

iiRDS Plugin for the DITA Open Toolkit

Contents

Chapter 1. Introduction.....	3
Chapter 2. Version history.....	5
Chapter 3. Requirements.....	6
Chapter 4. Install the iiRDS plugin.....	7
Chapter 5. How the plugin works.....	8
Chapter 6. Build and check an iiRDS package.....	9
Chapter 7. Metadata extraction.....	11
Mapping of standard DITA elements and attributes.....	12
Controlled iiRDS taxonomies using subject scheme maps.....	16
Subject scheme class mapping.....	18
Assign metadata using a subject scheme.....	20
iiRDS base subject scheme map and generator.....	23
Known limitations.....	24
Chapter 8. Unique identifiers.....	26
Chapter 9. Plugin parameters.....	28
Chapter 10. Customization.....	30
IRI and metadata handlers.....	31
Customize metadata extraction.....	32
Customize IRI generation.....	32
Customize HTML5 output.....	33
Chapter 11. License and contact information.....	35

Chapter 1. Introduction

The plugin *org.iirds.dita.package* for the DITA Open Toolkit (DITA-OT) generates iiRDS packages from a DITA map or a single topic.

“iiRDS is a standard for the delivery of intelligent information in the scope of user assistance for products. The information is provided with the product for the purpose of assisting the users in setting up, operating, and maintaining the product. Intelligent information is defined as technical documentation content enriched with metadata.

iiRDS consists of:

- A vocabulary for the metadata provided with the content. The vocabulary is specialized for and restricted to the domain of technical documentation or user assistance. iiRDS uses an RDF schema as technical format. RDF stands for Resource Description Framework and is a standard model for data interchange on the Web.
- A package format for the exchange of packages with intelligent information between different systems, for example, web portals or content delivery servers. The package format uses ZIP and has a predefined folder structure for content and metadata in RDF format.

For details about the iiRDS standard, see <https://www.iirds.org>

”

To create iiRDS packages from DITA, this plugin uses a predefined mapping of DITA elements and attributes to iiRDS classes and properties, see [Metadata extraction \(on page 11\)](#).

The plugin has the following basic characteristics:

- Based on DITA 1.3
- Supports DITA content in XML format.
- Provides the new transformation type *iirds*.
- Extends the HTML5 plugin.
- Parameters of the HTML5 transformation can be used to integrate custom HTML formatting.
- Resulting iiRDS packages conform to version 1.3, unrestricted, of the iiRDS standard.
- iiRDS metadata is generated based on the following:

- Existing metadata elements and attributes in DITA
- Data elements that are mapped to an iiRDS taxonomy using a subject scheme map.

The plugin uses extension points provided by the DITA-OT to implement its functionality. Custom parameters and new extension points are provided that can be used to further customize the functionality of this plugin.



Note:

DITA 2.0 content, including lightweight DITA in other formats than XML, are not yet supported. Transformations of such content may still work, but this is not guaranteed.

Chapter 2. Version history

The following table lists the released versions of the plugin and the most important changes in each version.

Version	Release	Changes
1.1.0	2026-08	• New: iiRDS metadata can be assigned with subject scheme maps that map DITA <code><data></code> elements to iiRDS classes and instances. • Generated iiRDS packages now conform to version 1.3, unrestricted, of the iiRDS standard. Previous versions generated iiRDS 1.2 packages. • New: support for the iiRDS handover domain, including the document category. • New: generation of qualifications and supply relations. • New: support for events and features. • Various bug fixes and documentation improvements.
1.0.1	2025-10	Minor bug fixes.
1.0.0	2024-03	Initial release. Generates iiRDS 1.2 packages, unrestricted, from a DITA map or a single topic.

Chapter 3. Requirements

This plugin was developed and tested with DITA-OT 3.7.3 and DITA-OT 4.1.1. To guarantee that this plugin works, the following versions are required:

- DITA-OT 3.7.x or later
- Java (version according to DITA-OT requirements)

The plugin can be used with a default version of the DITA-OT or a customized DITA-OT version with specialized document type definitions and modified transformation scenarios.

Chapter 4. Install the iiRDS plugin

The iiRDS plugin can be installed offline from a ZIP archive or online from the DITA-OT registry or the <https://iirds.org> website.

Plugin installation is described in the documentation of DITA-OT. Follow the installation instructions for your version of DITA-OT to install the plugin.

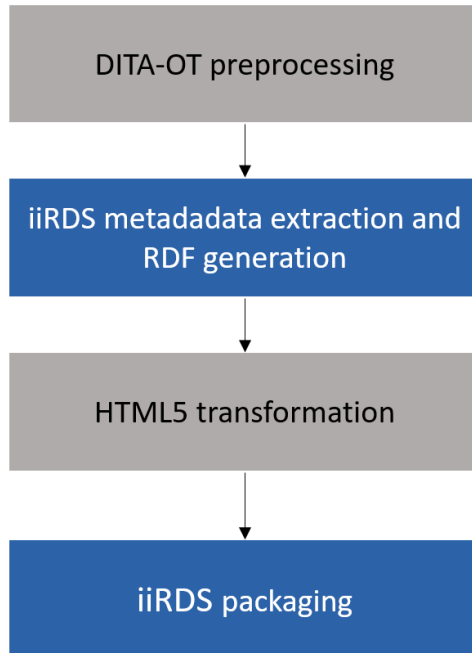
Related information

<https://www.dita-ot.org/4.3/topics/plugins-installing>

Chapter 5. How the plugin works

Once the iiRDS plugin has been installed in the DITA-OT, you can use the plugin to generate iiRDS packages from your DITA map. The processing steps are described in the following figure:

Figure 1. Processing steps of the iiRDS plugin



In detail, this means the following:

- The default preprocessing by the DITA-OT is performed. During the preprocessing, references are resolved, the DITA mechanisms for inheriting metadata from parent nodes are applied and the DITA content is filtered, if required. For details, see <https://www.dita-ot.org/dev/reference/preprocessing>.
- iiRDS metadata is extracted from the preprocessed XML and the RDF file for the iiRDS package is generated. For details, see [Metadata extraction \(on page 11\)](#).
- The default HTML5 transformation is applied to the preprocessed XML. The parameters of this transformation can be used to customize the generated HTML. For details, see [Customize HTML5 output \(on page 33\)](#).
- The HTML content and the RDF file are packed in an iiRDS container file that can be imported in any application that supports iiRDS 1.3.

