NISTIR 8290

SCAP Composer User Guide

Joshua Lubell

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8290

National Institute of
Standards and Technology
U.S. Department of Commerce

NISTIR 8290

SCAP Composer User Guide

Joshua Lubell
Systems Integration Division
Engineering Laboratory

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8290

February 2020

U.S. Department of Commerce
Wilbur L. Ross, Jr., Secretary

National Institute of Standards and Technology
Walter Copan, NIST Director and Undersecretary of Commerce for Standards and Technology

National Institute of Standards and Technology Interagency or Internal Report 8290
Natl. Inst. Stand. Technol. Interag. Intern. Rep. 8290, 31 pages (February 2020)

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8290

Abstract

SCAP Composer is a software application from the National Institute of Standards and Technology (NIST) for creating Security Content Automation Protocol (SCAP – pronounced "ess-cap") source data stream collections. A source data stream collection is a bundle of Extensible Markup Language (XML) documents, each of which must be valid with respect to a schema defined in an SCAP component specification. SCAP Composer's limited scope and small footprint make it easy to install, use, and integrate with other SCAP content development tools. SCAP Composer uses the DITA Open Toolkit, an open source publishing engine for content authored in the Darwin Information Typing Architecture (DITA). SCAP Composer may be used with the NIST SCAP Content Validation Tool to check the conformance of SCAP source data stream components to content requirements and recommendations.

Disclaimers

Any mention of commercial or other third-party products in this guide is for information purposes only; it does not imply recommendation or endorsement by the National Institute of Standards and Technology (NIST). For any of the web links in the software and this user's guide, NIST does not necessarily endorse the views expressed, or concur with the facts presented on those web sites.

SCAP Composer was developed at NIST by employees of the Federal Government in the course of their official duties. Pursuant to Title 17 Section 105 of the United States Code, this software is not subject to copyright protection and is in the public domain. This software is an experimental system. NIST assumes no responsibility whatsoever for its use by other parties, and makes no guarantees, expressed or implied, about its quality, reliability, or any other characteristic.

SCAP Composer can be redistributed and/or modified freely provided that any derivative works bear some notice that they are derived from it, and any modified versions bear some notice that they have been modified. NIST would appreciate acknowledgement if the software is used.

Key words

SCAP; Security Content Automation Protocol; Source Data Stream Collection; DITA; Darwin Information Typing Architecture; DITA Open Toolkit; cybersecurity.

Table of Contents

List of Tables

List of Figures

1. Introduction

SCAP Composer is a software application from the National Institute of Standards and Technology (NIST) for creating Security Content Automation Protocol (SCAP – pronounced "ess-cap") [1] source data stream collections. A source data stream collection is a bundle of Extensible Markup Language (XML) [2] documents, each of which must be valid with respect to a schema defined in an SCAP component specification.

SCAP Composer uses NIST's SCAP Content Validation Tool [3], also known as "SCAPVal", to check the conformance of SCAP Composer output to the SCAP source data stream collection data model. This data model is defined in NIST Special Publication (SP) 800-126 Revision 3, NIST Special Publication (SP) 800-126 Revision 3, *The Technical Specification for the Security Content Automation Protocol Version (SCAP) 1.3* [1]. SCAPVal produces two validation reports: one in an XML format useful for integration with other NIST-developed SCAP software tools and one in Hypertext Markup Language (HTML) intended for humans to view in a web browser.

SCAP Composer is distributed as a set of plug-ins to the DITA Open Toolkit (DITA-OT) [4][1], an open source tool for generating content authored in the Darwin Information Typing Architecture (DITA) [5]. As such, you can easily deploy SCAP Composer with XML authoring software products containing a built-in DITA-OT. Alternatively, if you lack a DITA-OT-enabled XML authoring software product, you can deploy SCAP Composer with a self-installed DITA-OT. Both scenarios are covered in this guide.

This guide assumes you are familiar with SP 800-126 Revision 3 [1], particularly the "SCAP Content Requirements and Recommendations" section. Prior familiarity with DITA, although helpful, is not necessary.

SCAP Composer is different from other SCAP authoring tools and environments in that its scope is limited to source data stream bundling and validation. It does not help users create documents using the individual languages comprising SCAP, such as the Open Vulnerability Assessment Language (OVAL) [6] or the Extensible Configuration Checklist Description Format (XCCDF) [7]. Nor is it part of a large, complex development environment with a long list of installation requirements (although SCAP Composer could be a useful piece of such an environment). SCAP Composer's limited scope and small footprint make it easy to install and to integrate with other SCAP content development tools.

Secs. 2 and 3 discuss SCAP Composer's rationale, its DITA-based architecture, and how the plug-ins function in concert with an XML authoring product and SCAPVal. Secs. 4 through 6 provide step-by-step instructions for obtaining, installing, and using SCAP Composer. Specific instructions are provided for two SCAP Composer deployment scenarios. The first deployment scenario assumes use with a standalone DITA-OT and the open source GNU Emacs [8][1] text editor. The second deployment scenario assumes use with Oxygen XML Editor/Author [9][1], a commercial software product with a built-in DITA-OT. Both deployment scenarios use the same source data stream collection containing both locally

[1] Certain commercial and third-party software products are identified in this paper to document SCAP Composer adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the software identified is necessarily the best available for the purpose.

stored resources and an Internet-accessible remote resource. This source data stream collection is included with the SCAP Composer download.

These two deployment scenarios are not the only ones that exist. However, they serve as exemplars for the two general scenarios of (1) deployment using authoring software with built-in DITA-OT support and (2) deployment using an XML authoring tool plus a self-installed DITA-OT.

2. Why SCAP Composer?

Like any information standard, SCAP's success depends on the existence of standard-conforming content and on the ease of producing that content. Several organizations provide freely or commercially available SCAP content. Some examples of such organizations are the nonprofit Center for Internet Security (CIS) [10], the United States Government Department of Defense Cyber Exchange [11], and the RedHat-supported ComplianceAsCode GitHub project [12]. These content producers have extensive knowledge of the SCAP XML languages and create large, broadly applicable, and comprehensive SCAP-conforming benchmarks. CIS defines a security benchmark as constituting "best practices for the secure configuration of a target system."

But there is also a need for SCAP content that achieves a highly organization-specific operational purpose. For example, suppose a small manufacturer needs to secure a Windows 10 workstation running Human-Machine Interface (HMI) software that allows a human operator to control a piece of factory equipment. A new version, 2.0, of the HMI software has recently been released that fixes a vulnerability whose exploitation could adversely impact the state and behavior of the factory equipment controlled by the HMI.

The manufacturer uses a configuration scanner software product certified by the SCAP Validation Program [3] that accepts SCAP source data streams as input and produces an SCAP result data stream as output. The manufacturer wishes to use this product to monitor the configuration of the HMI workstation, and thus ensure that good security practices are being followed. Doing so requires SCAP content that checks not only whether Windows 10 is properly configured, but also whether the new version of the HMI software is installed. The manufacturer has limited in-house SCAP expertise. A single employee is familiar with SCAP, but work responsibilities leave this employee little time to spend learning the intricacies of SCAP source data streams and XML languages.

The employee obtains an existing general-purpose SCAP-encoded benchmark for mission-critical Windows 10 systems from a trusted source. However, this benchmark lacks the rules for checking the HMI software version. The employee determines that the Windows 10 benchmark would be suitable for monitoring the HMI workstation's security, but only if it is supplemented with the rules needed to determine that (1) the scanning target is in fact a system running Windows 10 with the HMI software installed and (2) the HMI software version is 2.0 or greater.

This is the sort of situation for which SCAP Composer is intended. Although creating the new HMI-specific rules requires some XCCDF and OVAL expertise, a significant part of the

work is extracting relevant components from the Windows 10 benchmark and re-combining them with newly created HMI-specific components into a new source data stream collection.

