LOCKSS Documentation Portal

LOCKSS Program

2023-01-24

GENERAL INFORMATION

Part I

General Information

CHAPTER

ONE

RELEASES

1.1 LOCKSS 2.0-alpha6

development

- LOCKSS 2.0.61-alpha6 (2023-01-23) latest

1.2 LOCKSS 1.76

stable

- LOCKSS 1.76.5 (2023-01-23) latest

1.3 Archived 2.x Releases

1.3.1 LOCKSS 2.0-alpha5

- LOCKSS 2.0.55-alpha5 (2022-07-06)
- LOCKSS 2.0.54-alpha5 (2022-01-27)
- LOCKSS 2.0.53-alpha5 (2022-01-24)
- LOCKSS 2.0.52-alpha5 (2022-01-02)
- LOCKSS 2.0.51-alpha5 (2021-12-17)

1.3.2 LOCKSS 2.0-alpha4

- LOCKSS 2.0.43-alpha4 (2021-12-15)
- LOCKSS 2.0.42-alpha4 (2021-12-13)
- LOCKSS 2.0.41-alpha4 (2021-06-28)

1.3.3 LOCKSS 2.0-alpha3

- LOCKSS 2.0.34-alpha3 (2021-06-04)
- LOCKSS 2.0.33-alpha3 (2021-01-29)
- LOCKSS 2.0.32-alpha3 (2020-11-09)
- LOCKSS 2.0.31-alpha3 (2020-10-29)

1.3.4 LOCKSS 2.0-alpha2

- LOCKSS 2.0.26-alpha2 (2020-02-26)
- LOCKSS 2.0.25-alpha2 (2020-02-25)
- LOCKSS 2.0.24-alpha2 (2020-02-25)
- LOCKSS 2.0.23-alpha2 (2020-02-16)
- LOCKSS 2.0.22-alpha2 (2020-02-10)
- LOCKSS 2.0.21-alpha2 (2020-02-06)

1.3.5 LOCKSS 2.0-alpha1

- LOCKSS 2.0.11-alpha1 (2019-05-13)

1.3.6 LOCKSS 2.0-alpha0

- LOCKSS 2.0-alpha technology preview (2019-04-05)

1.4 Archived 1.x Releases

1.4.1 LOCKSS 1.75

- LOCKSS 1.75.9 (2022-01-10)
- LOCKSS 1.75.8 (2021-12-06)
- LOCKSS 1.75.7 (2021-06-03)
- LOCKSS 1.75.5 (2021-02-10)

1.4.2 LOCKSS 1.74

- LOCKSS 1.74.10 (2020-06-29)
- LOCKSS 1.74.7 (2019-02-06)
- LOCKSS 1.74.3 (2018-11-06)
- LOCKSS 1.74.2 (2018-07-23)

1.4.3 Older Releases

- LOCKSS from 1.54.x to 1.74.3
- LOCKSS from 1.37.x to 1.53.x
- LOCKSS from 1.3.x to 1.36.x

CHAPTER

TWO

SECURITY

This section discusses security issues related to the LOCKSS system.

Date	Reference	Summary
2022-01-02	CVE-2021-45105 CVE-2021-44832	Affects LOCKSS 2.x up to and including 2.0.51-alpha5 Resolved in LOCKSS 2.0.52-alpha5 *Read more...*
2021-12-13	CVE-2021-44228 CVE-2021-45046 CVE-2021-4104	Affects LOCKSS 2.x up to and including 2.0.42-alpha4 Resolved in LOCKSS 2.0.43-alpha4 *Read more...*

2.1 CVE-2021-45105 and CVE-2021-44832

First published: 2022-01-02

Attention: The LOCKSS 2.x system up to and including 2.0.51-alpha5, and the custom Solr and OpenWayback containers it includes, are affected by CVE-2021-45105 and CVE-2021-44832.

Description

Following the early December 2021[1] discovery of well-publicized critical remote code execution vulnerabilities in Apache Log4j 2.x, a ubiquitous Java library for recording information to software logs, additional Log4j 2.x vulnerabilities of moderate severity have been discovered, tracked as CVE-2021-45105 and CVE-2021-44832.

These vulnerabilities affect the LOCKSS system 2.x up to and including 2.0.51-alpha5 (originally released 2021-12-17), and the custom Solr and OpenWayback containers it includes, requiring an upgrade to fix.

Note that the LOCKSS 1.x system is not affected by these vulnerabilities, requiring no action at this time.

[1] See *CVE-2021-44228, CVE-2021-45046 and CVE-2021-4104*.

Remediation

Attention: The recommended remediation is to upgrade LOCKSS 2.0.51-alpha5 and earlier to LOCKSS 2.0.52-alpha5 or later.

- To upgrade from LOCKSS 2.0.51-alpha5 to 2.0.52-alpha5:
  1. Log in to the host system as the `lockss` user and navigate to the `lockss-installer` directory.
  2. Stop the LOCKSS system with this command:

     ```
     scripts/stop-lockss
     ```

  3. Upgrade the LOCKSS Installer to 2.0.52-alpha5 with this command:

     ```
     curl -sSfL https://www.lockss.org/downloader | sh -s - --git-tag=version-2.0.52-
     ↪alpha5
     ```

     or:

     ```
     wget -qO- https://www.lockss.org/downloader | sh -s - --git-tag=version-2.0.52-
     ↪alpha5
     ```

  4. Restart the LOCKSS system with this command:

     ```
     scripts/start-lockss
     ```

- To upgrade from LOCKSS 2.0-alpha4 (all variants) to LOCKSS 2.0.52-alpha5, see Upgrading From LOCKSS 2.0-alpha4 in the LOCKSS 2.0-alpha5 System Manual.
- To upgrade from LOCKSS 2.x version 2.0-alpha3 or earlier (all variants) to LOCKSS 2.0.52-alpha5, you will need to upgrade incrementally; see Upgrading From LOCKSS 2.0-alpha1, Upgrading From LOCKSS 2.0-alpha2, Upgrading From LOCKSS 2.0-alpha3, and Upgrading From LOCKSS 2.0-alpha4.

Important: If you use the LOCKSS 2.x system with an external Solr database or external OpenWayback replay engine, talk to your system administrator about ensuring these external systems, which can also be affected by these vulnerabilities, are up to date.

References

- https://nvd.nist.gov/vuln/detail/CVE-2021-45105
- https://nvd.nist.gov/vuln/detail/CVE-2021-44832
- https://logging.apache.org/log4j/2.x/security.html

2.2 CVE-2021-44228, CVE-2021-45046 and CVE-2021-4104

First published: 2021-12-13
Last updated: 2022-01-02

Attention: The LOCKSS 2.x system up to and including version 2.0.42-alpha4, and the custom Solr and OpenWayback containers it includes, are affected by CVE-2021-44228 ("Log4Shell"), CVE-2021-45046 and CVE-2021-4104.

Description

A critical remote code execution vulnerability has been identified in Apache Log4j 2.x, a ubiquitous Java library for recording information to software logs. Tracked as CVE-2021-44228 and also nicknamed "Log4Shell" or "LogJam", this vulnerability led to the discovery of another critical remote code execution vulnerability severe in Log4j 2.x (CVE-2021-45046) and a related vulnerability in Log4j 1.x (CVE-2021-4104).

These vulnerabilities affect the LOCKSS system 2.x up to and including version 2.0-alpha4b, and the custom Solr and OpenWayback containers it includes, requiring an upgrade to fix.

Note that the LOCKSS 1.x system is not affected by these vulnerabilities, requiring no action at this time.

Remediation

Attention: Because additional vulnerabilities in Log4j 2.x have been discovered, the recommended remediation is to upgrade to LOCKSS version 2.0.42-alpha4 and earlier to LOCKSS 2.0.52-alpha5 immediately.

If you cannot upgrade LOCKSS 2.0.42-alpha4 and earlier to LOCKSS 2.0.52-alpha5 in a timely manner, we recommend at least shutting it down by logging in as the `lockss` user, navigating to the `lockss-installer` directory, and running the command `scripts/stop-lockss`, until such time as you are able to perform an upgrade.

