



# **POLARION 2310**

## **Resource Traceability SDK**

**Polarion 2310**

**Unpublished work. © 2023 Siemens**

This material contains trade secrets or otherwise confidential information owned by Siemens Industry Software, Inc., its subsidiaries or its affiliates (collectively, "Siemens"), or its licensors. Access to and use of this information is strictly limited as set forth in Customer's applicable agreement with Siemens. This material may not be copied, distributed, or otherwise disclosed outside of Customer's facilities without the express written permission of Siemens, and may not be used in any way not expressly authorized by Siemens.

This document is for information and instruction purposes only. Siemens reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Siemens to determine whether any changes have been made. Representations about products, features or functionality in this document constitute technical information, not a warranty or guarantee, and shall not give rise to any liability of Siemens whatsoever. Siemens disclaims all warranties including, without limitation, the implied warranties of merchantability and fitness for a particular purpose. In particular, Siemens does not warrant that the operation of the products will be uninterrupted or error free.

The terms and conditions governing the sale and licensing of Siemens products are set forth in written agreements between Siemens and its customers. Siemens' End User License Agreement and Universal Contract Agreement may be viewed at: <https://www.sw.siemens.com/en-US/sw-terms/>

**TRADEMARKS:** The trademarks, logos, and service marks ("Marks") used herein are the property of Siemens or other parties. Noone is permitted to use these Marks without the prior written consent of Siemens or the owner of the Marks, as applicable. The use herein of third party Marks is not an attempt to indicate Siemens as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A list of Siemens' trademarks may be viewed at: <https://www.plm.automation.siemens.com/global/en/legal/trademarks.html>.

The registered trademark Linux® is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

**About Siemens Digital Industries Software**

Siemens Digital Industries Software is a leading global provider of product life cycle management (PLM) software and services with 7 million licensed seats and 71,000 customers worldwide. Headquartered in Plano, Texas, Siemens Digital Industries Software works collaboratively with companies to deliver open solutions that help them turn more ideas into successful products. For more information on Siemens Digital Industries Software products and services, visit <https://www.siemens.com/plm>

**Support Center:** <https://www.support.sw.siemens.com>

**Send Feedback on Documentation:** [https://www.support.sw.siemens.com/doc\\_feedback\\_form](https://www.support.sw.siemens.com/doc_feedback_form)

Overview .....	3
Custom Collector Example .....	3
Workspace Preparation .....	3
Create a Project Plugin .....	3
Implementation .....	3
Deployment to Installed Polarion .....	3
Execution from Workspace .....	3
Configuration .....	4
Custom Parser Example .....	4
Workspace Preparation .....	4
Creating Project Plugin .....	4
Implementation .....	4
Deployment to Installed Polarion .....	4
Execution from Workspace .....	4
Configuration .....	4

## Overview

Resource Traceability gives Polarion users the power to link source code assets semantically. (Link Work Items directly to the source code function, method or variable that implements them.) It can also be used to parse files of different formats not related to programming languages. The processing of resource files is done in two steps. In the first step, resource files that can be parsed are collected from the repository. Then collected files are parsed by the compatible parser. These two components of the Resource Traceability feature are called the Collector and Parser respectively. The Collector is responsible for connecting to a specific repository type like SVN or Git and for collecting the files that were updated since the last collection run. Parsers are responsible for parsing file syntax in a specific programming language or file format. They recognize semantic elements and related links and return them to the collection engine. Parsed links from the collected files are persisted by the collector engine in internal storage.

The Resource Traceability SDK provides a way to develop custom Parser and Collector components. Developers should implement the `com.siemens.polarion.rt.collectors.api.RtCollector` interface to define a custom Collector and the `com.siemens.polarion.rt.parsers.api.RtParser` interface for a custom Parser. Developed components are exposed to Polarion by using `extension` elements in the plug-in manifest file, `plugin.xml`. View the javadoc to get more details.

## Custom Collector Example

### Workspace Preparation

See the *Workspace Preparation* section in the main Polarion SDK Guide.