Source data stream manipulation is governed by the SCAP source data stream collection data model, an XML schema [13] defined in SP 800-126. This data model, discussed in detail in Refs. [14] and [15], has the following desirable properties:

- **Self-containment**. A source data stream collection contains all the data a scanner tool needs to perform an assessment on a target system. There are no references to information objects outside the collection, thus enabling lossless exchange of SCAP content between SCAP-conforming software products.
- **Reversibility**. Components in a source data stream collection are bundled such that the XML resources they contain, e.g., XCCDF checklists and OVAL definition collections, are unmodified from their original states. Resources can therefore be extracted and re-bundled into new collections without any changes to the XML, facilitating reuse.
- **Globally unique identifiers (GUIDs)**. Data stream collections, data streams, components and component references all must have GUIDs following naming conventions stipulated in SP 800-126. GUIDs reduce the possibility of name clashes and ambiguous references.
- **XML Catalogs** [16]. The SP 800-126 data model uses this standard to handle resource-to-resource references from within a source data stream collection, while maintaining self-containment and reversibility. Many low-cost XML parsers and code libraries support XML Catalogs, enabling implementers of SCAP scanners to leverage these tools and reduce software development costs.

But the same properties that are desirable for SCAP scanner software developers and users are undesirable for SCAP content authors. Self-containment and reversibility result in added schema complexity. GUID formatting requirements result in long, repetitive identifiers. XML Catalog syntax is verbose and author unfriendly.

The following example of a component reference within a source data stream illustrates how XML conforming to the SP 800-126 data model is verbose, hard to read, and challenging for content authors.

```
<sds:component-ref id="scap_gov.nist_cref_xccdf"
                   xlink:href="#scap_gov.nist_comp_xccdf">
   <cat:catalog>
      <cat:uri name="oval.xml" uri="#scap_gov.nist_cref_oval"/>
   </cat:catalog>
</sds:component-ref>
```

The XML below shows the same component reference represented using the SCAP Composer document type, discussed in Sec. 5. The long GUIDs, namespace prefixes, repetition, and XML Catalogs markup are gone, making for a more concise and easier to read source data stream collection. Furthermore, XML editing software typically allows authors to auto-complete long tag names such as “scapBenchmarkRef”, making it even easier to use the SCAP Composer document type.

```
<scapBenchmarkRef keyref="xccdf">
   <scapExternalLinks>
      <scapUri keyref="oval"/>
   </scapExternalLinks>
</scapBenchmarkRef>
```

SCAP Composer provides an intuitive and concise vocabulary for authors that is transformable to the more complex vocabulary that SCAP scanner software products require. Doing so provides authors with the best of both worlds. They can compose source data streams and collections using a data model consistent with their mental model with assurance that the transformed result will conform to the SP 800-126 data model.

3. SCAP Composer Architecture, Functionality, and Benefits of DITA

SCAP Composer's architecture is a specialization of DITA, an XML-based architecture for authoring, managing, reusing, and transforming technical content [5]. Standardized by the Organization for the Advancement of Structured Information Systems (OASIS), DITA consists of a set of architectural building blocks with standardized processing semantics. The DITA standard specifies guidelines for constructing *element types* from these building blocks. DITA also specifies rules for creating an element type's *document type*, an XML document grammar to aid in authoring an XML document valid with respect to the element type.

The SCAP Composer document type uses RELAX NG [17] as its grammar mechanism because, out of the three document type grammar mechanisms the DITA standard allows, RELAX NG has the easiest-to-use syntax and enables grammars with the strongest constraints. Because the SCAP Composer uses a RELAX NG document type, SCAP Composer requires an XML authoring solution that supports RELAX NG, as discussed further in Sec. 4. *But you, as a user of SCAP Composer, do not need to learn RELAX NG.* If (1) your XML authoring software supports RELAX NG and (2) you use the SCAP Composer-provided template for authoring your source data stream collections, an understanding of the SCAP Composer document type documentation in Sec. 5 is enough.

The SCAP Composer element type is a specialization of DITA's built-in *map* element type, so it inherits DITA's built-in processing for maps. A DITA map is a structured collection of references to other resources, which may not necessarily be DITA element types. The SCAP Composer element type references the non-DITA XML resources comprising a source data stream collection. The SCAP Composer DITA-OT plug-ins add additional SCAP-specific processing, needed for producing SP 800-126-conforming output, to the built-in DITA map processing.

Fig. 1 shows how SCAP Composer functions in concert with an XML authoring tool and SCAPVal. An SCAP Composer user creates a source data stream (SDS) collection DITA map with an XML authoring tool. The map contains references to the XML resources bundled in the collection. The authoring tool uses the SCAP Composer document type (defined using RELAX NG) to ensure the map is syntactically valid. The XML authoring tool may provide diagnostic messages in real time, hence the bidirectional arrow between the user and RELAX NG validation process. Once done editing, the user invokes the SCAP Composer plug-ins (via DITA-OT) with the SDS collection map as input.

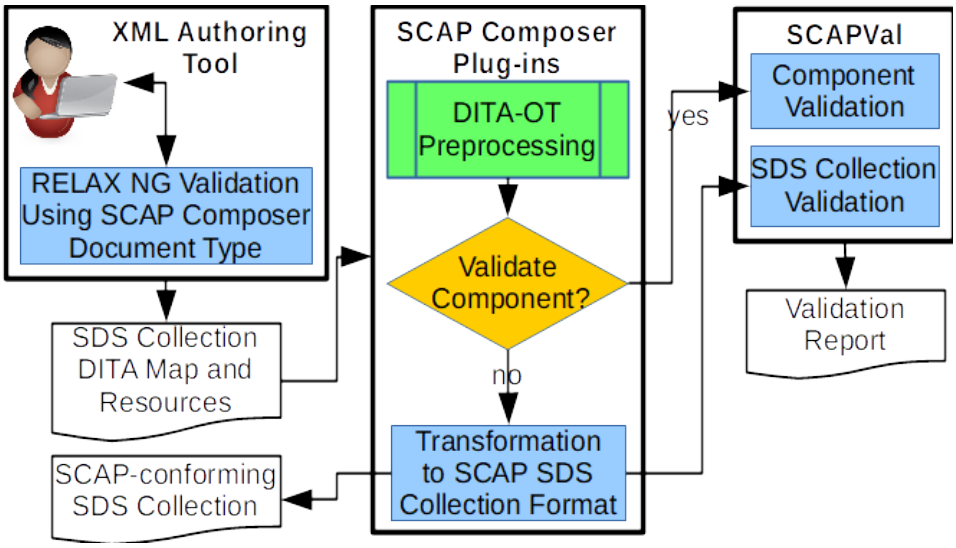

Fig. 1. SCAP Composer functionality in concert with an XML authoring tool and SCAPVal.

After DITA-OT performs initial preprocessing of the input map, the SCAP Composer plug-ins check whether the user provided an optional input argument requesting SCAPVal validation of a single component (i.e., a reference in the map pointing to an XML resource). If so, SCAPVal is invoked to validate only that resource (e.g., an XCCDF checklist or set of OVAL definitions) and produce a report on the component's conformance to its corresponding specification (e.g., the XCCDF [7] or OVAL [6] specifications). Otherwise, the plug-ins first transform the map into a single XML file whose contents is a self-contained source data stream collection formatted in accordance with the SP 800-126 data model [1], and then invoke SCAPVal to report the collection's conformance to SCAP Validation Program test requirements [3].

After reading the preceding paragraphs, you may wonder why no box in Fig. 1 explicitly refers to the SCAP Composer element type. The reason is that the element type is essentially the RELAX NG definitions and transformation code contained in the DITA-OT plug-ins. Thus, you can think of the SCAP Composer element type as the large middle box labeled "SCAP Composer Plug-ins". The interior box labeled "DITA-OT Preprocessing" represents processing semantics the SCAP Composer element type inherits from DITA's built-in map element type. The remaining interior box, diamond, and connecting arrows represent additional processing semantics needed to produce SP 800-126-conforming output.

Another point worth noting is that the XML Authoring Tool, SCAP Composer Plug-ins, and SCAPVal boxes in Fig. 1 are groupings of functionality and not look-and-feel. Invocation of SCAPVal is invisible to a user of the SCAP Composer application because, due to the DITA-OT plug-in mechanism, SCAPVal functionality feels as if it is built into DITA-OT. Similarly, if the SCAP Composer software application is deployed within an XML authoring tool with built-in DITA-OT support, then DITA-OT feels like an extension of the XML authoring tool rather than a separate entity.

To summarize, DITA provides the following implementation, deployment, and usage advantages:

- **Fitness for purpose**. DITA is a perfect fit for the problem of authoring and transforming SCAP content. DITA's built-in reuse mechanisms, exemplified by the SCAP Composer document type's `@keys and @keyref` attributes (discussed in Sec. 5), reduce repetition of long GUIDs (as shown in the example at the end of Sec. 2). DITA map processing semantics support source data stream self-containment and reversibility requirements more elegantly than the SP 800-126 data model.