Chapter 6. Build and check an iiRDS package

You can use the DITA-OT to build an iiRDS package for a DITA map or a single topic. The output directory will contain the generated iiRDS file with the extension `.iirds`.

1. In the `bin` folder of your DITA-OT installation, open a command-line client.
2. In the command-line client, enter the `dita` command to transform your DITA map or topic using the `iirds` transtype.

Example: `dita --input myContent.ditamap --o out -f iirds`

You can apply conditional filtering and use additional parameters as needed.

An iiRDS package with the following file name is created in the output directory:

`<file_name_of_ditamap>.iirds`.

3. Verify that the iiRDS package contains the desired result:
 - a. Use your ZIP tool to extract the content from the iiRDS file. If necessary, change the file extension to `.zip` first.
 - b. In the subfolder `META-INF`, open the file `metadata.rdf` and check if the iiRDS metadata contains the expected values from your DITA content.
 - c. In the subfolder `content`, open the file `index.html` in a browser and check whether the HTML content is as expected.

If the iiRDS package does not contain the expected result, you can try one of the following:

- If content is missing or you have unwanted content, check your conditional processing attributes.
- If the iiRDS metadata is not what you expected, check the metadata mapping topic to find out why. In case you need a different outcome, you need to customize the metadata extraction.
- If your HTML does not look the way you want, try to use the properties of the HTML5 transformation.



Tip:

Integrate the configured transformation into your editor of choice.



Tip:

Use the [iiRDS Validation Tool](#) to check the integrity of the generated iiRDS package.



Tip:

All DITA content contains some form of metadata. If you want to test the plugin, go ahead and run it with your DITA content without making any changes.

Related information

Metadata extraction (*on page 11*)

Customize metadata extraction (*on page 32*)

Customize HTML5 output (*on page 33*)

Chapter 7. Metadata extraction

The iiRDS plugin comes with a number of predefined metadata handlers to extract metadata from DITA and map them to iiRDS metadata. The iiRDS plugin can process DITA maps or individual topics.

In the RDF file with the iiRDS metadata, the plugin creates the following RDF resources:

- An `iiRDS:Package` and an `iiRDS:Document` for the root DITA map, if available. The `iiRDS:DocumentType` is set to `iiRDS:OperatingInstructions`.
- An `iiRDS:Topic` for each topic in a DITA map or for a single DITA topic. For each topic, an additional `iiRDS:InformationObject` is generated.

In addition to the information units, the plugin creates a directory of the structure of the DITA map, where each entry in the `index.html` of the resulting HTML output is represented by an `iiRDS:DirectoryNode`.

More iiRDS metadata is extracted from the following sources:

- Content of selected standard elements and attributes. See [Mapping of standard DITA elements and attributes \(on page 12\)](#) for a detailed description of the mapping from DITA content to iiRDS and [Known limitations \(on page 24\)](#) to understand how the metadata is mapped.

This approach is useful when:

- You already have DITA metadata that you use for other use cases, for example, filtering content for other pipelines.
 - You have already created a custom metadata handler or plan to create one to extract more metadata content from your DITA files.
 - You want to easily create iiRDS packages based on existing content and not worry about having to inject iiRDS-specific metadata terms.
- Taxonomy-based metadata, such as products, components, or qualifications, that is assigned to maps and topics using a subject scheme. For more information, see [Controlled iiRDS taxonomies using subject scheme maps \(on page 16\)](#).

This approach is useful when:

- The metadata values form a taxonomy, that is, a hierarchy of classes and instances, for example, a product structure with components and sub-components.
- You already have iIRDS taxonomies for other use cases or plan to create reusable taxonomies that can be shared across DITA and non-DITA projects.
- Metadata should be maintained centrally in a subject scheme map rather than scattered across DITA attribute declarations.

Both approaches to extract metadata for an iIRDS package can be combined to get the best of both worlds. However, there will not be any deduplication. If the resulting iIRDS metadata entry for a topic or map from both handlers is identical, there will only be one entry in the RDF file. If the metadata entries are different, for example, in spelling or because they have a different IRI, there will be multiple entries following the general rules for merging metadata in RDF.

Mapping of standard DITA elements and attributes

The following table describes how DITA elements and attributes are mapped to iIRDS resources in the RDF file of the iIRDS package. If nothing else is stated, available metadata is mapped to `iirds:Document` and `iirds:Topic` in the RDF file. The mapping also applies to any specializations of the listed elements and attributes, unless a specific mapping is provided for the specialized type.