- To upgrade from LOCKSS 2.0-alpha4 (all variants) to LOCKSS 2.0.52-alpha5, see Upgrading From LOCKSS 2.0-alpha4 in the LOCKSS 2.0-alpha5 System Manual.
- To upgrade from LOCKSS 2.0-alpha3 and earlier (all variants), you will need to upgrade incrementally; see Upgrading From LOCKSS 2.0-alpha1, Upgrading From LOCKSS 2.0-alpha2, Upgrading From LOCKSS 2.0-alpha3, and Upgrading From LOCKSS 2.0-alpha4.

Important: If you use the LOCKSS 2.x system with an external Solr database or external OpenWayback replay engine, talk to your system administrator about ensuring these external systems, which can also be affected by these vulnerabilities, are up to date.

References

- https://nvd.nist.gov/vuln/detail/CVE-2021-44228
- https://nvd.nist.gov/vuln/detail/CVE-2021-45046
- https://nvd.nist.gov/vuln/detail/CVE-2021-4104
- https://logging.apache.org/log4j/2.x/security.html

CHAPTER

THREE

ACKNOWLEDGMENTS

The LOCKSS Program is grateful to the many open source software projects on which LOCKSS software relies.

We use the JProfiler Java profiler by ej-technologies GmbH.

Part II

LOCKSS Guides

CHAPTER

FOUR

LOCKSS SOFTWARE DEVELOPER GUIDE

Note: The LOCKSS Software Developer Guide is under construction.

4.1 Classic LOCKSS Development

Note: This page is under construction.

The code base of the Classic LOCKSS system (version 1.x) is contained in a single Git repository, https://github.com/lockss/lockss-daemon.

4.1.1 Prerequisites

To do development work with the Classic LOCKSS system (version 1.x), you will need:

- Git
- Java 8 Development Kit (JDK 8), for example OpenJDK 8
- Apache Ant
- Python 3

Note: A few scripts in the repository use Python 2.7, invoked as **`python2`**.

Installing Git

You can check if Git is installed on your system by typing `git --version` at the command line and seeing if you get a valid response. If you need to install Git, select your operating system below and follow the instructions (as `root`, except for Homebrew on MacOS):

AlmaLinux

To install Git, run this Dnf command (as `root`):

```
dnf --assumeyes install git
```

Arch Linux

To install Git, run this Pacman command (as `root`):

```
pacman -Sy --noconfirm git
```

CentOS

CentOS 7

To install Git, run this Yum command (as `root`):

```
yum --assumeyes install git
```

CentOS 8

To install Git, run this Dnf command (as `root`):

```
dnf --assumeyes install git
```

CentOS Stream 8-9

To install Git, run this Dnf command (as `root`):

```
dnf --assumeyes install git
```

Debian

To install Git, run these Apt commands (as `root`):

```
apt update

apt install --assume-yes git
```

EuroLinux

EuroLinux 7

To install Git, run this Yum command (as `root`):

```
yum --assumeyes install git
```

EuroLinux 8-9

To install Git, run this Dnf command (as `root`):

```
dnf --assumeyes install git
```

Fedora

To install Git, run this Dnf command (as `root`):

```
dnf --assumeyes install git
```

Linux Mint

To install Git, run these Apt commands (as `root`):

```
apt update

apt install --assume-yes git
```

MacOS

Homebrew

To install Git, run this Homebrew command:

```
brew install git
```

MacPorts

To install Git, run this MacPorts command (as `root`):

```
sudo port install git
```

OpenSUSE

OpenSUSE Leap 15

To install Git, run these Zypper commands (as `root`):

```
zypper refresh

zypper --non-interactive install git
```

OpenSUSE Tumbleweed

To install Git, run these Zypper commands (as `root`):

```
zypper refresh

zypper --non-interactive install git
```

Oracle Linux

Oracle Linux 7

To install Git, run this Yum command (as `root`):

```
yum --assumeyes install git
```

Oracle Linux 8-9

To install Git, run this Dnf command (as `root`):

```
dnf --assumeyes install git
```

RHEL

RHEL 7

To install Git, run this Yum command (as `root`):

```
yum --assumeyes install git
```

RHEL 8-9

To install Git, run this Dnf command (as `root`):

```
dnf --assumeyes install git
```

Rocky Linux

To install Git, run this Dnf command (as `root`):

```
dnf --assumeyes install git
```

Scientific Linux 7

To install Git, run this Yum command (as `root`):

```
yum --assumeyes install git
```

Ubuntu

To install Git, run these Apt commands (as `root`):

```
apt update

apt install --assume-yes git
```

Installing the Java Development Kit

You can check if a Java Development Kit (JDK) is installed on your system by typing `javac -version` at the command line and seeing if you get a valid response. (The version numbers output by Java 8 software sometimes use the notation 1.8, for example `1.8.0_332`.)

If you need to install a JDK, we recommend OpenJDK; select your operating system below and follow the instructions (as `root`, except for Homebrew on MacOS):

AlmaLinux

To install OpenJDK, run this Dnf command (as `root`):

```
dnf --assumeyes install java-1.8.0-openjdk-devel
```

Arch Linux

To install OpenJDK, run this Pacman command (as `root`):

```
pacman -Sy --noconfirm jdk8-openjdk
```

CentOS

CentOS 7

To install OpenJDK, run this Yum command (as `root`):

```
yum --assumeyes install java-1.8.0-openjdk-devel
```

CentOS 8

To install OpenJDK, run this Dnf command (as `root`):

```
dnf --assumeyes install java-1.8.0-openjdk-devel
```

CentOS Stream 8-9

To install OpenJDK, run this Dnf command (as `root`):

```
dnf --assumeyes install java-1.8.0-openjdk-devel
```

Debian

To install OpenJDK, run these Apt commands (as `root`):

```
apt update

apt install --assume-yes openjdk-8-jdk
```

EuroLinux

EuroLinux 7

To install OpenJDK, run this Yum command (as `root`):

```
yum --assumeyes install java-1.8.0-openjdk-devel
```

EuroLinux 8-9

To install OpenJDK, run this Dnf command (as `root`):

```
dnf --assumeyes install java-1.8.0-openjdk-devel
```

Fedora

To install OpenJDK, run this Dnf command (as `root`):

```
dnf --assumeyes install java-1.8.0-openjdk-devel
```

Linux Mint

To install OpenJDK, run these Apt commands (as `root`):

```
apt update

apt install --assume-yes openjdk-8-jdk
```

MacOS

Homebrew

To install OpenJDK, run this Homebrew command:

```
brew install openjdk@8
```

Note: You may be directed to create symlinks and/or update your `PATH` to make OpenJDK visible to your system, for instance:

```
For the system Java wrappers to find this JDK, symlink it with
  sudo ln -sfn /usr/local/opt/openjdk@8/libexec/openjdk.jdk /Library/Java/
↪JavaVirtualMachines/openjdk-8.jdk

openjdk@8 is keg-only, which means it was not symlinked into /usr/local,
because this is an alternate version of another formula.

If you need to have openjdk@8 first in your PATH, run:
  echo 'export PATH="/usr/local/opt/openjdk@8/bin:$PATH"' >> ~/.zshrc

For compilers to find openjdk@8 you may need to set:
  export CPPFLAGS="-I/usr/local/opt/openjdk@8/include"
```

MacPorts

To install OPenJDK, run this MacPorts command (as `root`):

```
sudo port install openjdk
```

OpenSUSE

OpenSUSE Leap 15

To install OpenJDK, run these Zypper commands (as `root`):

```
zypper refresh

zypper --non-interactive install java-1_8_0-openjdk-devel
```

OpenSUSE Tumbleweed

To install OpenJDK, run these Zypper commands (as `root`):

```
zypper refresh

zypper --non-interactive install java-1_8_0-openjdk-devel
```

Oracle Linux

Oracle Linux 7

To install OpenJDK, run this Yum command (as `root`):

```
yum --assumeyes install java-1.8.0-openjdk-devel
```

Oracle Linux 8-9

To install OpenJDK, run this Dnf command (as `root`):

```
dnf --assumeyes install java-1.8.0-openjdk-devel
```

RHEL

RHEL 7

To install OpenJDK, run this Yum command (as `root`):

```
yum --assumeyes install java-1.8.0-openjdk-devel
```

RHEL 8-9

To install OpenJDK, run this Dnf command (as `root`):

```
dnf --assumeyes install java-1.8.0-openjdk-devel
```

Rocky Linux

To install OpenJDK, run this Dnf command (as `root`):

```
dnf --assumeyes install java-1.8.0-openjdk-devel
```