- **Specialization**. DITA specialization allows SCAP Composer to inherit built-in DITA semantics and DITA-OT functionality, reducing software maintenance effort and enabling interoperability with other DITA-based implementations. Specialization also allows for constraints on inherited elements, enabling the SCAP Composer element type to omit DITA elements and functionality not needed for SCAP source data stream collections. As a result, the SCAP Composer document type is simple and easy to use.

- **Robust ecosystem**. Many authoring and content management tools use DITA-OT, and a wide variety of DITA-OT plug-ins are available. DITA-OT's plug-in architecture plus an active developer community facilitate integration with other tools and addition of new functionalities.

- **Choice of deployment options**. Because DITA is a standard well-supported by open source and commercial developers, SCAP Composer users have multiple deployment options and are not locked into a single software product, operating system, or development environment.

4. Obtaining and Installing SCAP Composer

SCAP Composer requires either:

- XML editing software that bundles DITA-OT 3.1 or newer and supports RELAX NG compact syntax, or

- A standalone DITA-OT (obtainable from https://www.dita-ot.org), a Java Runtime Environment (required for DITA-OT and SCAPVal), and XML editing software that supports RELAX NG compact syntax. Refer to "Installing DITA Open Toolkit" in the DITA-OT documentation for Java version requirements and compatible Java distributions.

SCAP Composer also requires SCAPVal. The *Security content automation protocol (SCAP) version 1.3 validation program test requirements* (NISTIR 7511 Revision 5) [3] includes guidance on obtaining and installing SCAPVal.

The latest release of SCAP Composer is available as a zip file from the NIST website at https://www.nist.gov/services-resources/software/scap-composer. Unzip this file to obtain a directory tree containing:

- A `plugins` directory containing the DITA-OT plug-ins,

- An `examples` directory containing sample SCAP Composer DITA maps,

- An exported scenarios file, `scapComposer.scenarios`, for use with Oxygen XML Editor or XML Author (see Sec. 4.2).

The plug-ins in the `plugins` directory are:

- `gov.nist.scap.doctypes` – the SCAP Composer document type plug-in,
- `gov.nist.scap.datastream` – a plug-in for transforming a source data stream collection DITA map into an SP 800-126-conforming source data stream collection XML file and performing SCAPVal validation of DITA map components or the transformation result,
- `gov.nist.scap.split` – a plug-in for splitting an SP 800-126-conforming source data stream collection XML file into resources comprising its components.

The `gov.nist.scap.doctypes` document type plug-in includes a `template_folders` directory containing the following authoring templates for use with SCAP Composer:

- `SCAP SDS Collection (RNC).ditamap` – a "stub" serving as a starting point for authoring an XML document using the SCAP Composer document type,
- `Map (Single XML Resource).ditamap` – a "stub" for authoring a simple DITA map referencing a non-DITA XML document. Needed as input to the `gov.nist.scap.split` plug-in[2].

4.1. Installation for Use with GNU Emacs and Self-installed DITA-OT

1. Download DITA-OT from the DITA-OT website [4] and extract the zip file to your desired installation location. To easily run the `dita` command from anywhere on your filesystem, you may optionally add the absolute path for the `bin` directory to your `PATH` environment variable.
2. If you do not already have GNU Emacs (some Linux distributions include it by default), obtain and install it as per the instructions on the GNU Emacs website [8]. If you are new to Emacs, start the application, type 'h' while holding down the 'Ctrl' key, release both keys, and then type 't' to invoke a self-paced tutorial.
3. Emacs has many major modes for editing different kinds of text. nXML mode is a major mode for editing XML with RELAX NG support. If you are new to nXML, review the nXML manual, which is included as part of the Emacs "Info" manual. The "Getting More Help" part of the tutorial (see previous step) includes instructions for accessing the Info manual.
4. Configure nXML to edit DITA maps as follows:
    a. Add the following code to your `.emacs` file:
    ```
    (setq auto-mode-alist
    ```

[2] The `gov.nist.scap.datastream` plug-in generates a simple single-reference DITA map as part of its output, which can serve as input when using `gov.nist.scap.split` to split a collection created using SCAP Composer. Thus, you only need to use this template when splitting an SP 800-126 source data stream collection that *was not* created with SCAP Composer.

```
(cons '("\\.ditamap\\'" . nxml-mode)
      auto-mode-alist))
```

b. Save and close your `.emacs` file.

c. Restart Emacs and select "Customize Emacs" from the "Options" menu. Select "Specific Group…" from the cascading submenu. Type 'nxml' at the prompt at the bottom of the window. A list of nXML mode-specific options will appear. Search for "Nxml Slash Auto Complete Flag" and toggle its value to non-nil. Scroll to the top of the window and click the "Apply and Save" button.

5. Copy the contents of the SCAP Composer `plugins` directory to the DITA-OT `plugins` directory.

6. Install the SCAP Composer plug-ins with the following command:
   `dita --install`

4.2. Installation for Use with Oxygen Editor/Author

Oxygen XML Editor is a commercial XML software product for creating XML content and designing schemas and transformations. The Oxygen XML Author product is the subset of Oxygen XML Editor for creating XML content, without the additional capabilities. Oxygen XML Editor and XML Author both include the DITA-OT. The following instructions apply to both products. These instructions, as well as the instructions in Sec. 5.2 and Sec. 6.2, are based on version 21 of Oxygen Editor/Author, the current version as of January 2020. Older versions and future versions may require different configuration, behave differently, or provide a different look-and-feel.

Install SCAP Composer as follows:

1. Copy the contents of the SCAP Composer `plugins` directory to `OXYGEN_INSTALL_DIR/frameworks/dita/DITA-OT-DIR/plugins`, where `OXYGEN_INSTALL_DIR` is the location on your hard drive where Oxygen is installed[3] and `DITA-OT-DIR` is the location of Oxygen's built-in DITA-OT.

2. Start Oxygen. Select and run the DITA-OT Integrator transformation scenario, following the instructions for "Installing a DITA-OT Plugin" in the Oxygen XML Editor User Guide [18].

3. Import the `scapComposer.scenarios` transformation scenarios file, using the "Import Transformation Scenarios" action in the "Options" menu.

4. From the "Window" menu, choose "Show View" and then "Transformation Scenarios" from the cascading menu. Type "scap" in the Transformation Scenarios search bar to narrow the scenario choices to the imported SCAP Composer scenarios, as in Fig. 2.

[3] This requires write access to the Oxygen installation location. If you lack write access and cannot obtain it from your system administrator, an alternative is to install a standalone DITA-OT in a location where you have write access and install the SCAP Composer plug-ins there. See "Using an External DITA Open Toolkit in Oxygen XML Editor" in the Oxygen XML Editor User Guide for additional guidance.

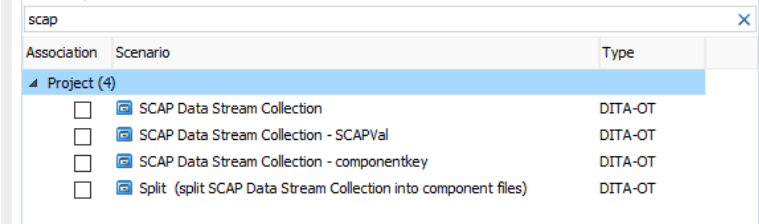

Fig. 2. Imported transformation scenarios.

5. Two of the scenarios – "SCAP Data Stream Collection – SCAPVal" and "SCAP Data Stream Collection – componentkey" – use SCAPVal. Therefore, you must edit them so that the "sds.scapval" parameter value is the relative URI of your SCAPVal jar file. To edit each scenario, right-click on it and choose "Edit" from the context menu. An "Edit DITA Scenario" dialog will appear listing all the parameters. Select "sds.scapval", click the "Edit" button, and click on the folder ("Browse") icon. Navigate to and select the jar file.

5. SCAP Composer Document Type

This section describes the SCAP Composer document type in detail and then provides instructions for editing a source data stream collection. Three tables describe the document type. Step-by-step editing instructions provide guidance for both the GNU Emacs and Oxygen deployment scenarios. Both sets of editing instructions use the same example.

