Towards a rich interface for ODD customization: the new JavaScript version of Roma

Raffaele Viglianti / TEI Technical Council

The Text Encoding Initiative and ODD Customization

- The Text Encoding Initiative (TEI) is a consortium which collectively develops and maintains a standard for the representation of texts in digital form.
- The TEI’s chief deliverable is a set of Guidelines which specify encoding methods for machine-readable texts in XML.
- TEI’s source is defined using a subset of the TEI itself, as detailed in Chapter 22 of the Guidelines: “Documentation Elements”.
- The same subset of elements can be used to define customizations of the TEI.
- These set of operations are typically referred to as “One Document Does-it-all” or ODD.

Roma "Roma JS" (code name)

- Roma (http://www.tei-c.org/Roma/) is an online interface to create TEI customizations without needing to write ODD.
- The current version of Roma has been afflicted by a number of bugs and issues, many of which remain unresolved and are too complex to be fixed by the TEI Technical Council.
- The TEI Technical Council has opted to create a replacement for Roma from scratch, using modern web technologies, expanding the number of features supported, and focusing on user interface.
- The new web application is code-named "RomaJS". We welcome suggestions for a new name!

Features in Development Coming Soon (2018)

- Custom content model of macros
- Create new classes and new elements in your own namespace
- Customize and define new datatypes
- Validate ODD and report on validity against TEI-all
- Edit documentation for attribute values
- Enter Schematron constraints (via XML editor)

Medium term goals

- Localization support
- RELAX NG support for legacy customizations
- Decision-making tools: answer questions to create a starting set of modules / elements

RomaJS: Current Features

- Start with a TEI preset or upload your own Pure ODD customization.
- Customize various features of existing elements.
- An XML editor is provided to edit documentation texts.
- Simply search, add and remove elements and classes.
- Customize an element’s content model graphically.
- Customize element attributes, including inherited from classes.

Thorough Unit Testing

From the start, we have aimed at consistently testing RomaJS code. Tests have two main roles:

- Minimize the chances of new code breaking existing code.
- Show and document how specific operations are invoked and what results they should yield.

Example: testing INCLUDE_ELEMENTS

Action: called when the user selects to include an element previously excluded.

Test: is the data as expected?

Reducer: takes data from the action and adjusts internal representation of the ODD.

Test: was it changed as expected?

Changes to customization ODD

- Test a number of scenarios:
  - module not selected (no moduleRef)
  - moduleRef[@include]
  - moduleRef[@except]
  - moduleRef + elementRef[@mode="delete"]

- Is the ODD updated as expected?

How to Contribute (coding optional!)

- Try it and report issues on GitHub.
- Help us build more test cases: suggest customization scenarios. See below to learn more about testing.
- Know JavaScript? Help us with coding new features and tests.