Table 1. Mapping of DITA to iIRDS metadata

DITA	iIRDS RDF	Comment
<code><ditamap></code>	<code>iirds:Document</code> with document type <code>iirds:OperatingInstructions</code>	All DITA maps are treated as operating instructions because DITA does not provide a document type by default.
<code><topic></code> , <code><task></code> , <code><concept></code> , <code><reference></code> , <code><troubleshooting></code>	<code>iirds:Topic</code> with the topic type set as follows: • <code>iirds:GenericTask</code> for <code><task></code> • <code>iirds:GenericTroubleshooting</code> for <code><troubleshooting></code>	Topic types are derived from the <code>@class</code> attribute of the corresponding topic. Specialized topics are mapped according to the topic type that they specialize, for example, a special-

Table 1. Mapping of DITA to iiRDS metadata (continued)

DITA	iiRDS RDF	Comment
	• <code>iiirds:GenericReference</code> for <code><ref- erence></code> • <code>iiirds:GenericConcept</code> for <code><topic></code>, <code><concept></code>, and all other topic types	ization of <code><task></code> is mapped to <code>iiirds:GenericTask</code>. Any topic type without a specific mapping is mapped to <code>iiirds:GenericConcept</code>.
<code><shortdesc></code>	<code>iiirds:Topic</code> > <code>iiirds:has-abstract</code>	Content of topic-level short description is used as abstract property. If <code><shortdesc></code> is wrapped in <code><abstract></code> and multiple <code><shortdesc></code> elements are present, only the first short description is evaluated.  Note: Short descriptions on map level are not evaluated. Only the <code><shortdesc></code> of the topic itself is used as the <code>iiirds:has-abstract</code> property.
<code><title></code>	<code>iiirds:title</code>	<code><title></code> elements of root map and topic roots are used. Other <code><title></code> elements are not evaluated, for example, section or table titles.

Table 1. Mapping of DITA to iiRDS metadata (continued)

DITA	iiRDS RDF	Comment
<code>@xml:lang</code>	<code>iiRDS:language</code>	Language attribute of root map or topic root is used.
<code><prodname></code> or <code>@product</code>	<code>iiRDS:ProductVariant</code>	All applicable values of <code><prodname></code> elements and <code>@product</code> attributes from the root map and topics are used.
<code>@audience</code> or (<code><audience></code> with <code>@type</code> and/or <code>@experiencelevel</code>)	<code>iiRDS:Role</code> and <code>iiRDS:Skilllevel</code>	<code>@audience</code> attribute of root map or topic root is mapped to <code>iiRDS:Role</code> . For <code><audience></code> , the content of <code>@type</code> is mapped to <code>iiRDS:Role</code> and <code>@experiencelevel</code> is mapped to <code>iiRDS:SkillLevel</code> .
<code><component></code>	<code>iiRDS:Component</code>	<code><component></code> elements of root map and topic root are used.
<code><copyright></code> with <code>@year</code> and <code>@holder</code>	<code>iiRDS:rights</code>	<code><copyright></code> elements of root map and topic root are used.
<code><created></code> with <code>@date</code>	<code>iiRDS:dateOfCreation</code>	<code>@date</code> attribute of <code>created</code> element of root map or topic root is used.
<code><revised></code> with <code>@modified</code>	<code>iiRDS:dateOfLastModification</code> and <code>iiRDS:dateOfStatus</code>	<code>@modified</code> attribute of <code>revised</code> element of root map or topic root is used.
<code><created></code> or <code>revised</code> with <code>@golive</code>	<code>iiRDS:dateOfEffect</code>	<code>@golive</code> attribute of <code>created</code> or <code>revised</code> element of root map or topic root is used.

Table 1. Mapping of DITA to iiRDS metadata (continued)

DITA	iiRDS RDF	Comment
		If both <code><created></code> and <code><revised></code> with <code>@golive</code> are present, then the latest <code><revised></code> value is used.
<code><created></code> or <code><revised></code> with <code>@expiry</code>	<code>iirds:dateOfExpiry</code>	<code>@expiry</code> attribute of <code>created</code> or <code>revised</code> element of root map or topic root is used. If both <code><created></code> and <code><revised></code> with <code>@expiry</code> are present, then the latest <code><revised></code> value is used.
<code><topichead></code>	<code>iirds:DirectoryNode</code>	<code><topichead></code> elements are title-only entries in a navigation map. Therefore, they do not have an <code>iirds:Topic</code> equivalent, but they are included as a directory node. The <code>@navtitle</code> attribute or <code><navtitle></code> element becomes the label of the directory node.
<code><navtitle></code>	<code>iirds:DirectoryNode</code>	<code><navtitle></code> elements from <code><topicref></code> elements, <code><topichead></code> elements, or from a topic are used as labels for the corresponding iiRDS directory nodes.

Related information

Unique identifiers (on page 26)

Subject scheme class mapping (on page 18)

Controlled iiRDS taxonomies using subject scheme maps

In addition to the predefined metadata handler described in [Metadata extraction \(on page 11\)](#), the iiRDS plugin provides a `subjectscheme` metadata handler. This handler lets authors assign metadata based on an iiRDS taxonomy provided via a subject scheme map. This approach provides a generic way to define taxonomies in a central location and assign the metadata to all relevant DITA maps and topics. Taxonomy values are referenced from the subject scheme map using `<data>` elements with a `@keyref` attribute. That way, you can build complex taxonomy trees based on the core classes and instances provided by iiRDS, for example, for products, components, qualifications, or information subjects.