Table 1 lists the content model, attributes, and SP 800-126 source data stream collection data model equivalent for each SCAP Composer element. The content model is an ordered sequence of allowable child elements. The leftmost column lists each element in the document type. The middle column lists the content models and attributes. The rightmost column lists the element from the SP 800-126 source data stream collection data model to which the element in the leftmost column corresponds.

Table 1 uses the following shorthand symbols:

- "?" – means "optional",
- "+" – means "one or more",
- "*" – means zero or more,
- "@" – prefaces an attribute,
- "sds:" – indicates a namespace prefix assumed to map to the SCAP source data stream collection schema namespace provided in SP 800-126.

For example, `scapDataStreamCollection` – the SCAP Composer document root element – contains an optional `title`, followed by one or more `scapComponent` elements, followed by one or more `scapDataStream` elements. `scapDataStreamCollection` has three required attributes: `@reverseDNS`, `@scapName`, and `@schematronVersion`. `scapDataStreamCollection` corresponds to `sds:data-stream-collection` in SP 800-126.

Table 2 and Table 3 define each SCAP Composer document type element and attribute, respectively.

Table 1. SCAP Composer document type elements: content models, attributes, and SP 800-126 equivalents.

Element	Content Model and Attributes	SP 800-126 Equivalent
scapDataStreamCollection	title? scapComponent+ scapDataStream+ @reverseDNS @scapName @schematronVersion	sds:data-stream-collection
title	No element content	None
scapComponent	No element content @keys @href @scope?	sds:component
scapDataStream	scapDictionaries? scapChecklists? scapChecks @scapName @scapVersion @useCase	sds:data-stream
scapDictionaries	scapCpeListRef+	sds:dictionaries
scapChecklists	scapBenchmarkRef+ scapTailoringRef+	sds:checklists
scapChecks	OvalRef+ OcilRef*	sds:checks
scapCpeListRef scapBenchmarkRef scapTailoringRef scapOvalRef scapOcilRef	scapExternalLinks? @keyref	sds:component-ref
scapExternalLinks	scapUri+	cat:catalog
scapUri	No element content @keyref @localUri?	cat:uri

Table 2. Element definitions.

Element	Definition
`scapDataStream-collection`	The top-level element for a data stream collection. It contains all components and data streams.
`title`	The data stream collection's title. SCAP Composer processing ignores this element. It exists only for display of the DITA map in a DITA authoring or content management system.
`scapComponent`	A source data stream collection component whose key is the `@keys` attribute value, and whose XML document location is the value of the `@href` attribute.
`scapDataStream`	A source data stream, represented as a structured collection of component references.
`scapDictionaries`	Contains one or more references to Common Platform Enumeration (CPE) [19] dictionary components.
`scapChecklists`	Contains one or more references to XCCDF benchmark or tailoring components.
`scapChecks`	Contains one or more references to OVAL or Open Checklist Interactive Language (OCIL) [20] components.

`scapCpeListRef`
`scapBenchmarkRef`
`scapTailoringRef`
`scapOvalRef`
`scapOcilRef`
	Each of these elements references a component (CPE dictionary, XCCDF benchmark, XCCDF tailoring component, OVAL definitions, OCIL questionnaire). The `@keyref` attribute value specifies the component's key.

`scapExternalLinks`	Contains `scapUri` mappings to each component internally referenced within the XML document pointed to by the containing component reference element.
`scapUri`	Resolves a reference from within an XML document to another XML document, enabling both documents to be bundled within a self-contained SCAP source data stream. The DITA-OT transformation plug-in uses the `@href` attribute value of the `scapComponent` whose `@keys` attribute value matches `scapUri`'s `@keyref` attribute value to construct an XML Catalogs [16] mapping to the corresponding location within the context of the data stream generated by the plug-in.

Table 3. Attribute definitions.

Attribute	Definition
`@reverseDNS`	The *namespace* portion of all GUIDs in the collection[4].
`@scapName`	Provides the *name* portion of all GUIDs in the collection[4].
`@schematronVersion`	Specifies which version of the SCAP Requirements Schematron [21] schema SCAPVal should use for checking conformance of a transformation result (discussed in Sec. 6) to SP 800-126 requirements.
`@keys`	Defined in the DITA standard to provide a list of key names but SCAP Composer further constrains it to represent a single key name as shorthand for an XML resource. `@keyref` uses this name in place of an XML resource location (provided by `@href`).
`@href`	The location of the XML resource a component encapsulates.
`@scope`	Optional attribute specifying whether `@href` points to a local resource (the default) or an external resource on the Internet.
`@scapVersion`	The version of the SCAP standard to which the data stream content should conform.
`@useCase`	A data stream's use case. One of `CONFIGURATION`, `VULNERABILITY`, `INVENTORY`, or `OTHER`.
`@keyref`	References a component using the short name corresponding to the `scapComponent` element's `@keys` value.
`@localUri`	Optional attribute overriding the URI obtained when SCAP Composer resolves a `@keyref`. This may be needed when an XML resource (e.g., an XCCDF benchmark document) contains references to another resource (e.g., an OVAL document) in a different local directory or Internet location.

Sec. 5.1 and Sec. 5.2 provide step-by-step instructions illustrating use of the SCAP Composer document type to author a source data stream collection DITA map referencing the following SCAP XML resources:

- A CPE dictionary,
- An XCCDF benchmark document,
- An OVAL definitions document containing checks needed for testing compliance with rules in the XCCDF benchmark document,

[4] As discussed in the "Globally Unique Identifiers" subsection of SP 800-126 [1].

- An OVAL definitions document containing checks needed to identify platforms listed in the CPE dictionary.

To follow these steps as a hands-on tutorial, first copy the `nist-example` subdirectory from the SCAP Composer `examples` directory to the directory where your DITA map will reside. The source data stream collection DITA map will reference XML resources from the copied directory except for the reference to the OVAL definitions for checking XCCDF rule compliance, which will be to an external location on the Internet at `https://raw.githubusercontent.com/usnistgov/sctools/master/dita/examples/nist-example/checklist-content/oval.xml`. This will show how you can use the SCAP Composer document type to create a source data stream collection with a mix of local and external XML resources.

5.1. Editing Using GNU Emacs nXML Major Mode

1. Start Emacs and open the directory where your SCAP Composer DITA map will reside. Create a new file `mycollection.ditamap`. Insert the contents of the file `SCAP SDS Collection (RNC).ditamap`, which is in `gov.nist.scap.doctypes/template_folders/Maps/Relax NG (Compact Syntax)`. `mycollection.ditamap` now contains an incomplete (and therefore invalid) "stub" XML document. Errors are highlighted in red. Choose "First Error" from the "XML" menu. Fig. 3 shows that the first error in the document is a missing `@href` attribute.

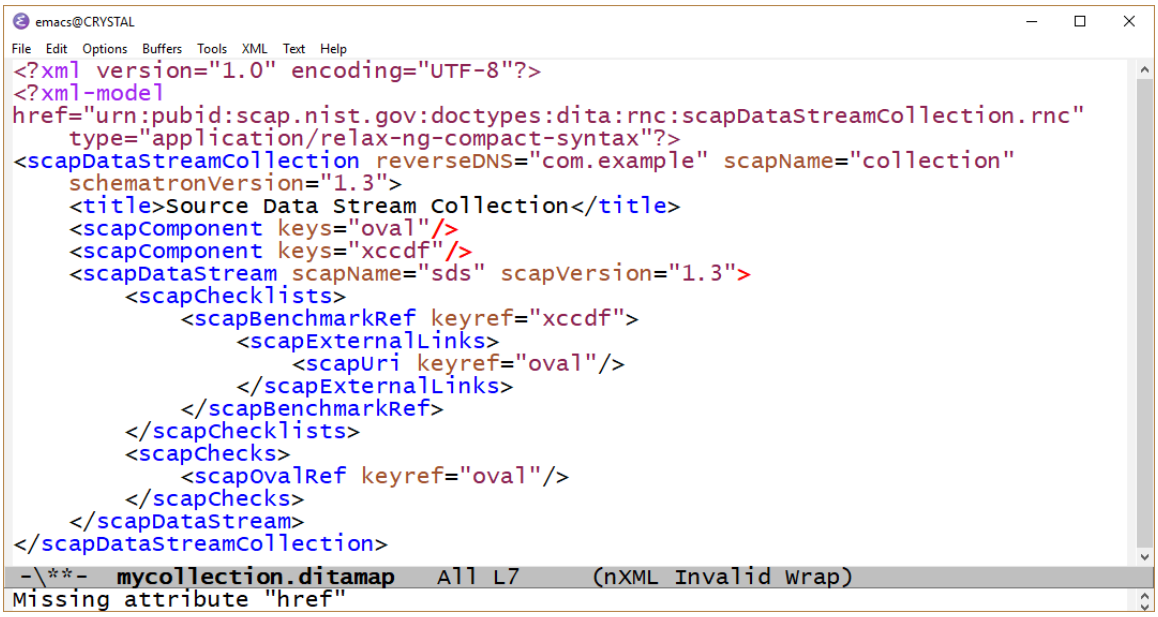

Fig. 3. Missing `@href` nXML error message.