Scientific Linux 7

To install OpenJDK, run this Yum command (as `root`):

```
yum --assumeyes install java-1.8.0-openjdk-devel
```

Ubuntu

To install OpenJDK, run these Apt commands (as `root`):

```
apt update

apt install --assume-yes openjdk-8-jdk
```

Installing Apache Ant

You can check if Ant is installed on your system by typing `ant -version` at the command line and seeing if you get a valid response. If you need to install Ant, select your operating system below and follow the instructions (as `root`, except for Homebrew on MacOS):

AlmaLinux

To install Ant, run this Dnf command (as `root`):

```
dnf --assumeyes install ant
```

Arch Linux

To install Ant, run this Pacman command (as `root`):

```
pacman -Sy --noconfirm ant
```

CentOS

CentOS 7

To install Ant, run this Yum command (as `root`):

```
yum --assumeyes install ant
```

CentOS 8

To install Ant, run this Dnf command (as `root`):

```
dnf --assumeyes install ant
```

CentOS Stream 8-9

To install Ant, run this Dnf command (as `root`):

```
dnf --assumeyes install ant
```

Debian

To install Ant, run these Apt commands (as `root`):

```
apt update

apt install --assume-yes ant
```

EuroLinux

EuroLinux 7

To install Ant, run this Yum command (as `root`):

```
yum --assumeyes install ant
```

EuroLinux 8-9

To install Ant, run this Dnf command (as `root`):

```
dnf --assumeyes install ant
```

Fedora

To install Ant, run this Dnf command (as `root`):

```
dnf --assumeyes install ant
```

Linux Mint

To install Ant, run these Apt commands (as `root`):

```
apt update

apt install --assume-yes ant
```

MacOS

Homebrew

To install Ant, run this Homebrew command:

```
brew install ant
```

MacPorts

To install Ant, run this MacPorts command (as `root`):

```
sudo port install apache-ant
```

OpenSUSE

OpenSUSE Leap 15

To install Ant, run these Zypper commands (as `root`):

```
zypper refresh

zypper --non-interactive install ant
```

OpenSUSE Tumbleweed

To install Ant, run these Zypper commands (as `root`):

```
zypper refresh

zypper --non-interactive install ant
```

Oracle Linux

Oracle Linux 7

To install Ant, run this Yum command (as `root`):

```
yum --assumeyes install ant
```

Oracle Linux 8-9

To install Ant, run this Dnf command (as `root`):

```
dnf --assumeyes install ant
```

RHEL

RHEL 7

To install Ant, run this Yum command (as `root`):

```
yum --assumeyes install ant
```

RHEL 8-9

To install Ant, run this Dnf command (as `root`):

```
dnf --assumeyes install ant
```

Rocky Linux

To install Ant, run this Dnf command (as `root`):

```
dnf --assumeyes install ant
```

Scientific Linux 7

To install Ant, run this Yum command (as `root`):

```
yum --assumeyes install ant
```

Ubuntu

To install Ant, run these Apt commands (as `root`):

```
apt update

apt install --assume-yes ant
```

4.1.2 Cloning the Git Repository

To clone the `lockss-daemon` repository from Git, use one of these commands:

```
# GitHub account with SSH key
git clone git@github.com:lockss/lockss-daemon

# Anonymous access
git clone https://github.com/lockss/lockss-daemon
```

This will create a `lockss-daemon` directory.

Additional Prerequisites

- JUnit 3.8.1 is included is included in the LOCKSS source distribution to run unit tests, but the Ant targets that invoke JUnit (`test-xxx`) require the JUnit JAR to be on Ant's `CLASSPATH`. The easiest way to do this is to copy `lib/junit.jar` (relative to the root of the `lockss-daemon` Git tree) into Ant's `lib` directory (relative to its installation directory on the system).
- For some tools and Ant targets, the `JAVA_HOME` environment variable must be set to the directory in which the JDK is installed, i.e. it is expected that `tools.jar` can be found in `$JAVA_HOME/lib`.
- For runtime contexts that process split Zip files and some unit tests, the command-line **`zip`** program must be installed.

  Tip: Most Linux systems have **`zip`** and **`unzip`** installed by default.

4.2 License Templates

Unless otherwise noted, software released by the LOCKSS Program is made available under the terms of the **3-Clause BSD License**, a permissive open-source license[1]. You can use the templates below to add the license to files of various kinds:

- Bash: see *Shell*
- Dockerfile: see *Shell*
- HTML: see *XML*
- *Java*
- *Plain Text*
- *Python*

[1] See also the Software License page on the LOCKSS Web site.

- *Shell*
- *XML*
- YAML: see *Shell*

4.2.1 Plain Text

The following template can be used in **plain text** contexts, for example the `LICENSE` file at the top of a Git repository.

```
Copyright (c) 2000-2023, Board of Trustees of Leland Stanford Jr. University

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors
may be used to endorse or promote products derived from this software without
specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
```

4.2.2 Java

The following template can be used for **Java** files.

```
/*

Copyright (c) 2000-2023, Board of Trustees of Leland Stanford Jr. University

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.

```

(continues on next page)

(continued from previous page)

```
2. Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors
may be used to endorse or promote products derived from this software without
specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.

*/
```

4.2.3 Python

The following template can be used for **Python** files.

```
__copyright__ = '''\
Copyright (c) 2000-2023, Board of Trustees of Leland Stanford Jr. University
'''

__license__ = '''\
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors
may be used to endorse or promote products derived from this software without
specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
```

(continues on next page)

(continued from previous page)

```
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
'''
```

4.2.4 Shell

The following template can be used for **Shell** files.

Tip: This also works for **Bash** files, **Dockerfile** files, **Python requirements** files, or **YAML** files.

```
# Copyright (c) 2000-2023, Board of Trustees of Leland Stanford Jr. University
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions are met:
#
# 1. Redistributions of source code must retain the above copyright notice,
# this list of conditions and the following disclaimer.
#
# 2. Redistributions in binary form must reproduce the above copyright notice,
# this list of conditions and the following disclaimer in the documentation
# and/or other materials provided with the distribution.
#
# 3. Neither the name of the copyright holder nor the names of its contributors
# may be used to endorse or promote products derived from this software without
# specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
# AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
# ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
# LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
# CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
# SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
# INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
# CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
# ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
# POSSIBILITY OF SUCH DAMAGE.
```

4.2.5 XML

The following template can be used for **XML** files.

Tip: This also works for **HTML** files.

```
<!--

Copyright (c) 2000-2023, Board of Trustees of Leland Stanford Jr. University

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors
may be used to endorse or promote products derived from this software without
specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.

-->
```

4.3 REST APIs

The API of each LOCKSS REST service is described in a Swagger 2.0 specification, which can be found relative to the root of the component's Git repository in the file `src/main/resources/swagger/swagger.yaml`. The specification can be used as input into another tool, to produce clients and server stubs in a variety of languages and frameworks, and documentation. This guide contains HTML renderings of each specification generated with Swagger Codegen.

CHAPTER

FIVE

LOCKSS PLUGIN DEVELOPER GUIDE

Note: The LOCKSS Plugin Developer Guide is under construction.

5.1 Introduction

This section offers a conceptual tour of LOCKSS plugins and a brief XML format reference.

5.1.1 LOCKSS Plugin Concepts

LOCKSS Plugin

A **LOCKSS plugin** is a bundle of descriptors, rules and code loaded into the LOCKSS software, describing how to harvest and process a preservation target.

A preservation target might be an individual Web site, or a family of related Web sites (for instance all Web sites powered by the same content publishing platform), or a corpus of Web-accessible resources discovered through some interface (for example an OAI-PMH server, or a service with an API).

Archival Unit

In almost all cases, the preservation target is preserved in plugin-defined chunks called **archival units** (or **AUs** for short).

What these chunks are depends on the situation, but generally the goal is to split the preservation target into chunks of manageable size that are expected to become unchanging eventually, for instance time-bound chunks. For example, in a LOCKSS plugin for a preservation target that consists of serial publications, an AU could be equated with one volume or one year of one publication.

Plugin Configuration Parameters

A LOCKSS plugin leaves placeholders in rules and code called *Plugin Configuration Parameters*. An AU is identified by a plugin and values for each of the plugin's configuration parameters. When the parameter values for an AU are substituted for the placeholders in the plugin's rules and code, the result is rules and code suited for that specific AU.