Metadata definitions in the subject scheme map

A DITA subject scheme map (`<subjectScheme>`) defines a hierarchy of `<subjectdef>` elements. Each `<subjectdef>` that is relevant to iiRDS carries a `<resourceid>` element in its `<topicmeta>` with two attributes:

- `@appid` contains the IRI that identifies the iiRDS class or instance.
- `@appname` is set to one of the following:
 - `class-iri` if the `<subjectdef>` represents an iiRDS class, for example, `iirds:Component`
 - `instance-iri` if the `<subjectdef>` represents a concrete instance of a class, for example, a specific component of a product

Each `<subjectdef>` also declares one or more `@keys` values so that it can be referenced from DITA content with a `@keyref` attribute. The following example defines the iiRDS class `Component` and one instance of it, a product part called "Rotor":

```
<subjectdef keys="iirds.Component">
  <topicmeta>
    <navtitle>component</navtitle>
    <resourceid appid="http://iirds.tekom.de/iirds#Component" appname="class-iri"/>
  </topicmeta>
  <subjectdef keys="product.Rotor">
    <topicmeta>
      <navtitle>Rotor</navtitle>
      <resourceid appid="https://example.com/product#Rotor" appname="instance-iri"/>
    </topicmeta>
  </subjectdef>
</subjectdef>
```

```

</subjectdef>

</subjectdef>

```

Class hierarchy resolution

The `subjectscheme` handler maps an instance to an iiRDS property based on a fixed set of top-level iiRDS classes that are built into the plugin. See [Subject scheme class mapping \(on page 18\)](#) for the complete list of these classes and the iiRDS properties they are mapped to.

To determine the applicable property for an instance, the handler follows the parent-child structure declared in the subject scheme map, from the instance upwards, until it reaches one of these top-level classes. This means a project is not limited to using the top-level classes directly: any class nested underneath one of them, at any depth, is resolved to the same property as its top-level ancestor.

This mirrors how the iiRDS standard itself allows implementers to extend the taxonomy: new classes and instances can be added anywhere in the hierarchy, as long as they are declared as subclasses, equivalent classes, or instances of an existing iiRDS class. For the full rules on extending iiRDS, see the [iiRDS specification](#), sections "iiRDS Standard Extensions" and "Proprietary iiRDS Extensions".

For example, a project can define a custom class `Error Code` as a subclass of the iiRDS class `iirds:Event`, and add concrete error code instances underneath it:

```

<subjectdef keys="iirds.Event">
  <topicmeta>
    <navtitle>event</navtitle>
    <resourceid appid="http://iirds.tekom.de/iirds#Event" appname="class-iri"/>
  </topicmeta>
  <subjectdef keys="product.Error">
    <topicmeta>
      <navtitle>Error Code</navtitle>
      <resourceid appid="https://example.com/product#Error" appname="class-iri"/>
    </topicmeta>
    <subjectdef keys="product.Error.1x111">
      <topicmeta>
        <navtitle>1.x111</navtitle>
        <resourceid appid="https://example.com/product#Error.1x111" appname="instance-iri"/>
      </topicmeta>
    </subjectdef>
  </subjectdef>
</subjectdef>

```

Because `product.Error` is not itself one of the top-level classes, an instance such as `product.Error.1x111` is resolved to its nearest top-level ancestor, `iirds:Event`, and mapped to the iiRDS property `iirds:relates-to-event`.



Important:

The handler determines the property from the hierarchy that is declared in your subject scheme map. It does not independently verify against the iiRDS schemas whether a referenced `@appid` is a valid iiRDS or iiRDS extension IRI, or that the declared parent-child structure matches the real iiRDS taxonomy. Make sure that the IRIs and hierarchy you use are correct so that the resulting iiRDS package conforms to the standard.

Assigning metadata to DITA content

To assign subject scheme metadata to a DITA map or topic, add a `<data>` element with a `@keyref` attribute that references the `@keys` value of an instance `<subjectdef>`. This `<data>` element can be placed:

- In the `<topicmeta>` of the root map, to assign metadata to the resulting `iirds:Document`.
- In the `<topicmeta>` of a `<topicref>`, to assign metadata to the referenced topic. DITA-OT propagates this metadata to the topic itself, so this is a working alternative to placing the metadata directly in the topic.
- In the `<prolog>/<metadata>` element of a topic, to assign metadata directly to that topic.

Example, assigning a product variant and one component to a topic:

```
<data keyref="product.X5-DH2"/>
<data keyref="product.Rotor"/>
```

For details on how to reference and extend subject schemes, and on best practices for organizing keys, see [Assign metadata using a subject scheme \(on page 20\)](#).

Subject scheme class mapping

The `subjectscheme` metadata handler maps instances to iiRDS properties based on the following fixed set of top-level iiRDS classes, which are built into the plugin. Any class nested underneath one of these top-level classes in a subject scheme map, at any depth, is resolved to the same property. See [Controlled iiRDS taxonomies using subject scheme maps \(on page 16\)](#) for details about this resolution.

Table 2. Overview of top-level iiRDS classes and iiRDS properties

Top-level iiRDS class	iiRDS property	Comment
<code>iirds:ProductVariant</code>	<code>iirds:relates-to-product-variant</code>	
<code>iirds:ProductFeature</code>	<code>iirds:relates-to-product-feature</code>	
<code>iirds:Component</code>	<code>iirds:relates-to-component</code>	
<code>iirds:Supply</code>	<code>iirds:relates-to-supply</code>	
<code>iirds:Event</code>	<code>iirds:relates-to-event</code>	
<code>iirds:Qualification</code>	<code>iirds:relates-to-qualification</code>	Includes the predefined classes <code>iirds:Role</code> and <code>iirds:SkillLevel</code> .
<code>iirds:InformationSubject</code>	<code>iirds:has-subject</code>	
<code>iirds:ProductLifeCyclePhase</code>	<code>iirds:relates-to-product-life-cycle-phase</code>	Example instance: <code>iirds:Repair</code> .
<code>iirds:DocumentType</code>	<code>iirds:has-document-type</code> (on <code>iirds:Document</code>) or <code>iirds:is-applicable-for-document-type</code> (on <code>iirds:Topic</code>)	The property depends on whether the metadata is assigned to the root map (<code>iirds:Document</code>) or a topic. Example instance: <code>iirds:OperatingInstructions</code> .
<code>iirds:DocumentCategory</code> (iiRDS handover domain)	<code>iirds:has-document-category</code>	Only applied when the metadata is assigned to the root map (<code>iirds:Document</code>).