2. Choose "Next Error" from the "XML" menu to display the next error message. Repeat again to display the third error message. You now know that you need make the following fixes:

    a. Add missing `@href` values to both `scapComponent` elements,

    b. Add a `@useCase` value to `scapDataStream`.

Add the missing values. In addition, since the `scapComponent` element with `@keys="oval"` references an external resource, give its `@scope` attribute the value 'external'. `mycollection.xml` is now valid, as shown in Fig. 4.

mycollection.ditamap

File Edit Options Buffers Tools XML Text Help

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model
href="urn:pubid:scap.nist.gov:doctypes:dita:rnc:scapDataStreamCollection.rnc"
    type="application/relax-ng-compact-syntax"?>
<scapDataStreamCollection reverseDNS="com.example" scapName="collection"
    schematronVersion="1.3">
    <title>Source Data Stream Collection</title>
    <scapComponent keys="oval" scope="external"
href="https://raw.githubusercontent.com/usnistgov/sctools/master/dita/examples/n
ist-example/checklist-content/oval.xml"/>
    <scapComponent keys="xccdf"
href="nist-example/checklist-content/xccdf.xml"/>
    <scapDataStream scapName="sds" scapVersion="1.3" useCase="CONFIGURATION">
        <scapChecklists>
            <scapBenchmarkRef keyref="xccdf">
                <scapExternalLinks>
                    <scapUri keyref="oval"/>
                </scapExternalLinks>
            </scapBenchmarkRef>
        </scapChecklists>
        <scapChecks>
            <scapOvalRef keyref="oval"/>
        </scapChecks>
    </scapDataStream>
</scapDataStreamCollection>
```

U\--- **mycollection.ditamap** All L1 (nXML Valid Wrap)

Fig. 4. Valid SCAP Composer document.

3. Add additional `scapComponent` elements for the CPE dictionary and its associated OVAL definitions document. Give the CPE dictionary component's `@keys` attribute a value of `"cpedict"` and its OVAL definition component's `@keys` attribute a value of `"cpe-oval"`. The DITA map now has four `scapComponent` elements, as shown in Fig. 5.

```
    <scapComponent keys="oval" scope="external"
href="https://raw.githubusercontent.com/usnistgov/sctools/master/dita/examples/n
ist-example/checklist-content/oval.xml"/>
    <scapComponent keys="xccdf"
href="nist-example/checklist-content/xccdf.xml"/>
    <scapComponent keys="cpedict" href="nist-example/cpe-dictionary.xml"/>
    <scapComponent keys="cpe-oval" href="nist-example/cpe-oval.xml"/>
```

Fig. 5. Additional `scapComponent` elements added.

4. Edit the `scapDataStream` element to complete the source data stream collection DITA map as follows:

    a. Add an `scapDictionaries` element, with a nested `scapCpeListRef`, before `scapChecklists`. Because `nist-example/cpe-dictionary.xml` references an OVAL definitions document with relative URI of `cpe-oval.xml`, `scapCpeListRef` needs nested `scapExternalLinks` markup (like that of `scapBenchmarkRef`) to translate this relative URI to reference the `scapComponent` using `@keyref="cpe-oval"`.

b. Add a second `scapOvalRef` child element to `scapChecks` referencing the `scapComponent` using `@keyref="cpe-oval"`.

The `scapDataStream` element now appears as shown in Fig. 6.