### Create a Project Plugin

1. Select **File > New > Java Project**.
2. In the dialog that appears, enter your new plugin project name and press the **Next** button.
3. On the **Projects** tab add the required projects to the build path. Plugins `com.siemens.polarion.rt.api`, `com.polarion.platform.repository` and `com.polarion.subterra.base` are mandatory and required for projects to develop the Collector. You will also need to add a plugin that implements the `com.polarion.platform.repository.external.IExternalRepositoryProvider` interface to accept the repository connection configuration in your Collector. Other dependencies like `com.polarion.core.util` are optional. (NOTE: Do not create dependencies on the Polarion platform plugins.)
4. Click **Finish**.
5. Create a `plugin.xml` file and add an `extension` element for your Collector so it looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin>
  <extension point="com.siemens.polarion.rt.collectors">
    <collector
      class="com.example.collectors.CustomCollector"
      repositoryProvider="exampleProvider">
    </collector>
  </extension>
</plugin>
```

### Implementation

Implement the `com.siemens.polarion.rt.collectors.api.RtCollector` interface to define a custom Collector. The general algorithm for a custom Collector is the following:

1. Initialize the collector using the context. Use `com.siemens.polarion.rt.collectors.api.RtCollectionContext.getConfiguration()` to get the repository connection configuration and cast the object to a specific RT repository configuration interface. Open the connection to the repository.
2. For each branch that is specified in the configuration do the following:
  1. Determine the last revision for the current branch that was processed during the last collection run.
  2. Collect information about added, modified and removed files since the last processed revision. If the last processed revision is not defined, like when it is first ran, collect information about the files located in the repository.
  3. For each added or modified file invoke the `com.siemens.polarion.rt.collectors.api.RtCollectionContext.shouldBeCollected(RtFileProperties)` method to determine if the file *should* be processed by the parsers. If the file should be collected, notify the collection engine about it by invoking the `com.siemens.polarion.rt.collectors.api.RtCollectionContext.collect(RtFile)` method. For each removed file and folder, invoke the `com.siemens.polarion.rt.collectors.api.RtCollectionContext.locationRemoved(String, ILocation)` method.
3. Dispose used objects.

### Deployment to Installed Polarion

See the *Deployment to Installed Polarion* section in the main Polarion SDK guide.

### Execution from Workspace

See the *Execution from Workspace* section in the main Polarion SDK guide.

## Configuration

After a successful deployment of the plug-in into Polarion, you can start using the Collector with repository providers that have an id equal to the id specified in the `plugin.xml`. Using the mentioned `plugin.xml` a Collector can be developed that will work with the repository provider that has the `exampleProvider` id.

Add such a repository to the `repositories.xml`:

```
<exampleProvider>
  <id>exampleProviderRepository</id>
  <!-- repository settings -->
</exampleProvider>
```

Reference it in `resource-traceability.xml`:

```
<repositoryConfiguration id="exampleRepositoryConfiguration" repository="exampleProviderRepository">
  <!-- configuration parameters -->
</repositoryConfiguration>
```

## Custom Parser Example

### Workspace Preparation

See the *Workspace preparation* section in the main Polarion SDK Guide.

### Creating Project Plugin

1. Select **File > New > Java Project**.
2. In the dialog that appears, enter a new plugin project name and click the **Next** button.
3. On the **Projects** tab add the required projects to the build path. The `com.siemens.polarion.rt.api` plugin is mandatory and required for a project to develop a Parser. Other dependencies like, `com.polarion.core.util` are optional.  
(NOTE: Do not create dependencies on the Polarion platform plugins.)
4. Click **Finish**.
5. Create a `plugin.xml` file and add the extension element for the Parser so it looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin>
  <extension
    point="com.siemens.polarion.rt.parsers">
    <parser
      class="com.example.parsers.CustomParser"
      descriptor="com.example.parsers.CustomParserDescriptor"
      name="com.example.parsers.custom">
    </parser>
  </extension>
</plugin>
```

### Implementation

Implement the `com.siemens.polarion.rt.parsers.api.RtParser` interface to define a custom Parser. The general algorithm for a custom Parser is the following:

1. Receive the file input stream using `com.siemens.polarion.rt.collectors.api.model.RtFile.getContent()`.
2. Read the file content and recognise the semantic elements inside the file. Depending on the source file language, you can use third-party parser implementations to build an AST tree from the file content or make other transformations that create the file content model.
3. Find comments or other text fragments that contain links to Work Items and are related to the semantic elements of the content model.
4. Create link objects from the text using the `com.siemens.polarion.rt.parsers.api.RtParsingContext.parseLinks(String, RtPosition)` method.
5. Build and return element objects that will be persisted in internal storage.

Info: The source code for the C parser example plugin is provided on demand. Please contact Polarion Support.

### Deployment to Installed Polarion

See the *Deployment to Installed Polarion* section in the main Polarion SDK guide.

### Execution from Workspace

See the *Execution from Workspace* section in the main Polarion SDK guide.

### Configuration

After a successful deployment of the plug-in into Polarion, you can start using the Parser for files with a specific extension.

For example, add the following Parser configuration in the `resource-traceability.xml` to use a custom parser for `.sql` files:

```
<fileParser id="customParser" name="com.example.parsers.custom" extensions=".sql">
  <properties>
```

```
</properties>  
</fileParser>
```