A class that is not one of these top-level classes, and not nested underneath one of them, is not mapped to an iiRDS property by the `subjectscheme` handler. Adding a genuinely new top-level class, one that is not a descendant of any class in this table, requires a change to the plugin. See [Known limitations \(on page 24\)](#) for this and other current limitations.

Predefined iiRDS subclasses and instances

The iiRDS standard already defines many classes and instances underneath the top-level classes above, for common cases such as document types, product life cycle phases, and information subjects. For easy application of these predefined entries, this plugin ships with a subject scheme map that

was generated from the iiRDS RDF schemas. After integrating this subject scheme map into your DITA content, the corresponding `<subjectdef>` entries can be referenced using `<data keyref="...">`. Custom subclasses and instances are typically needed for project- or domain-specific concepts, such as product variants and components, that are not already part of the iiRDS standard. These can be integrated into the subject scheme map using custom IRIs and then referenced the same way as the predefined entries.

See [Assign metadata using a subject scheme \(on page 20\)](#) for best practices on extending and using the subject scheme map.

Related information

iiRDS base subject scheme map and generator (on page 23)

Assign metadata using a subject scheme

This procedure describes a best practice for modeling and assigning iiRDS taxonomy metadata in DITA using a project-specific subject scheme map that builds on a base map with default iiRDS classes and instances.

- You have an iiRDS base subject scheme map that contains the standard iiRDS classes and instances. The copy that ships with this plugin is stored in `<DITA-OT>/plugins/org.iirds.dita.package/subjectscheme/iirds-generated-subjectscheme.ditamap`.
- Your root DITA map is available for editing.

A project wants to assign taxonomy-based iiRDS metadata to DITA content, for example, a product structure with components, instead of relying on plain DITA attribute values.

1. Create an empty project-specific subject scheme map that will hold your custom subclasses and instances.
2. In the project-specific subject scheme map, add a reference to the iiRDS base subject scheme map using a `<schemeref>` element.

By using a schema reference, you can easily extend existing classes in the base subject scheme map.

Example:

```
<subjectScheme>
  <schemeref href="iirds-generated-subjectscheme.ditamap"/>
</subjectScheme>
```

**Note:**

Modular subject schemes let you keep the standard iiRDS taxonomies separate from project-specific values. Create this project-specific map even if you do not yet need any project-specific classes or instances. Referencing the project-specific map instead of the generic one from the root map means you can extend the taxonomy later without changing that reference.

3. Reference the project-specific subject scheme map from the root map using a `<mapref>` element with `processing-role="resource-only"`.

Example:

```
<mapref href="product-subjectscheme.ditamap"
  processing-role="resource-only" toc="no"/>
```

**Note:**

The plugin only resolves subject schemes that are directly referenced from the root map, independent of the value of `@processing-role`. Subject schemes provided to DITA-OT via the `--args.resources` parameter are not processed.

4. Start adding your own classes or instances in the project-specific subject scheme map. For each taxonomy, reopen the relevant iiRDS class by repeating its `<topicmeta>` and `<resourceid>` exactly as declared in the base subject scheme map, and nest your own `<subjectdef>` elements underneath it, using a custom IRI.

Example:

```
<subjectScheme>
  <schemeref href="iirds-generated-subjectscheme.ditamap"/>
  <subjectdef keys="iirds.Component">
    <topicmeta>
      <navtitle>component</navtitle>
      <resourceid appid="http://iirds.tekom.de/iirds#Component" appname="class-iri"/>
    </topicmeta>
    <subjectdef keys="myproduct.Rotor">
      <topicmeta>
        <navtitle>Rotor</navtitle>
        <resourceid appid="https://my-company.com/Component#Rotor" appname="instance-iri"/>
      </topicmeta>
    </subjectdef>
  </subjectdef>
</subjectScheme>
```

```

    </subjectdef>

  </subjectdef>

</subjectScheme>

```

**Important:**

Nesting the custom classes and instances underneath an existing iiRDS class is very important and required for the plugin to resolve the new entries with the correct iiRDS property. In this example, `iirds.Component` is a copy from the base subject scheme, while `myproduct.Rotor` is a custom instance.

5. Assign the new metadata values to your content: Add a `<data>` element with a `@keyref` attribute for every metadata value that you want to assign.

The `@keyref` value must match the `@keys` value of a `<subjectdef>`.

Place the `<data>` element in one of the following locations:

- In the `<topicmeta>` of the root map, to assign metadata to the resulting document.
- In the `<topicmeta>` of a `<topicref>`, to assign metadata to the referenced topic.
- In the `<prolog>/<metadata>` element of a topic.

Example, in the root map:

```

<topicmeta>

  <data keyref="product.X5-DH2"/>

  <data keyref="iirds.OperatingInstructions"/>

</topicmeta>

```

6. Run the `dita` command to transform your DITA map using the `iirds` transtype.

The `subjectscheme` metadata handler is applied automatically together with the other default metadata handlers, unless the `--iirds.metadataahandler` parameter is used to restrict the set of handlers. See [Plugin parameters \(on page 28\)](#).

The generated iiRDS package contains the metadata resolved from the subject scheme.