```
<scapDataStream scapName="sds" scapVersion="1.3" useCase="CONFIGURATION">
    <scapDictionaries>
        <scapCpeListRef keyref="cpedict">
            <scapExternalLinks>
                <scapUri keyref="cpe-oval"/>
            </scapExternalLinks>
        </scapCpeListRef>
    </scapDictionaries>
    <scapChecklists>
        <scapBenchmarkRef keyref="xccdf">
            <scapExternalLinks>
                <scapUri keyref="oval"/>
            </scapExternalLinks>
        </scapBenchmarkRef>
    </scapChecklists>
    <scapChecks>
      <scapOvalRef keyref="oval"/>
      <scapOvalRef keyref="cpe-oval"/>
    </scapChecks>
</scapDataStream>
```

Fig. 6. Completed `scapDataStream` element.

5. Save `mycollection.ditamap` for later use.

5.2. Editing Using Oxygen XML Editor/Author DITA Maps Manager

1. Start Oxygen. Use the "New" action in the "File" menu. Type "scap" into the text field at the top of the dialog box to narrow the choices. Choose the "SCAP SDS Collection (RNC)" file template. Oxygen will display a dialog box asking where you want the DITA map file to be opened. Choose DITA Maps Manager, which displays a map as a table of contents[5]. Save the file in DITA Maps Manager by clicking on the disk icon (or using the Control-S keyboard shortcut), as shown below on the left. A "Save as" dialog box will appear. Save the file as `mycollection-oxygen.ditamap` to the directory where your SCAP Composer DITA map will reside. Choose "Expand All" from the DITA Maps menu to view all elements. `mycollection-oxygen.ditamap` now contains an incomplete (and therefore invalid) "stub" XML document. Errors are highlighted in red with messages listed below the DITA Maps Manager view, as shown in Fig. 7.

[5] DITA Maps Manager is one of several ways to view or edit a DITA map in Oxygen. There is also a Text mode (similar to Emacs/nXML) and an Author mode with a word processor look-and-feel.

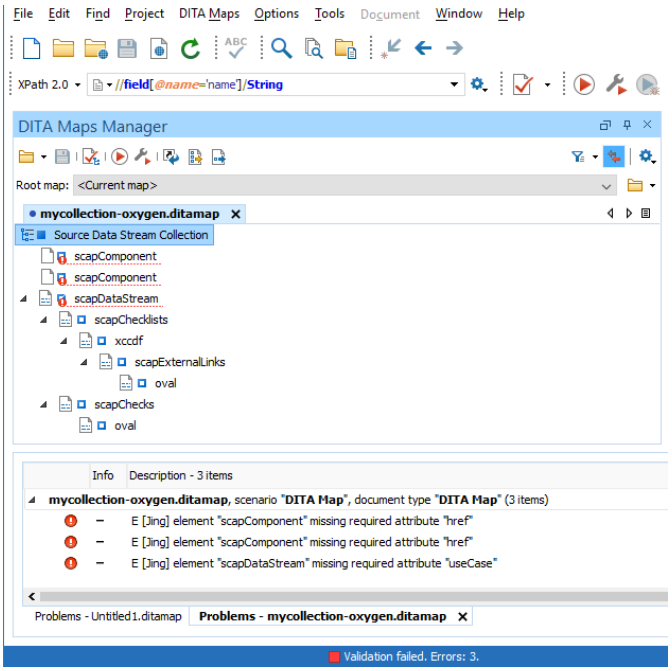

Fig. 7. DITA Maps Manager view with errors highlighted.

2. To fix these errors:

   a. Add missing `@href` values to both Component elements,

   b. Add a `@useCase` value to `scapDataStream`.

   Choose "Edit Attributes…" from the "DITA Maps" menu to add the missing values. In addition, since the `scapComponent` element whose `@keys` value is `'oval'` references an external resource, give its `@scope` attribute the value `'external'`. `mycollection-oxygen.xml` is now valid, as shown in Fig. 8.

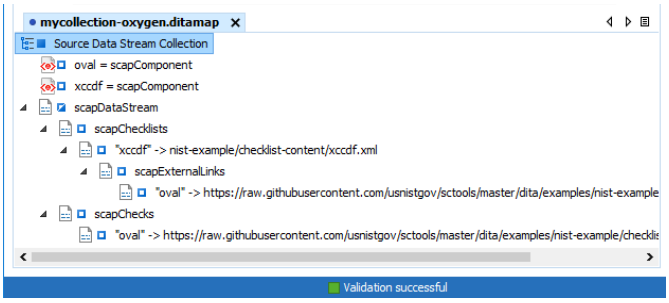

Fig. 8. `mycollection-oxygen.ditamap` now valid.

3. Add a CPE dictionary component by right-clicking on the "xccdf = scapComponent" `scapComponent`. Choose "Insert After" from the context menu that appears, and then choose "scapComponent…" from the cascading submenu as shown in Fig. 9.

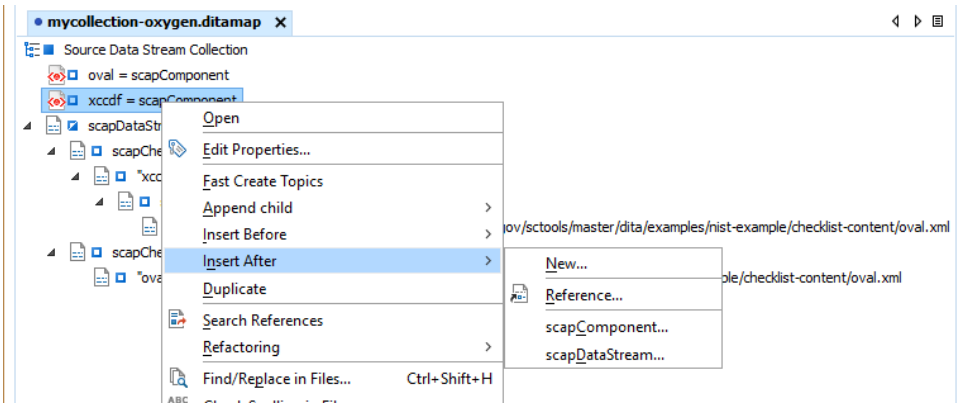

Fig. 9. Inserting an `scapComponent` element

4. An “Insert scapComponent” dialog box will appear. Type “cpedict” in the “Define Keys:” text box, as shown in Fig. 10, and click the “Choose target for defined key(s)” hyperlink.

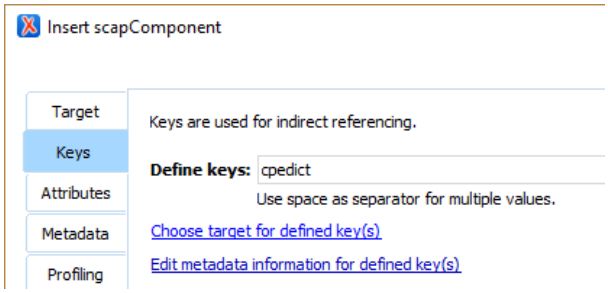

Fig. 10. “Insert scapComponent” dialog box: defining a key.

5. Choose `cpe-dictionary.xml` in the `nist-example` directory as the target, as shown in Fig. 11.

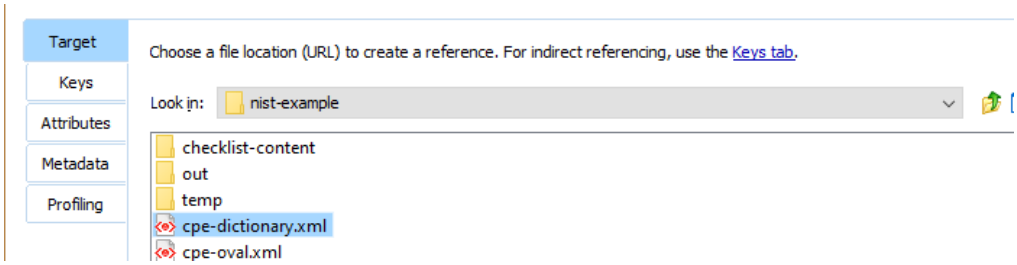

Fig. 11. “Insert scapComponent” dialog box: choosing a target.

6. Add an OVAL component for the CPE dictionary by repeating steps 3, 4, and 5. Set the `@keys` value to “cpe-oval”, and the target to `nist-example` /`cpe-oval.xml`. The DITA map now has four `scapComponent` elements, as shown in Fig. 12.

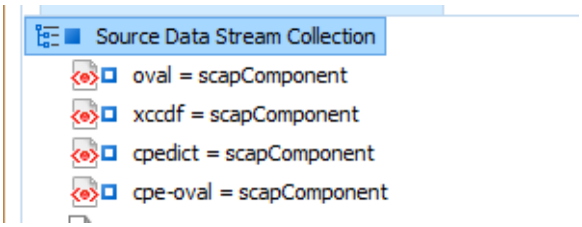

Fig. 12. Four `scapComponent` elements.

7. Insert an `scapDictionaries` element before `scapChecklists` by right-clicking "scapChecklists", choosing "Insert Before", and then choosing "scapDictionaries…" from the cascading menu as shown in Fig. 13.

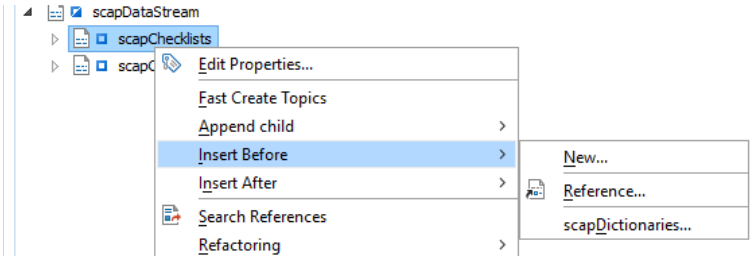

Fig. 13. Inserting an `scapDictionaries` element.

8. Click the "Insert and close" button in the "Insert scapDictionaries" dialog box that appears. An error message will notify you the newly created `scapDictionaries` element is missing a nested `scapCpeListRef`, as shown in Fig. 14.

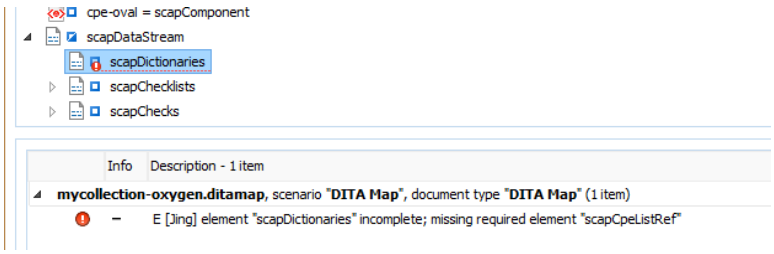

Fig. 14. Error message: Missing required element "scapCpeListRef".

9. Insert the missing `scapCpeListRef` child element by right-clicking "scapDictionaries", choosing "Append child", and then choosing "scapCpeListRef…" from the cascading menu as shown in Fig. 15.

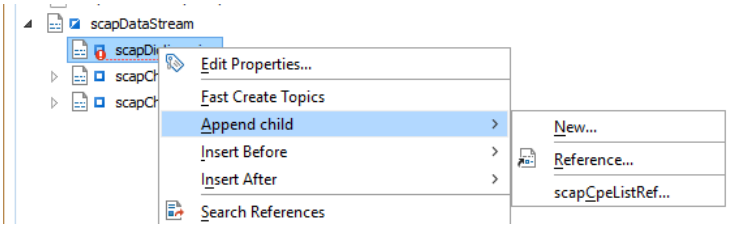

Fig. 15. Appending an `scapCpeListRef` child element.

10. An “Insert scapCpeListRef” dialog box will appear. Set `@keyref` to `“cpedict”` as shown in Fig. 16. Click the “Insert and Close” button.

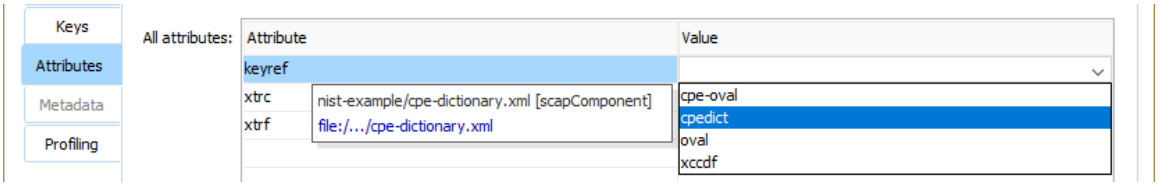

Fig. 16. Setting the `scapCpeListRef` element’s `@keyref` attribute.

