="2" x-occurrences="2" x-ugnt="Θϵo"\*
\w of|x-occurrence="3" x-occurrences="4"\w*
\w God's|x-occurrence="2" x-occurrences="2"\w*
\zaln-e\*
\zaln-s | x-strong="G15880" x-occurrence="1" x-occurrences="1" x-ugnt="κλϵκτν"\*
\w chosen|x-occurrence="1" x-occurrences="1"\w*
\w people|x-occurrence="1" x-occurrences="1"\w*
\zaln-e\*
\zaln-s | x-strong="G25320" x-occurrence="1" x-occurrences="1" x-ugnt="κα"\*
\w and|x-occurrence="2" x-occurrences="2"\w*
\zaln-e\*
\zaln-s | x-strong="G02250" x-occurrence="1" x-occurrences="1" x-ugnt="ληθϵα"\*
\w the|x-occurrence="2" x-occurrences="3"\w*
\zaln-e\*
\zaln-s | x-strong="G19220" x-occurrence="1" x-occurrences="1" x-ugnt="πγνωσιν"\*
\w knowledge|x-occurrence="1" x-occurrences="1"\w*
\zaln-e\*
\zaln-s | x-strong="G02250" x-occurrence="1" x-occurrences="1" x-ugnt="ληθϵα"\*
\w of|x-occurrence="4" x-occurrences="4"\w*
\w truth|x-occurrence="1" x-occurrences="1"\w*
\w the|x-occurrence="3" x-occurrences="3"\w*
\zaln-e\*
\zaln-s | x-strong="G35880" x-occurrence="1" x-occurrences="1" x-ugnt="τ"\*
\w that|x-occurrence="1" x-occurrences="1"\w*
\zaln-e\*
\zaln-s | x-strong="G25960" x-occurrence="1" x-occurrences="1" x-ugnt="κατ'"\*
\w agrees|x-occurrence="1" x-occurrences="1"\w*
\w with|x-occurrence="1" x-occurrences="1"\w*
\zaln-e\*
\zaln-s | x-strong="G21500" x-occurrence="1" x-occurrences="1" x-ugnt="ϵσβϵιαν"\*
\w godliness|x-occurrence="1" x-occurrences="1"\w*
\zaln-e\*,
```

Unaligned words or phrases should show up outside of the *zaln* milestones. For example, the English "of God" did not get aligned to "Θϵo" in the example below.

```
\v 1 \zaln-s | x-strong="G39720" x-occurrence="1" x-occurrences="1" x-ugnt="Παλo"\*
\w Paul|x-occurrence="1" x-occurrences="1"\w*
\zaln-e\*,
\zaln-s | x-strong="G14010" x-occurrence="1" x-occurrences="1" x-ugnt="δoλo"\*
\w a|x-occurrence="1" x-occurrences="1"\w*
\w servant|x-occurrence="1" x-occurrences="1"\w*
\zaln-e\*
\w of|x-occurrence="1" x-occurrences="4"\w*
\w God|x-occurrence="1" x-occurrences="2"\w*
\zaln-s | x-strong="G06520" x-occurrence="1" x-occurrences="1" x-ugnt="πστoλo"\*
\w and|x-occurrence="1" x-occurrences="2"\w*
\w an|x-occurrence="1" x-occurrences="4"\w*
\w apostle|x-occurrence="1" x-occurrences="1"\w*
\zaln-e\*
```

Note that since the "base text" in these files is the translation (English in the example), that needs to be text complete. Missing source text (UGNT in this example) words is OK because the software should provide that text independently.