Related information

Subject scheme class mapping *(on page 18)*

iiRDS base subject scheme map and generator *(on page 23)*

Known limitations *(on page 24)*

iiRDS base subject scheme map and generator

Instead of authoring a subject scheme for the standard iiRDS classes and instances by hand, a base subject scheme map is shipped with this plugin, stored in `<DITA-OT>/plugins/org.iirds.dita.package/subjectscheme/iirds-generated-subjectscheme.ditamap`. It reflects all available classes and instances in the iiRDS standard at the publication time of this plugin.

The plugin also comes with the generator that was used to produce this base subject scheme map. It reads the iiRDS RDF schemas that are bundled with the plugin, that is, the core iiRDS schema and the machinery, software, and handover domain schemas, and produces a DITA subject scheme map that contains one `<subjectdef>` per relevant iiRDS class and instance, keeping the class hierarchy expressed in the RDF schemas as nested `<subjectdef>` elements. Classes that are not relevant for metadata assignment, for example, internal iiRDS classes such as `iirds:InformationUnit` or `iirds:DirectoryNode`, are skipped.



Important:

This generator is provided as an internal, developer-facing tool and is not a supported, standalone command-line tool of the iiRDS plugin. It is included in the plugin source code as a convenience, and can be used at your own risk. Its API and output format may change in future plugin versions without prior notice.



Important:

The generator only reads the four RDF schema files that are bundled with the plugin, at a fixed location in the plugin source code. It cannot be pointed at a custom or project-specific RDF file: there is no command-line parameter or other configuration option for this. To add project-specific classes and instances to a subject scheme, use a project-specific subject scheme map instead, as described in [Assign metadata using a subject scheme \(on page 20\)](#).

Running the generator

The generator is the Java class `org.iirds.dita.ot.plugin.IirRDS taxonomyExtractor`, located in the plugin's test sources. It has a `main` method but is not built or packaged as a runnable tool, so it must be run manually, for example, from an IDE, with the plugin's test sources compiled and the plugin's runtime and test dependencies, such as Apache Jena, on the classpath.

Running the class writes two DITA XML fragments to standard output, separated by a line of equals signs (=====):

1. The generated subject scheme map, with the DOCTYPE for a DITA subject scheme map.
2. A DITA concept topic that lists a `<data>` element for every generated instance, with matching `@id` and `@keyref` values.

Copy each of the two fragments into its own file, for example, `iirds-generated-subjectscheme.ditamap` for the first fragment. Since the generator always produces the same output for a given version of the plugin, you only need to run it again after the plugin's bundled iiRDS RDF schemas have changed, for example, after upgrading to a plugin version that supports a newer version of the iiRDS standard.

The source files for the generator in the plugin's test sources are available on GitHub, see <https://github.com/iirds-consortium/dita-ot-plugin/tree/main/org.iirds.dita.ot.plugin/src/test/java/org/iirds/dita/ot/plugin>.

Known limitations

If an iiRDS package generated from DITA content does not contain the expected metadata, note the following about the behavior of the metadata handlers.

To find out how a custom metadata handler can be used with the iiRDS plugin, see [Customize metadata extraction \(on page 32\)](#).

Mapping of standard DITA elements and attributes

- The metadata handler extracts the metadata that is present in the preprocessed XML. The DITA standard defines which metadata is propagated from maps to topics or gets inherited from parent topics. The DITA standard also defines whether multiple values for the same metadata element or attribute are merged into a single value or whether multiple values can be aggregated in a topic. To change this behavior, implement a custom metadata handler.
- The metadata handler processes attribute values as they come. If you use a subject scheme map to control attribute values (not `<data>` elements) and this subject scheme map contains navigation titles with verbose labels for the attribute values, then the resulting RDF will only contain the attribute value. Example: A subject scheme defines the value "product-a" for the `@product` attribute. "product-a" corresponds to the product Name "Awesome Product". The RDF will contain an instance of `iirds:Product = "product-a"`. To change this behavior, implement a custom metadata handler.

- Metadata values are often language-dependent. While product names are usually the same in any language, metadata such as component names or user roles differ from language to language. The plugin has no way of knowing the component "engine" (EN) is the same as the component "Motor" (DE). Therefore, the metadata entries for this component differ in the English and the German iiRDS package even if they mean the same object in the real world. If you need to have the same iiRDS metadata values across languages, you either need to use controlled metadata values in the same language or implement a custom metadata handler that performs an additional processing step during which the values are mapped and harmonized.

Controlled iiRDS taxonomies using subject scheme maps

The `subjectscheme` metadata handler, described in [Controlled iiRDS taxonomies using subject scheme maps \(on page 16\)](#), has the following current limitations:

- Complex metadata structures, for example, product life cycle phases with additional properties or complex identities, are not supported yet.
- Metadata assigned through a subject scheme is not inherited beyond the standard DITA propagation of `<data>` in `<topicmeta>` from a `<topicref>` to its topic. For example, metadata is not automatically inherited from a parent `<topicref>` to its child `<topicref>` elements.
- Subject schemes are only resolved when they are directly referenced from the root map with `<mapref>` or `<schemeref>`, independent of the value of `@processing-role`. Subject schemes provided to DITA-OT with the `--args.resources` parameter are not processed.
- Key scopes are not supported. All `@keys` values of `<subjectdef>` elements are resolved in a single, global scope.
- The language of labels taken from a subject scheme, for example, for generated class labels, is not derived from `@xml:lang`.
- Only the top-level iiRDS classes listed in [Subject scheme class mapping \(on page 18\)](#) are built into the plugin as anchors for resolving an iiRDS property. Adding a genuinely new top-level class, one that is not a descendant of any of these, requires a change to the plugin, for example, by implementing a custom metadata handler.
- The handler resolves iiRDS properties from the class hierarchy as declared in your subject scheme map. It does not verify against the iiRDS schemas that a referenced `@appid` is a valid iiRDS or iiRDS extension IRI, or that the declared hierarchy matches the real iiRDS taxonomy.