11. Using the “Append child” menu action, nest an `scapExternalLinks` element inside `scapCpeListRef`, and an `scapUri` element inside `scapExternalLinks`. An “Insert scapUri” dialog box will appear. Set `@keyref` to `“cpe-oval”.`

12. Using the “Append child” menu action, next a second `scapOvalRef` element inside `scapChecks` as shown in Fig. 17.

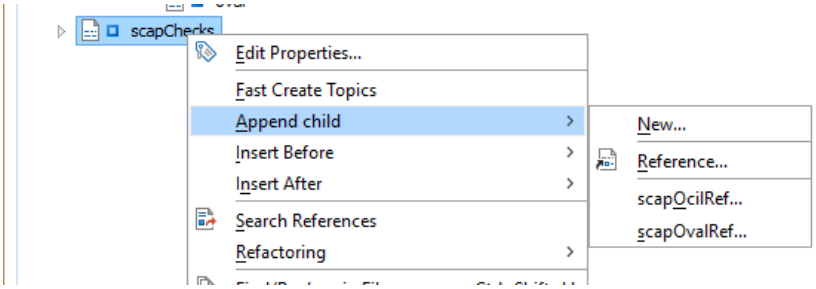

Fig. 17. Nesting a second `scapOvalRef` element.

13. Set the newly created `scapOvalRef` element’s `@keyref` attribute to `“cpe-oval”.` Save `mycollection-oxygen.ditamap` for later use.

6. Using the Transformation Plug-ins

The SCAP Composer transformation plug-ins, `gov.nist.scap.datastream` and `gov.nist.scap.split`, accept a DITA map as input and produce a zip file as output. Both plug-ins may be invoked either using a self-installed DITA-OT `dita` command or from a third-party DITA-OT-enabled software product. The plug-ins are invoked from within Oxygen XML Editor/Author by applying imported transformation scenarios.

`gov.nist.scap.datastream` provides the following functionalities:

- Transforms a DITA map conforming to the SCAP Composer document type into an SCAP source data stream collection,
- Uses SCAPVal to validate the generated SCAP source data stream collection,
- Uses SCAPVal to validate an XML resource referenced by an `scapComponent` in an SCAP Composer DITA map.

`gov.nist.scap.split` splits an SCAP source data stream collection into a set of XML resources comprising its components. Because all DITA-OT transformations require DITA

as input, input is a map with a single reference to the SCAP source data stream collection to be split. This map may be created using the `Map (Single XML Resource).ditamap` authoring template mentioned in Sec. 4, replacing "resource.xml" in the template with the SCAP source data stream collection URI.

As an SCAP Composer user, you do not need to be cognizant of which DITA-OT plug-in you are using. It is simpler instead to keep in mind the four basic transformation functionalities: (1) SP 800-126 source data stream collection generation either with or (2) without SCAPVal validation, (3) component validation, and (4) splitting.

Table 4 lists the inputs and output zip file contents for each of the four functionalities. Outputs in italics are input artifacts that are not generated by SCAP Composer. These are included in the zip file because they might possibly be helpful to you but can be ignored if you do not find them helpful.

Table 4. SCAP Composer plug-in inputs and outputs

Functionality	Input DITA Map	Output Zip File Contents
Generate SCAP without SCAPVal validation	SCAP Composer document	*XML resources referenced by SCAP Composer document*
		Map referencing SCAP source data stream collection file
		SCAP source data stream collection file
Generate SCAP with SCAPVal validation	SCAP Composer document	*XML resources referenced by SCAP Composer document*
		Map referencing SCAP source data stream collection file
		SCAP source data stream collection file
		SCAPVal validation report in HTML and XML
SCAPVal validation of component XML resource	SCAP Composer document	*XML resources referenced by SCAP Composer document*
		XML resource validated
		SCAPVal validation report in HTML and XML
Split SCAP collection into component resources	Map referencing an SCAP source data stream collection	*SCAP source data stream collection file referenced by the map*
		XML resource files

6.1. Invoking the Transformation Plug-ins using the `dita` Command

The `dita` command may be entered from a command prompt or from within GNU Emacs as a shell command. Basic `dita` command syntax is

`dita -i` *`input-file`* `-f` *`output-format`* [*`options`*]

where *input-file* is a DITA map, and *output-format* is either `sds` (for generating an SCAP source data stream collection file) or `split` (for splitting a collection file into separate XML resource files).

[*options*] include additional optional command line arguments and parameters. One such command line argument is the `-v` flag, which prints additional information to the console that may be useful in diagnosing errors or unexpected output results.

The following two parameters apply for the `sds` output format:

- `--sds.scapval=`*jar-file*
  Specifies the SCAPVal jar file location.
- `--sds.componentkey=`*key*
  Specifies a component to validate using SCAPVal, where *key* identifies the component. Requires `--sds.scapval.`

The following subsections provide examples of using the `dita` command to achieve the four functionalities from Table 4. Input for the first three functionalities is the `mycollection.ditamap` document, authored in Sec. 5.1. Input for the example in Sec. 6.1.4 for the fourth functionality is `composed-mycollection.ditamap`, an output from the first two examples.

6.1.1. Generate SCAP Without SCAPVal Validation