Typical plugin parameters include a URL prefix or URL fragment (e.g. base URL, directory name...), an identifier (e.g. ISSN of a journal, ISBN of a book, publication code, database identifier of an object...), a year or a date range, a part number (e.g. volume number, numbered subdivision...), and more.

Plugin Format

A LOCKSS plugin is expressed as a mapping from keys to values. Except for rare exceptions that are built into the LOCKSS software, LOCKSS plugins consist of an XML file containing these key-value pairs, accompanied by optional Java class files (compiled Java code), bundled together in a JAR file (a Zip file of Java class files and associated metadata).

Plugin Feature Categories

This guide groups plugin components into conceptual categories, which are introduced briefly in subsections below and have a dedicated section each in this guide:

- *Identification Features*: features related to the identification, versioning and parameterization of the plugin.
- *Crawl Control Features*: features related to the definition and behavior of content crawls.
- *Crawl Validation Features*: features related to content validation in the context of a crawl.
- *Poll Control Features*: features that influence the operation of the LOCKSS audit and repair protocol.
- *Hash Filtering Features*: features related to content canonicalization for inter-node comparison purposes.
- *Metadata Extraction Features*: features related to the extraction and interpretation of metadata from preserved content.
- *Web Replay Features*: features related to supporting the replay of Web content.
- *Inheritance Features*: features related to sharing similar behavior among a set of plugins.
- *Miscellaneous Features*.

Identification Features

All plugins define a number of identifying aspects:

- *Plugin Identifier*: a unique identifier for the plugin.
- *Plugin Name*: a user-friendly name for the plugin.
- *Plugin Version*: the plugin's version number.
- *Plugin Configuration Parameters*: a list of configuration parameter descriptors, defining the placeholders in use in the plugin's rules and code.
- *AU Name*: a rule to generate a default name for each AU based on the plugin name and the plugin parameters, in the event the AU does not have a name in the AU inventory.
- *Required Daemon Version*: the release number of the earliest version of the LOCKSS software that supports all the features required by the plugin.

The *Identification* chapter covers these plugin aspects.

Crawl Control Features

The following plugin aspects can be involved in controlling how content is crawled:

- *Start URLs*: one or more URLs from which the crawl of an AU begins.
- *Crawl Seed*: in lieu of a list of start URLs, code called a crawl seed can compute the starting points of the crawl of an AU, for instance by interacting with an API.
- *Permission URLs*: one or more URLs giving the LOCKSS software permission to crawl an AU, if permission is not given on the start URLs.
- *Per-Host Permission Path*: path where permission statement may be found on hosts not listed in start URLs or Permission URLs. Useful for sites such as Internet Archive that have banks of similar hosts with unpredictable names.
- *Permitted Host Pattern*: pattern rules to allow collection from hosts that cannot explicitly grant permission, for example CDN hosts used to distribute standard components used by web sites such as Javascript libraries.
- *Crawl Rules*: sequential rules determining if a URL discovered during the crawl of an AU should in turn be fetched as part of the AU or not.
- *Crawl Window*: a crawl window controls what times of day or days of the week crawls against the preservation target are allowed; by default an AU is eligible to crawl at any time.
- *Recrawl Interval*: the amount of time before an AU that has previously been crawled successfully is eligible to attempt crawling again.
- *Refetch Depth*: number of links away from the start URL(s) that will be fetched by normal crawls. Deep crawls may be used to cause all URLs in an AU to be refetched (subject to If-Modified-Since).
- *Fetch Pause Time*: the minimum amount of time between two fetches of consecutive URLs in the crawl of an AU.
- *Crawl Rate Limiter*: fine-grained control of the maximum rate at which URLs may be fetched, based on media type, URL pattern, day of week or time of day.
- *Crawl Pool*: controls the number of simultaneous crawls that may be running against any one host or platform.
- *Response Handler*: custom action taken when fetching a URL results in certain error conditions or HTTP response codes.
- *URL Normalizer*: code that normalizes URL variants into canonical URLs.
- *Link Extractor*: media type-specific code that extracts or extrapolates URLs from the collected content, to allow the crawler to follow links. Link extractors are built in for most standard media types that contain links (HTML, CSS, PDF, etc.); plugins may supply link extractors for additional media types or extend the built-in extractors to handle additional constructs.
- *Crawl Filter*: code that filters content before a link extractor is run. Supplements the crawl rules in cases where more context it needed to determine whether a link should be followed.
- *URL Fetcher*: custom code to fetch URLs, for cases that require a more elaborate interaction than a single GET.
- *URL Consumer*: custom code to store collected URLs in the repository. E.g., for sites that redirect permanent URLs to one-time URLs, to store the content at the permanent URL, or to adapt to sites undergoing HTTP to HTTPS transitions

The *Crawl Control* chapter covers these plugin aspects.

Crawl Validation Features

A plugin can optionally define aspects that help verify that the crawl is obtaining the content it is supposed to:

- *Redirect to Login URL Pattern*: determines whether an HTTP redirect returned by the site is actually a redirect to a login page.
- *Login Page Checker*: determines if a URL fetched successfully (HTTP 200) is in fact a login page or some other undesirable substitute for the intended content.
- *Content Validator*: code that determines if certain URLs pass a validation test, most often a media type check or format validation test.
- *Substance Patterns*: pattern rules to check that at least one URL processed during the crawl of an AU is substantive (non-trivial), for example to verify that at least one substantive object was processed rather than just tables of contents.
- *Substance Predicate*: code that determines whether a collected URL has substantive content. Alternative to substance patterns, allows programmatic substance determination.

The *Crawl Validation* chapter covers these plugin aspects.

Poll Control Features

These plugin elements include:

- *Exclude URLs From Polls Pattern*: patterns for URLs that should not be included in polls.
- *Poll Result Weight*: patterns for URLs to allow some disagreements to influence the results more than others.
- *Repair From Publisher When Too Close*: instructs the poller to fetch a new copy of files from the origin site when too-few peers agree on the content.
- *Repair From Peer If Missing*: patterns for URLs that should be fetched from a peer, when the poller detects that they're missing.

The *Poll Control* chapter covers these plugin aspects.

Hash Filtering Features

Many plugins designed to harvest and preserve Web-native content need to go to some lengths to enable comparison of (hashes of) content between the nodes in a LOCKSS network, because fetching a given URL is likely to result in non-identical results from node to node, or from fetch to fetch on the same node. This is due to a raft of causes: advertising banners, personalization ("You are logged in as...", "Downloaded by..."), time-variable content (current date, news ticker), location-variable content (CDN URLs, institution-dependent integration with a link resolver), related content widgets ("You may also be interested in..."), reverse citations and pingbacks ("This page has been referenced by..."), tracking data and watermarking embedded in the content, temporary system messages ("The site will be down for maintenance from..."), and more. Some of these variations now appear outside HTML in media types like PDF or Microsoft PowerPoint files.

To canonicalize content before comparison between nodes in the LOCKSS audit and repair protocol, a plugin can define a **hash filter** for each affected media type. The LOCKSS plugin framework offers a variety of utility classes specifically for **HTML** and **PDF** filtering, as part of its general content filtering framework.

See the *Hash Filtering* chapter for more details.

Metadata Extraction Features

The LOCKSS plugin framework enables the extraction of metadata from ingested content, through an extensible metadata extraction framework; a plugin can optionally define:

- *Article Iterator*: code that traverses the AU and enumerates all the logical items (journal articles, electronic books, electronic theses and dissertations, repository objects...) found in it, as bundles of related URLs.
- *Article Metadata Extractor*: code that extracts metadata from the logical items enumerated by the article iterator using file metadata extractors, and that post-processes and stores the extracted metadata in the LOCKSS metadata database.
- *File Metadata Extractor*: code that extracts metadata from files with a given media type.

The *Metadata Extraction* chapter covers these plugin aspects.

Web Replay Features

A plugin can define optional elements that are applied by the embedded ServeContent Web replay engine:

- *Link Rewriter*: code used by the built-in ServeContent replay engine that changes intra-site links or other URLs to point back to the ServeContent host. Link rewriters are built in for most standard media types that contain links (html, css, javascript, etc.); plugins may supply link rewriters for additional media types or extend the built in rewriters to handle additional constructs.
- *Rewrite HTML Meta URLs*: pattern that determines which HTML meta tags have values that should be rewritten during web replay. Some tags (e.g., citation URLs) should not be rewritten to point back to the ServeContent host.