Chapter 8. Unique identifiers

To exchange metadata between different systems and map content from one source or language to content from a different source or language, stable and unique identifiers are required. In iiRDS, topics with different metadata or content must have different unique identifiers.

iiRDS metadata is provided as RDF, which uses Internationalized Resource Identifiers (IRIs), the internationalized form of [Uniform Resource Identifiers \(URIs\)](#). DITA does not use IRIs so that unique and stable IRIs have to be derived from the DITA content. Because topic IDs are not unique in all contexts, they cannot be used as IRIs on their own. For example, the same topic may have a different content after conditional filtering has been applied. Therefore, iiRDS topics that are generated from the same source but with different conditional filters, require different IRIs.

The following mechanisms apply:

- The IRI of each `iirds:Topic` consists of the `@id` value of the topic and a hash of the content. The hash is generated after preprocessing and before rendering. The hash is stable so that the same value is generated with each call of the iiRDS transformation, provided that the topic content has not changed.
- The IRI of the `iirds:Document` consists of the `@id` value of the DITA map and a hash of the content. The hash is generated after preprocessing and before rendering. The hash is stable so that the same value is generated with each call of the iiRDS transformation, provided that the map content has not changed.

If a DITA map has no `@id` value, then a UUID is generated and used as the IRI of the `iirds:Document`. This UUID is not identical with each call of the iiRDS transformation.

- For each `iirds:InformationUnit`, an `iirds:InformationObject` is created. This `iirds:InformationObject` has an IRI, which is derived from the topic ID. All information units with the same `@id` value in DITA have the same `iirds:is-version-of` property that refers to this information object.
- The IRI of the `iirds:Package` is a generated UUID.

In addition to this default behavior, the iiRDS plugin provides an extension point that allows to implement other mechanisms for assigning IRIs, for example, to include an additional lookup from other data sources or specialized attributes.

Related information

IRI and metadata handlers (*on page 31*)

Chapter 9. Plugin parameters

The following command-line parameters are provided by the iiRDS plugin:

--iirds.contentpath

Defines the name of the folder in the iiRDS package containing all content files. When not set, `content` is used.

Example: `--iirds.contentpath=files`

--iirds.metadataahandler

Defines iiRDS metadata handlers for metadata extraction from DITA. When not set, all known iiRDS metadata handlers are applied. The parameter value is a + separated list of iiRDS metadata handler names. When a list is provided, then only the listed metadata handlers are used. Priority of the handlers is defined by the order of the list from left to right.

The following example extracts metadata only for `shortdesc` and `prodname` DITA elements: `--iirds.metadataahandler=shortdesc+prodname`.

The following metadata handlers are valid default list values. The default list values can be combined with custom metadata handlers.:

shortdesc

Sets `iirds:has-abstract` to the value of the DITA `shortdesc` element.

prodname

Sets `iirds:relates-to-product-variant` to the value of the DITA `prodname` element.

component

Sets `iirds:relates-to-component` to the value of the DITA `component` element.

product-p

Sets `iirds:relates-to-product-variant` to the value of the DITA `product` attribute on the root element.

critdates

Sets `iirds:dateOfLastModification`, `iirds:dateOfCreation`, `iirds:dateOfEffect`, and `iirds:dateOfExpiry` to the value of the DITA child elements of the `critdates` element.

copyright

Sets `iirds:rights` to the value of the DITA `copyright` element.

audience

Sets `iirds:relates-to-qualification` to the value of the DITA `audience` element.

audience-p

Sets `iirds:relates-to-qualification` to the value of the DITA `audience` attribute on the root element.

subjectscheme

Sets the iIRDS property that matches the resolved class of each `<subjectdef>` referenced by a `<data>` element with a `@keyref` attribute.

default

Applies all metadata handlers. Allows to combine custom metadata handlers with the default handlers.

--iirds.irihandler

Defines which IRI handlers to be used for assigning IRIs to metadata, information units and information objects. The parameter value is a + separated list of IRI handler names. Priority of the handlers is defined by the order of the list from left to right.

The following example combines a custom IRI handler with the default IRI handler: `--`

`iirds.irihandler=customirihandler+default.`

Related information

Customize metadata extraction (*on page 32*)

Customize IRI generation (*on page 32*)

Chapter 10. Customization

The iiRDS plugin can be customized to your specific requirements. The plugin provides the following ways of customization:

- DITA-OT Ant extension points
- Java interface for metadata extraction and IRI generation
- HTML5 DITA-OT build parameters

DITA-OT Ant extension points

The iiRDS plugin provides the following extension points that use Ant targets:

iiRDS.extractMetadata.pre

Runs an Ant target after the DITA-OT preprocessing and before the iiRDS metadata extraction.

iiRDS.extractMetadata.post

Runs an Ant target after the iiRDS metadata extraction. Can be used to access the `metadata.rdf` file.

iiRDS.package.pre

Runs an Ant target after the HTML was built and before the iiRDS package is generated.

For details on adding ANT targets to extension points, see <https://www.dita-ot.org/4.3/topics/plugin-antpreprocess>.

Java interfaces

The iiRDS plugin provides the following Java interfaces:

IiRdsMetadataHandler

Allows to implement project-specific metadata extraction for all RDF resources from DITA content.

IRIHandler

Allows to implement project-specific IRI generation for all RDF resources.

For details, see [Customize metadata extraction \(on page 32\)](#) and [Customize IRI generation \(on page 32\)](#).