```
dita -i mycollection.ditamap -f sds
```

Output is a zip file `mycollection.zip` whose contents include the following outputs:

- `mycollection.xml` – a source data stream collection conforming to the SP 800-126 data model,
- `composed-mycollection.ditamap` – a DITA map referencing `mycollection.xml`, for use as an input argument when splitting a collection into its resources (see Sec. 6.1.4).

6.1.2. Generate SCAP With SCAPVal Validation

```
dita -i mycollection.ditamap -f sds --sds.scapval=scapval.jar
```

Output is a zip file `mycollection.zip` with the same contents as in Sec. 6.1.1 plus the following files:

- `validation-report.html` – a browser-viewable HTML report file,
- `validation-result.xml` – a validation result file.

6.1.3. SCAPVal Validation of Component OVAL resource

```
dita -i mycollection.ditamap -f sds --sds.scapval=scapval.jar --
sds.componentkey=oval
```

Output is a zip file `mycollection.zip` whose contents includes:

- `oval-validation-report.html` – a browser-viewable HTML component report file for `oval.xml`,
- `oval-validation-result.xml` – a validation result file.

6.1.4. Split SCAP Collection into Component Resources

```
dita -i composed-mycollection.ditamap -f split
```

Output is a zip file `composed-mycollection.zip` whose contents includes:

- `cpedict.xml` – CPE dictionary resource,
- `cpe-oval.xml` – OVAL definitions document containing checks needed for the CPE dictionary resource,
- `oval.xml` – OVAL definitions document containing checks needed for the XCCDF resource
- `xccdf.xml` – XCCDF resource.

6.2. Invoking the Transformation Plug-ins from Oxygen Editor/Author

Invoking the transformation plug-ins from Oxygen Editor/Author is simply a matter of selecting and applying the appropriate imported transformation scenario to the currently edited document. Each transformation scenario corresponds to a functionality from Table 4. Transformation zip file output is the same as with the `dita` command as discussed in Sec. 6.1.

If you apply the "SCAP Data Stream Collection – componentkey" scenario, a dialog prompts for a key name as shown in Fig. 18.

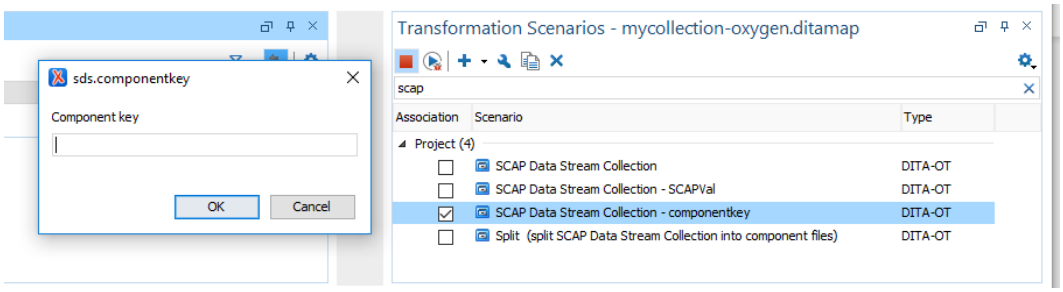

Fig. 18. SCAP Data Stream Collection – "componentkey" scenario.

A "Transformation in progress" message then appears in the status bar at the bottom of the Oxygen window while the transformation scenario is running. If the transformation is successful, the message changes to "Transformation successful" and a console output icon appears to the right. Clicking on the icon causes console output to display, as shown in Fig. 19.

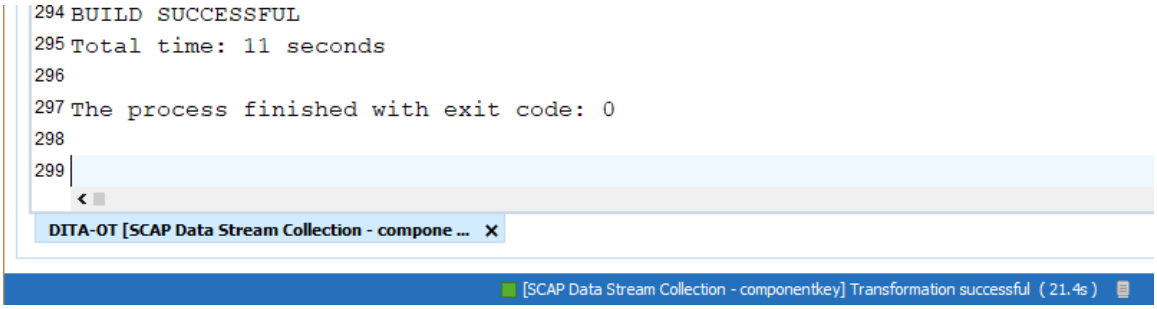

Fig. 19. Console output with "Transformation successful" message in status bar.

7. Concluding Remarks

This guide documented SCAP Composer, a software application for creating SCAP source data stream collections. Installation and usage instructions targeted two deployment scenarios: one requiring freely available open source tools, and the other requiring a commercial software product meeting SCAP Composer's requirements. These two scenarios are intended to cover the most likely ways in which SCAP Composer will be obtained, installed, and used.

SCAP Composer's release coincides with a new effort underway to improve upon SCAP version 1.3. Within the proposed scope of this "SCAP 2.0" initiative [22] is improved "SCAP content creation, acquisition, and reuse." SCAP Composer can contribute to this goal by making it easier to combine content into new source data streams. Also, SCAP Composer's DITA-based implementation approach could be applied to other content authoring problems such as easing the development of XCCDF or OVAL content. Reference [23] discusses preliminary work done to determine the feasibility of specializing DITA to represent XCCDF configuration compliance rules and groupings of rules, but more implementation is needed to scale this initial effort to a real-world XCCDF checklist or OVAL definitions document. If such implementation were successful, the DITA-OT plug-in architecture would make it easy to combine the existing SCAP Composer implementation with new plug-ins to support XCCDF or OVAL authoring.

SCAP Composer is a research prototype whose purpose is to spur development of third-party software products that advance the adoption of SCAP. Would-be software developers are encouraged to obtain and experiment with the source code, available at https://github.com/usnistgov/sctools. For questions about the code, or about SCAP Composer in general, please email the point of contact listed on NIST's SCAP Composer information page (https://www.nist.gov/services-resources/software/scap-composer). Feedback, bug reports, and suggestions for improvement are all welcome. The SCAP Composer information page also contains links to publications and presentations discussing implementation details not covered in this document.

Acknowledgments

I am grateful to my NIST colleagues Stephen Banghart, Yan Lu, and Frank Riddick for feedback on earlier drafts of this guide, and to Neeraj Shah, CheeYee Tang, and Tim Zimmerman for assistance with deploying SCAP Composer-generated content in NIST's

Cybersecurity for Manufacturing Systems Testbed – an experience that inspired Sec. 2's HMI scenario.

I also wish to thank Gabe Alford, Bernd Grobauer, Bill Munyan, David Ries, Charles Schmidt, and other SCAP 2.0 Content Authoring Working Group members for their insights on distinct operational roles of various categories of SCAP users. These insights were helpful to me when writing the introductory paragraphs in Sec. 2.

References

[1] Waltermire D, Quinn S, Booth H, Scarfone K, Prisaca D (2018) *The technical specification for the security content automation protocol (SCAP) version 1.3* (National Institute of Standards and Technology, Gaithersburg, MD), NIST SP 800-126r3. https://doi.org/10.6028/NIST.SP.800-126r3

[2] Extensible Markup Language (XML) 1.0 (Fifth Edition) (2008) , W3C Recommendation. Available at http://www.w3.org/TR/xml/

[3] Cook M, Quinn S, Waltermire D, Prisaca D (2018) *Security content automation protocol (SCAP) version 1.3 validation program test requirements* (National Institute of Standards and Technology, Gaithersburg, MD), NIST IR 7511r5. https://doi.org/10.6028/NIST.IR.7511r5

[4] DITA Open Toolkit Available at https://www.dita-ot.org/

[5] DITA Version 1.3 Specification (2018) (Organization for the Advancement of Structured Information Standards), OASIS Standard. Available at http://docs.oasis-open.org/dita/dita/v1.3/dita-v1.3-part0-overview.html

[6] Home / Oval Repository Available at https://oval.cisecurity.org/

[7] Waltermire D, Schmidt C, Scarfone K, Ziring N (2011) *Specification for the Extensible Configuration Checklist Description Format (XCCDF) Version 1.2*, NIST Interagency Report 7275 Revision 4. Available at http://csrc.nist.gov/publications/PubsNISTIRs.html

[8] GNU Emacs - GNU Project Available at https://www.gnu.org/software/emacs/

[9] Oxygen XML Editor Available at https://www.oxygenxml.com/

[10] CIS Security Benchmarks Available at https://benchmarks.cisecurity.org/

[11] Security Technical Implementation Guides (STIGs) – DoD Cyber Exchange Available at https://public.cyber.mil/stigs/

[12] ComplianceAsCode/content: Security compliance content in SCAP, Bash, Ansible, and other formats Available at https://github.com/ComplianceAsCode/content

[13] XML Schema Part 0: Primer Second Edition (2004) , W3C Recommendation. Available at https://www.w3.org/TR/xmlschema-0/

[14] Lubell J (2018) A New SCAP Information and Data Model for Content Authors. *Critical Infrastructure Protection XII*, IFIP Advances in Information and Communication Technology., eds Staggs J, Shenoi S (Springer International Publishing), Vol. 542, pp 127–146.

[15] Lubell J (2019) SCAP Composer: A DITA Open Toolkit Plug-in for Packaging Security Content. *Proceedings of Balisage: The Markup Conference*, Balisage Series on Markup Technologies. (Washington, DC). https://doi.org/10.4242/BalisageVol23.Lubell01

[16] XML Catalogs (2005) , OASIS Standard V1.1. Available at https://www.oasis-open.org/committees/download.php/14809/xml-catalogs.html

[17] Information technology — Document Schema Definition Language (DSDL) — Part 2: Regular-grammar-based validation — RELAX NG (2008) (International Organization for Standardization), ISO/IEC 19757-2.

[18] Oxygen Documentation Available at https://www.oxygenxml.com/documentation.html#user_guide

[19] Cichonski P, Waltermire D, Scarfone K (2011) *Common platform enumeration :: dictionary specification version 2.3* (National Institute of Standards and Technology, Gaithersburg, MD), NIST IR 7697.

[20] Waltermire D, Scarfone K, Casipe M (2011) *Specification for the open checklist interactive language (OCIL) version 2.0* (National Institute of Standards and Technology, Gaithersburg, MD), NIST IR 7692.

[21] Information technology — Document Schema Definition Language (DSDL) — Part 3: Rule-based validation — Schematron (2016) (International Organization for Standardization), ISO/IEC 19757-3.

[22] Waltermire D, Fitzgerald-McKay J (2018) *Transitioning to the Security Content Automation Protocol (SCAP) Version 2* (National Institute of Standards and Technology, Gaithersburg, MD), NIST CSWP 09102018. https://doi.org/10.6028/NIST.CSWP.09102018

[23] Lubell J (2017) Using DITA to Create Security Configuration Checklists: A Case Study. *Proceedings of Balisage: The Markup Conference*, Balisage Series on Markup Technologies. (Washington, DC). https://doi.org/10.4242/BalisageVol19.Lubell01