The *Web Replay* chapter covers these plugin aspects.

Inheritance Features

Commonalities among a set of similar plugins may be abstracted out in to a parent plugin, to reduce duplication. Each child plugin inherits all the elements of the parent plugin.

- *Parent Plugin*: names the parent plugin from which this plugin should inherit elements.
- *Parent Plugin Version*: the version number of the parent plugin, to guard against changed to a parent inadvertently changing the behavior of a child plugin.

The *Inheritance* chapter covers these plugin aspects.

Miscellaneous Features

- **Feature Version Map**: associates version strings with several of the plugin elements. For polling-related elements such as hash filters, the version is used to determine which other peers a peer may invite into polls - the plugin's polling version must be the same across all peers participating in a poll. For metadata extractors and substance checker patterns, the version is used to detect when a change in the plugin may require content to be reprocessed.
- **Feature URLs**: provides information to allow the Open URL resolver to locate articles, issue ToCs, etc.
- **Bulk Content**: declares that the AUs managed by the plugin are not organized semantically (e.g, they may span publications). Affects metadata extraction.
- **Archive File Types**: specifying the types of archive files (zip, tar, etc.) in this plugin's AUs whose members will be individually accessible. Usually used with bulk content plugins to index metadata for archive members.

- **AU Config User Message**: text displayed when a user adds one or more AUs managed by this plugin. Typically used when a site requires crawlers to register with them.
- **Plugin Notes**: commentary displayed along with a plugin's definition in the UI.

Minimalistic Plugin

A simple plugin will likely have at minimum:

- Identifying aspects, including configuration parameters.
- Start URLs, and optionally permission URLs.
- Crawl rules.

If extracting metadata from the preserved content into the LOCKSS metadata database is desired, the plugin will also need:

- Metadata extraction elements, including an article iterator.

If the preserved content consists of non-static HTML Web pages, it will likely need:

- Hash filters.

Use of other aspects is situation-dependent, varying in need based on characteristics and behavior of the preservation target. This guide gives some guidance about when certain components are needed and to what purpose.

Plugin Compatibility Between LOCKSS 1.x and LOCKSS 2.x

Conceptually, LOCKSS plugins are the same in the classic LOCKSS system (LOCKSS 1.x) and in the rearchitected LOCKSS system (LOCKSS 2.x), although future features will only be developed for the rearchitected LOCKSS system without being backported to the classic LOCKSS system, as the classic system becomes deprecated.

5.1.2 LOCKSS Plugin Format

A LOCKSS plugin is expressed as a mapping from keys to values. Except for rare exceptions that are built into the LOCKSS software, LOCKSS plugins consist of an **XML file** containing these key-value pairs, accompanied by optional **Java class files** (compiled Java code), bundled together in a **JAR file** (a Java-specific Zip file).

The XML format of the plugin is a single <map> element, containing any number of **map entries** expressed as <entry> elements. Each map entry is a key-value pair, namely a **plugin key** which must be the first child of the <entry> element and must be of type *String*, and a **plugin value** which must be the second child of the <entry> element. See the *Plugin Value Types* for more about possible plugin values.

The order of the key-value pairs does not matter. The effect of specifying multiple entries with the same key is undefined.

Example:

```
<map>

  <!-- plugin key with string value -->
  <entry>
    <string>key_one</string>
    <string>Lots Of Copies Keep Stuff Safe</string>
  </entry>

  <!-- plugin key with integer value -->
```

(continues on next page)

(continued from previous page)

```
  <entry>
    <string>key_two</string>
    <int>3000</int>
  </entry>

  <!-- plugin key with long integer value (e.g. number of milliseconds) -->
  <entry>
    <string>key_three</string>
    <long>1209600000</long>
  </entry>

  <!-- plugin key with list value (e.g. list of strings) -->
  <entry>
    <string>key_four</string>
    <list>
      <string>List item one</string>
      <string>List item two</string>
      <string>List item three</string>
    </list>
  </entry>

  <!-- plugin key with map value (e.g. mapping from string to string) -->
  <entry>
    <string>key_five</string>
    <map>
      <entry>
        <string>subkey1</string>
        <string>subvalue1</string>
      </entry>
      <entry>
        <string>subkey2</string>
        <string>subvalue2</string>
      </entry>
      <entry>
        <string>subkey3</string>
        <string>subvalue3</string>
      </entry>
    </map>
  </entry>

</map>
```

Plugin Value Types

Plugin values can be of the following types:

XML Element	Plugin Value Type
`<string>`	*String*
`<int>`	*Integer*
`<long>`	*Long Integer*
`<list>`	*List*
`<map>`	*Map*

String

XML Element
`<string>`

Description
An arbitrary string of characters.

Because the plugin is expressed as XML, some characters in the string must be properly encoded:

- & is encoded as `&amp;`.
- < is encoded as `&lt;`.
- > is encoded as `&gt;`.
- Non-printable characters and characters outside the 7-bit ASCII set are encoded with numeric character references[1], either decimal character references (for example é encoded as `&#0233;` or `&#233;`) or hexadecimal character references (for example é encoded as `&#x00e9;` or `&#x00E9;` or `&#xe9;` or `&#xE9;`).

Examples

```
<string>This is a string value</string>

<string>Taylor &amp;amp; Francis</string>

<string>Vive la diff&amp;#x00e9;rence !</string>
```

Integer

XML Element
`<int>`

Description
An integer value. Represented internally as a 32-bit integer.

Example

```
<int>1234</int>
```

[1] See also:
- https://www.w3.org/TR/2006/REC-xml11-20060816/#dt-charref
- https://www.w3.org/TR/2008/REC-xml-20081126/#dt-charref
- https://en.wikipedia.org/wiki/Numeric_character_reference

Long Integer

XML Element
<long>

Description
A long integer value. Represented internally as a 64-bit integer.

Example

```
<long>5000000000</long>
```

List

XML Element
<list>

Description
A <list> elements containing an ordered sequence of values (typically all of the same type).

Example

```
<!-- list of strings -->
<list>
  <string>Item one</string>
  <string>Item two</string>
  <string>Item three</string>
</list>
```

Map

XML Element
<map>

Description
A <map> element containing an unordered sequence of map entries expressed as <entry> elements. Each map entry is a key-value pair; the key must be the first child of the <entry> element and must be a *String*, and the value must be the second child of the <entry> element.

The effect of specifying the same key in more than one map entry is undefined.

Example

```
<!-- mapping from string to string -->
<map>
  <entry>
    <string>key1</string>
    <string>value1</string>
  </entry>
  <entry>
    <string>key2</string>
    <string>value2</string>
  </entry>
  <entry>
```

(continues on next page)

(continued from previous page)

```
    <string>key3</string>
    <string>value3</string>
  </entry>
</map>
```

5.2 Identification

This section introduces plugin features related to the identification, versioning and parameterization of the plugin.

5.2.1 Plugin Identifier

Plugin Key
`plugin_identifier`

Plugin Value Type
String

Sample

```
<entry>
  <string>plugin_identifier</string>
  <string>edu.example.plugin.publisherx.PublisherXPlugin</string>
</entry>
```

Description
A unique identifier for the plugin.

The plugin identifier uniquely identifies the plugin. It is used as part of AUIDs (the unique identifier of each AU) and to name a Java package where the plugin's Java code lives. As such, the plugin identifier is really a **fully-qualified Java class name**, which consists of a Java package name and a Java class name.

Just like a file path such as `/home/jsmith/documents/myfile.txt` is a hierarchical path from more general directories (`home`) to more specific directories (`jsmith`, then `documents`) and ending with a file name (`myfile.txt`), a fully-qualified Java class name is a hierarchical path starting with the institution responsible for the plugin (conventionally by reversed Internet domain name), for example `edu.stanford.library` for the organization whose Web site is `library.stanford.edu`), followed by more levels (typically the next one `plugin`, and then another to identify the plugin family or individual plugin), and finally ending with a "file name". The separators are periods.