HTML5 build parameters

The iiRDS plugin extends the HTML5 transtype and supports the DITA-OT HTML5 parameters. For example, you can add project-specific CSS, inject XML into the HTML5 header element, or use custom XSLT.

For an example, see [Customize HTML5 output \(on page 33\)](#).

IRI and metadata handlers

The iiRDS plugin supports multiple ways to extract metadata out of DITA content and generate IRIs for RDF resources. Metadata is extracted by the plugin's metadata handlers. IRIs are generated by the plugin's IRI handlers.

When the DITA files are processed, the iiRDS plugin tries to extract metadata and generate an IRI based on a sequence of metadata and IRI handlers. The sequence of handlers defines the priority of the handlers. When building iiRDS packages, the `iirds.metadataahandler` and `iirds.irihandler` parameters specify the sequence of handlers. When a parameter is not set, the default fallback is used.

When a handler does not provide an IRI, then the next handler in the sequence is called. As a default fallback, the iiRDS plugin generates unique UUIDs as IRIs for topics, documents, metadata, and information objects.

Custom handlers can be built by extending the Java interfaces `IirdsMetadataHandler` and `IRIHandler`.



Note:

When a metadata handler extracts metadata for a topic which can only be assigned once and another custom metadata handler in the sequence extracts conflicting metadata, then the custom metadata handler has to resolve the conflict.

Related information

[Unique identifiers \(on page 26\)](#)

[Plugin parameters \(on page 28\)](#)

Customize metadata extraction

You can add a customized Java library for metadata extraction that implements the `IirdsMetadataHandler` interface.

- You have a Java library that implements the Service Provider Interface `org.iirds.dita.ot.plugin.spi.IirdsMetadataHandler` and `org.iirds.dita.ot.plugin.spi.IirdsMetadataHandlerProvider`. See https://docs.oracle.com/javase/8/docs/technotes/guides/jar/jar.html#Service_Provider.
- Your Java library is integrated into the DITA Open Toolkit by extending the `dita.conductor.lib.import` extension point. See <https://www.dita-ot.org/4.3/topics/plugin-javalib>.
- Your content contains metadata for your metadata handler to process.

A project wants to build an iIRDS package using a project-specific metadata extraction to anticipate specialized DITA elements.

1. In the `bin` folder of your DITA-OT installation, open a command-line client.
2. In the command-line client, enter the `dita` command to transform your DITA map or topic using the `iirds` transtype and the `iirds.metadataahandler` parameter to use your metadata handler.
Example: `dita --input myContent.ditamap --o out -f iirds -- iirds.metadataahandler=YourMetadataHandler+default`

An iIRDS package with the following file name is created in the output directory:

`<file_name_of_ditamap>.iirds`. The iIRDS package contains metadata based on the project-specific requirements.

Customize IRI generation

You can add a customized Java library for IRI generation that implements the `IRIHandler` interface.

- You have a Java library that implements the Service Provider Interface `org.iirds.dita.ot.plugin.spi.IRIHandler` and implements and registers `org.iirds.dita.ot.plugin.spi.IRIHandlerProvider`. See https://docs.oracle.com/javase/8/docs/technotes/guides/jar/jar.html#Service_Provider.

- Your Java library is integrated into the DITA Open Toolkit by extending the `dita.conductor.lib.import` extension point. See <https://www.dita-ot.org/4.3/topics/plugin-javalib>.
- You provide IRI mappings if required by your IRI handler.

A project wants to build an iiRDS package using a project-specific IRI mapping. The IRI mapping is provided as a CSV file. The CSV file is stored next to the DITA map.

1. In the `bin` folder of your DITA-OT installation, open a command-line client.
2. In the command-line client, enter the `dita` command to transform your DITA map or topic using the `iiRDS` transtype and the `iiRDS.iriHandler` parameter to use your IRI handler.

Example: `dita --input myContent.ditamap --o out -f iiRDS --
iiRDS.iriHandler=YourIriCsvHandler+default`

An iiRDS package with the following file name is created in the output directory:
<file_name_of_ditamap>.iiRDS. The iiRDS package contains IRIs based on the project-specific requirements.

Customize HTML5 output

The iiRDS plugin allows you to customize the HTML5 content of the iiRDS package.

- Custom XSLT is available.
- Custom CSS is available.

A project wants to generate an iiRDS package that contains custom HTML5 content that uses the project's CSS.

1. In the `bin` folder of your DITA-OT installation, open a command-line client.
2. In the command-line client, enter the `dita` command to transform your DITA map or topic using the `iiRDS` transtype and the HTML5 parameters to use your CSS and XSLT.

Example: `dita --input myContent.ditamap --o out -f iiRDS --
args.cssroot=absolutPathToMyCssFolder --args.css=myCustom.css --args.copycss=yes --
args.csspath=outputCssFolderName --args.xsl=relativePathToMyCustomStylesheet.xsl`

An iiRDS package with the following file name is created in the output directory:

`<file_name_of_ditamap>.iirds`. The iiRDS package contains HTML files generated by the project's XSLT and using the project's CSS.

Chapter 11. License and contact information

The plugin is licensed under the open-source license „Apache License, Version 2.0“, including the accompanying documentation and its source code. For details, see <https://www.apache.org/licenses/LICENSE-2.0>.

The plugin was developed for the iIRDS consortium by [Empolis Information Management GmbH](#), testing and documentation was done by [parson AG](#). A special thanks to Robert Johnson for his proposal to use subject scheme maps with keys and @appid attributes to model iIRDS taxonomies.

The sources are available on GitHub together with an example that demonstrates the extensibility of the plugin.

In case of questions, contact the iIRDS consortium, see <https://www.iirds.org/contact>.