Example
In the Global LOCKSS Network (GLN), the plugin maintained by the LOCKSS Program to process volumes of journals by Oxford University Press (OUP) hosted on the Silverchair platform has the identifier `org.lockss.plugin.silverchair.oup.OupSilverchairPlugin`:

- `org.lockss` is the reverse of `lockss.org`.
- `plugin` is the root level of all plugins maintained by the LOCKSS Program.
- `silverchair` is a level grouping all Silverchair-based plugins.
- `oup` is the most specific level identifying the OUP plugin.
- `OupSilverchairPlugin` is the class name.

```
<entry>
  <string>plugin_identifier</string>
  <string>org.lockss.plugin.silverchair.oup.OupSilverchairPlugin</string>
</entry>
```

(You can find this plugin on GitHub at https://github.com/lockss/lockss-daemon/blob/master/plugins/src/org/lockss/plugin/silverchair/oup/OupSilverchairPlugin.xml.)

File Path

In a source code repository, the plugin identifier translates into a file path like so: `org.lockss.plugin.silverchair.oup.OupSilverchairPlugin` corresponds to `org/lockss/plugin/silverchair/oup/OupSilverchairPlugin.xml`.

5.2.2 Plugin Name

Plugin Key

`plugin_name`

Plugin Value Type

String

Sample

```
<entry>
  <string>plugin_name</string>
  <string>Publisher X Journals Plugin</string>
</entry>
```

Description

A user-friendly name for the plugin.

This name is only displayed in a few places in the LOCKSS Web user interface, for instance in the Plugins table.

Example

In the Global LOCKSS Network (GLN), the plugin to process volumes of journals by Oxford University Press (OUP) hosted on the Silverchair platform has the name `Oxford University Press Journals Plugin`.

```
<entry>
  <string>plugin_name</string>
  <string>Oxford University Press Plugin</string>
</entry>
```

(You can find this plugin on GitHub at https://github.com/lockss/lockss-daemon/blob/master/plugins/src/org/lockss/plugin/silverchair/oup/OupSilverchairPlugin.xml.)

5.2.3 Plugin Version

Plugin Key
`plugin_version`

Plugin Value Type
String (not integer)

Sample

```
<entry>
  <string>plugin_version</string>
  <string>7</string>
</entry>
```

Description
The plugin's version number.

The first release of a plugin is typically numbered 1, and the next revision 2, and so on, although the releases do not have to be numerically consecutive as long as they only go up with time.

Although only the numeric part is important, the integer can be followed by a hyphen and an arbitrary string, which is why this value is of type is **string** and not integer.

5.2.4 Plugin Configuration Parameters

Plugin Key
`plugin_config_props`

Plugin Value Type
List of `<org.lockss.daemon.ConfigParamDescr>` stanzas

Sample

```
<entry>
  <string>plugin_config_props</string>
  <list>
    <org.lockss.daemon.ConfigParamDescr>
      <key>base_url</key>
      <displayName>Base URL</displayName>
      <description>Usually of the form http://&lt;journal-name&gt;.com/</
↪description>
      <type>3</type>
      <size>40</size>
      <definitional>true</definitional>
      <defaultOnly>false</defaultOnly>
    </org.lockss.daemon.ConfigParamDescr>
    <org.lockss.daemon.ConfigParamDescr>
      <key>journal_id</key>
      <displayName>Journal Identifier</displayName>
      <description>Identifier for journal (often used as part of file names)</
↪description>
      <type>1</type>
      <size>40</size>
      <definitional>true</definitional>
```

(continues on next page)

(continued from previous page)

```
      <defaultOnly>false</defaultOnly>
    </org.lockss.daemon.ConfigParamDescr>
    <org.lockss.daemon.ConfigParamDescr>
      <key>volume_name</key>
      <displayName>Volume Name</displayName>
      <type>1</type>
      <size>20</size>
      <definitional>true</definitional>
      <defaultOnly>false</defaultOnly>
    </org.lockss.daemon.ConfigParamDescr>
  </list>
</entry>
```

Description

A list of **configuration parameter descriptors**, defining the placeholders in use in the plugin's rules and code.

A plugin's rules and code (start and permission URLs, crawl rules, substance patterns...) are made general by identifying placeholders for AU-specific values and substituting them later. These placeholders for variable values are called **plugin configuration parameters**.

Defining the necessary configuration parameters for a given plugin comes mostly from studying the URL structure of the preservation target, finding patterns, and identifying the parts of those patterns that differ between Archival Units.

Structure

Each plugin configuration parameter is represented by a <org.lockss.daemon.ConfigParamDescr> stanza that looks like this:

```
<org.lockss.daemon.ConfigParamDescr>
  <key>...</key>
  <type>...</type>
  <displayName>...</displayName>
  <description>...</description>
  <size>...</size>
  <definitional>...</definitional>  <!-- default: true -->
  <defaultOnly>...</defaultOnly>    <!-- default: false -->
</org.lockss.daemon.ConfigParamDescr>
```

Only <key> and <type> are required.

Each <org.lockss.daemon.ConfigParamDescr> stanza contains the following important elements:

- <key>: the **parameter key**, an identifier for the configuration parameter, standing in as a placeholder for the AU-specific value in rules and code. Example: `base_url` for a base URL (URL prefix common to all or most URLs in an AU).
- <type>: the **parameter type**, an integer describing the type of value the configuration parameter represents (string, integer, etc.). See *Parameter Types* below for details.
- <definitional>: whether the parameter is a **definitional parameter** or **non-definitional parameter**, expressed as the booleans `true` or `false`. Most parameters are definitional (`true`), meaning the parameter is part of the set of parameters that together form the unique identity of the AU.
- <defaultOnly>: set to `false` in almost all cases.

The other elements only play a role in the *Manual Add/Edit* screen in the LOCKSS Web user interface:

- `<displayName>`: the **parameter display name**, a user-friendly name for the parameter in in the *Manual Add/Edit* screen.
- `<description>`: the **parameter description**, a user-friendly text string describing the parameter and giving an example value in the *Manual Add/Edit* screen.
- `<size>`: the parameter display size in characters in the *Manual Add/Edit* screen.

Parameter Types

The following plugin configuration parameter types are defined in the LOCKSS software:

Parameter Type Code	Parameter Type
1	*String*
2	*Integer*
3	*URL*
4	*Year*
5	*Boolean*
6	*Non-Negative Integer*
7	*String Range*
8	*Numeric Range*
9	*Set*
10	*User Credentials*
11	*Long Integer*
12	*Time Interval*

String

Parameter Type Code
1

Description
A non-empty string.

Built-In Examples
Volume Name, *Journal Directory*, *Journal Abbreviation*, *Journal Identifier*, *Journal ISSN*, *Publisher Name*, *OAI Spec*, *Crawl Proxy*, *Crawl Test Substance Threshold*

URL

Parameter Type Code
3

Description
Used most frequently as a URL prefix. This must be a valid URL string according to Java's `java.net.URL` constructor (https://docs.oracle.com/javase/8/docs/api/java/net/URL.html#URL-java.lang.String-).

Built-In Examples
Base URL, *Second Base URL*, *OAI Request URL*

See Also
Derivative URL Parameters

User Credentials

Parameter Type Code
`10`

Description
A colon-separated username and password, for instance `myuser:mypass`.

Built-In Examples
Username and Password

Integer

Parameter Type Code
`2`

Description
The integer can be negative. Represented internally as a 32-bit integer.

Non-Negative Integer

Parameter Type Code
`6`

Description
The integer can be zero but cannot be negative. Represented internally as a 32-bit integer.

Built-In Examples
Volume Number

Long Integer

Parameter Type Code
`11`

Description
The value can be negative. Represented internally as a 64-bit integer.

Year

Parameter Type Code
`4`

Description
A four-digit year, or the special value *0* to denote an unspecified year.

Built-In Examples
Year

See Also
Derivative Year Parameters

Time Interval

Parameter Type Code
12

Description
Specified as a long integer followed by a suffix indicating a time unit: `ms` for milliseconds, `s` for seconds, `m` for minutes, `h` for hours, `d` for days, `w` for weeks (7 days), `y` for years (365 days). If there is no suffix, the default interpretation is milliseconds. The time unit suffixes are case-insensitive.

Built-In Examples
New Content Crawl Interval

String Range

Parameter Type Code
7

Description
The range is specified with two strings separated by a dash (`-`) and is inclusive. If there is a single string with no dash, the range is interpreted to contain only that string.

Built-In Examples
Issue Range

Numeric Range

Parameter Type Code
8

Description
The range is specified with two integers separated by a dash (`-`). If there is a single integer, the range is interpreted to contain only that integer.

Built-In Examples
Numeric Issue Range

Set

Parameter Type Code
9

Description
Specified as a comma-separated list of strings, with whitespace surrounding strings ignored, and empty strings discarded.

The string `{n,m}`, where *n* and *m* are integers, will be replaced by all the integers in the range from *n* to *m* inclusive. For instance, the set `{2002-2005}, 2003Supp, 2004Supp` is equivalent to `2002, 2003, 2003Supp, 2004, 2004Supp, 2005`.

Built-In Examples
Issue Set

Boolean

Parameter Type Code
5

Description
The canonical values are `true` or `false`, although `yes`, `on` and `1` are recognized as `true`, and `no`, `off` and `0` are recognized as `false`. All these value strings are case-insensitive.

Built-In Examples
AU Down, *AU Off-Limits*, *AU Closed*

Built-In Definitional Parameters

The LOCKSS software defines a number of built-in definitional parameters.

Definitional parameters give an AU its identity -- change the value for a definitional parameter and you will be describing a different slice of content (different year, different directory, etc.).

Base URL

Parameter Key
`base_url`

Parameter Type
URL

Canonical Form

```
<org.lockss.daemon.ConfigParamDescr>
  <key>base_url</key>
  <type>3</type>
  <displayName>Base URL</displayName>
  <description>Usually of the form http://&lt;journal-name&gt;.com/</description>
  <size>40</size>
  <definitional>true</definitional>
  <defaultOnly>false</defaultOnly>
</org.lockss.daemon.ConfigParamDescr>
```

Second Base URL

Parameter Key
`base_url2`

Parameter Type
URL

Canonical Form

```
<org.lockss.daemon.ConfigParamDescr>
  <key>base_url2</key>
  <type>3</type>
  <displayName>Second Base URL</displayName>
```

(continues on next page)

(continued from previous page)

```
  <description>Use if AU spans two hosts</description>
  <size>40</size>
  <definitional>true</definitional>
  <defaultOnly>false</defaultOnly>
</org.lockss.daemon.ConfigParamDescr>
```

Year

Parameter Key
year

Parameter Type
Year

Canonical Form

```
<org.lockss.daemon.ConfigParamDescr>
  <key>year</key>
  <type>4</type>
  <displayName>Year</displayName>
  <description>Four digit year (e.g., 2004)</description>
  <size>4</size>
  <definitional>true</definitional>
  <defaultOnly>false</defaultOnly>
</org.lockss.daemon.ConfigParamDescr>
```

Volume Number

Parameter key
volume

Parameter Type
Non-Negative Integer

Canonical Form

```
<org.lockss.daemon.ConfigParamDescr>
  <key>volume</key>
  <type>6</type>
  <displayName>Volume No.</displayName>
  <description>Numeric volume number, e.g. 7</description>
  <size>8</size>
  <definitional>true</definitional>
  <defaultOnly>false</defaultOnly>
</org.lockss.daemon.ConfigParamDescr>
```

Volume Name

Parameter Key
volume_name

Parameter Type
String

Canonical Form

```
<org.lockss.daemon.ConfigParamDescr>
  <key>volume_name</key>
  <type>1</type>
  <displayName>Volume Name</displayName>
  <description>Volume name, e.g. 23A</description>
  <size>20</size>
  <definitional>true</definitional>
  <defaultOnly>false</defaultOnly>
</org.lockss.daemon.ConfigParamDescr>
```

Issue Range

Parameter Key
issue_range

Parameter Type
String Range

Canonical Form

```
<org.lockss.daemon.ConfigParamDescr>
  <key>issue_range</key>
  <type>7</type>
  <displayName>Issue Range</displayName>
  <description>A Range of issues in the form: aaa-zzz</description>
  <size>20</size>
  <definitional>true</definitional>
  <defaultOnly>false</defaultOnly>
</org.lockss.daemon.ConfigParamDescr>
```

Numeric Issue Range

Parameter Key:
num_issue_range

Parameter Type
Numeric Range

Canonical Form

```
<org.lockss.daemon.ConfigParamDescr>
  <key>num_issue_range</key>
  <displayName>Numeric Issue Range</displayName>
```

(continues on next page)

(continued from previous page)

```
  <description>A Range of issues in the form: min-max</description>
  <type>8</type>
  <size>20</size>
  <definitional>true</definitional>
  <defaultOnly>false</defaultOnly>
</org.lockss.daemon.ConfigParamDescr>
```

Issue Set

Parameter Key
`issue_set`

Parameter Type
Set

Canonical Form

```
<org.lockss.daemon.ConfigParamDescr>
  <key>issue_set</key>
  <type>9</type>
  <displayName>Issue Set</displayName>
  <description>A comma delimited list of issues. (eg issue1, issue2)</description>
  <size>20</size>
  <definitional>true</definitional>
  <defaultOnly>false</defaultOnly>
</org.lockss.daemon.ConfigParamDescr>
```

Journal Directory

Parameter Key
`journal_dir`

Parameter Type
String

Canonical Form

```
<org.lockss.daemon.ConfigParamDescr>
  <key>journal_dir</key>
  <type>1</type>
  <displayName>Journal Directory</displayName>
  <description>Directory name for journal content (i.e. 'american_imago').</
↪description>
  <size>40</size>
  <definitional>true</definitional>
  <defaultOnly>false</defaultOnly>
</org.lockss.daemon.ConfigParamDescr>
```

Journal Abbreviation

Parameter Key
journal_abbr

Parameter Type
String

Canonical Form

```
<org.lockss.daemon.ConfigParamDescr>
  <key>journal_abbr</key>
  <type>1</type>
  <displayName>Journal Abbreviation</displayName>
  <description>Abbreviation for journal (often used as part of file names).</
↪description>
  <size>10</size>
  <definitional>true</definitional>
  <defaultOnly>false</defaultOnly>
</org.lockss.daemon.ConfigParamDescr>
```

Journal Identifier

Parameter Key
journal_id

Parameter type
String

Canonical Form

```
<org.lockss.daemon.ConfigParamDescr>
  <key>journal_id</key>
  <type>1</type>
  <displayName>Journal Identifier</displayName>
  <description>Identifier for journal (often used as part of file names).</
↪description>
  <size>40</size>
  <definitional>true</definitional>
  <defaultOnly>false</defaultOnly>
</org.lockss.daemon.ConfigParamDescr>
```

Journal ISSN

Parameter Key
journal_issn

Parameter Type
String

Canonical Form

```
<org.lockss.daemon.ConfigParamDescr>
  <key>journal_issn</key>
  <type>1</type>
  <displayName>Journal ISSN</displayName>
  <description>International Standard Serial Number.</description>
  <size>20</size>
  <definitional>true</definitional>
  <defaultOnly>false</defaultOnly>
</org.lockss.daemon.ConfigParamDescr>
```

Publisher Name

Note: Use of this parameter is not recommended. It is unlikely the publisher name will appear in URLs, as opposed to a publisher abbreviation or code.

Parameter Key
publisher_name

Parameter Type
String

Canonical Form

```
<org.lockss.daemon.ConfigParamDescr>
  <key>publisher_name</key>
  <type>1</type>
  <displayName>Publisher Name</displayName>
  <description>Publisher Name for Archival Unit</description>
  <size>40</size>
  <definitional>true</definitional>
  <defaultOnly>false</defaultOnly>
</org.lockss.daemon.ConfigParamDescr>
```

OAI Request URL

Parameter Key
oai_request_url

Parameter Type
URL

Canonical Form

```
<org.lockss.daemon.ConfigParamDescr>
  <key>oai_request_url</key>
  <type>3</type>
  <displayName>OAI Request URL</displayName>
  <description>Usually of the form http://&lt;journal-name&gt;.com/</description>
  <size>40</size>
  <definitional>true</definitional>
```

(continues on next page)

(continued from previous page)

```
  <defaultOnly>false</defaultOnly>
</org.lockss.daemon.ConfigParamDescr>
```

OAI Spec

Parameter Key
`oai_spec`

Parameter Type
String

Canonical Form

```
<org.lockss.daemon.ConfigParamDescr>
  <key>oai_spec</key>
  <type>1</type>
  <displayName>OAI Spec</displayName>
  <description>Spec for journal in the OAI crawl</description>
  <size>40</size>
  <definitional>true</definitional>
  <defaultOnly>false</defaultOnly>
</org.lockss.daemon.ConfigParamDescr>
```

Built-In Non-Definitional Parameters

The LOCKSS software also defines a number of non-definitional parameters.

Non-definitional parameters are necessary as placeholders in plugin rules and code, but they do not contribute to the AU's identity -- you may need to change the value of a non-definitional parameter but it will not change which slice of content the AU corresponds to.

Some non-definitional parameters might be listed in the plugin itself, like the `user_pass` parameter for user credentials, if all AUs are expected to supply a value for the parameter, but most others are involved in the lifecycle of an AU and need not be listed in the plugin, like the `pub_down` parameter for AUs that are not currently allowed to crawl.

Username and Password

Parameter Key
`user_pass`

Parameter Type
User Credentials

Canonical Form

```
<org.lockss.daemon.ConfigParamDescr>
  <key>user_pass</key>
  <type>10</type>
  <displayName>Username:Password</displayName>
  <description>Colon-separated username and password string, e.g. myuser:mypass</
↪description>
  <size>30</size>
```

(continues on next page)

(continued from previous page)

```
  <definitional>false</definitional>
  <defaultOnly>false</defaultOnly>
</org.lockss.daemon.ConfigParamDescr>
```

Description

Some harvesting processes may require user credentials (username and password). A non-definitional parameter is needed because the username and password might be different for different harvesting nodes, or may change over time, without changing the identity of the AU (for instance its year).

AU Down

Parameter Key

pub_down

Parameter Type

Boolean

Canonical Form

```
<org.lockss.daemon.ConfigParamDescr>
  <key>pub_down</key>
  <type>5</type>
  <displayName>Pub Down</displayName>
  <description>If true, AU is no longer available from the publisher</description>
  <size>4</size>
  <definitional>false</definitional>
  <defaultOnly>true</defaultOnly>
</org.lockss.daemon.ConfigParamDescr>
```

Description

This non-definitional parameter is used routinely in the title database files of LOCKSS networks, but does not need to appear explicitly in plugins.

When this parameter value is supplied as `true` for an AU, the AU is considered to be "down", meaning that it is currently unavailable from its source and should not attempt to crawl or recrawl.

The name pub_down, for "publisher down", reflects the idea that the entire publisher site (content provider) might be unavailable, but this parameter can be used to mark individual AUs as being down outside the context of an entire content provider being unavailable.

AU Off-Limits

Parameter Key

pub_never

Parameter Type

Boolean

Canonical Form

```
<org.lockss.daemon.ConfigParamDescr>
  <key>pub_never</key>
  <type>5</type>
```

(continues on next page)

(continued from previous page)

```
  <displayName>Pub Never</displayName>
  <description>If true, don't try to access any content from publisher</description>
  <size>4</size>
  <definitional>false</definitional>
  <defaultOnly>true</defaultOnly>
</org.lockss.daemon.ConfigParamDescr>
```

Description

This non-definitional parameter is used routinely in the title database files of LOCKSS networks, but does not need to appear explicitly in plugins.

When this parameter value is supplied as `true` for an AU, the AU is considered to be "off-limits", meaning that the LOCKSS software will not satisfy a proxy request for a URL it determines to be in this AU by going to the original Web site.

AU Closed

Parameter Key

`au_closed`

Parameter Type

Boolean

Canonical Form

```
<org.lockss.daemon.ConfigParamDescr>
  <key>au_closed</key>
  <type>5</type>
  <displayName>AU Closed</displayName>
  <description>If true, AU is complete, no more content will be added</description>
  <size>4</size>
  <definitional>false</definitional>
  <defaultOnly>true</defaultOnly>
</org.lockss.daemon.ConfigParamDescr>
```

Description

This non-definitional parameter is used routinely in the title database files of LOCKSS networks, but does not need to appear explicitly in plugins.

When this parameter value is supplied as `true` for an AU, the AU is marked as "closed", meaning it is considered that no more content will be added to it in the future.

Crawl Proxy

Parameter Key

`crawl_proxy`

Parameter Type

String

Canonical Form

```
<org.lockss.daemon.ConfigParamDescr>
  <key>crawl_proxy</key>
  <type>1</type>
  <displayName>Crawl Proxy</displayName>
  <description>If set to host:port, crawls of this AU will be proxied. If set to␣
↪DIRECT, crawls will not be proxied, even if the LOCKSS node has been configured␣
↪with a default crawl proxy.</description>
  <size>40</size>
  <definitional>false</definitional>
  <defaultOnly>true</defaultOnly>
</org.lockss.daemon.ConfigParamDescr>
```

Description

This non-definitional parameter is used routinely in the title database files of LOCKSS networks, but does not need to appear explicitly in plugins.

When this parameter value is supplied as a host:port pair (for example `proxy.myuniversity.edu:8080`) for an AU, crawls of the AU will be proxied through the given proxy. When this parameter value is supplied as the special value `DIRECT` for an AU, crawls of the AU will not be proxied, even if the LOCKSS node is configured to always use a crawl proxy.

New Content Crawl Interval

Parameter Key

`nc_interval`

Parameter Type

Time Interval

Canonical Form

```
<org.lockss.daemon.ConfigParamDescr>
  <key>nc_interval</key>
  <type>12</type>
  <displayName>Crawl Interval</displayName>
  <description>The interval at which the AU should crawl the publisher site.</
↪description>
  <size>10</size>
  <definitional>false</definitional>
  <defaultOnly>true</defaultOnly>
</org.lockss.daemon.ConfigParamDescr>
```

Description

This non-definitional parameter is used routinely in the title database files of LOCKSS networks, but does not need to appear explicitly in plugins.

When this parameter value is supplied as a time interval for an AU, crawls of the AU will be attempted with the given requested interval rather than the LOCKSS node's default new content crawl interval.

Crawl Test Substance Threshold

Parameter Key
`crawl_test_substance_threshold`

Parameter Type
String

Canonical Form

```
<org.lockss.daemon.ConfigParamDescr>
  <key>crawl_test_substance_threshold</key>
  <type>1</type>
  <displayName>Crawl Test Substance Threshold</displayName>
  <description>Minimum number of substance URLs necessary for successful␣
↪abbreviated crawl test.</description>
  <size>20</size>
  <definitional>false</definitional>
  <defaultOnly>true</defaultOnly>
</org.lockss.daemon.ConfigParamDescr>
```

Description
This non-definitional parameter is used in special circumstances, for networks set up to perform abbreviated test crawls.

Derivative Parameters

For parameters of type *URL* and *Year*, the system automatically brings into existence **derivative parameters** with special names, as if those parameters had also been defined by the plugin.

Tip: Derivative parameters have fallen out of favor. The contemporary way to achieve the same effect is through **parameter functors**.

Derivative URL Parameters

For any parameter of type *URL* with key *urlkey*, the following derivative parameters are automatically defined:

- *urlkey*`_host` of type *String*, whose value is just the host portion of the corresponding URL value. For example, if `base_url` has a value of `https://www.publisher.com/jabc/`, `base_url_host` has a value of `www.publisher.com`.
- *urlkey*`_path` of type *String*, whose value is just the path portion of the corresponding URL value. For example, if `base_url` has a value of `https://www.publisher.com/jabc/`, `base_url_path` has a value of `/jabc/`.

Derivative Year Parameters

For any parameter of type *Year* with key *yearkey*, the following derivative parameter is automatically defined:

- au_short_*yearkey* of type *Integer*, whose value is the corresponding year value modulo 100. For example, if year has a value of 1998, au_short_year has a value of 98; if year has a value of 2002, au_short_year has a value of 2 (the integer 2, not the string 02.

> **Tip:** In many cases, what is useful is the zero-padded, two-character string from the derivative short year, not the potentially single-digit integer; use %02d in the printf format string.

5.2.5 AU Name

Plugin Key
au_name

Plugin Value Type
String

Plugin Value Format
The value is a printf format string, that expands to a string. The *printf* format string can use plugin configuration parameter keys (e.g. base_url, journal_issn, volume_name) as values.

Sample

```
<entry>
  <string>au_name</string>
  <string>"Publisher X Journals Plugin, Base URL %s, Journal Identifier %s, Volume
↪%s", base_url, journal_id, volume_name</string>
</entry>
```

Description
A rule to generate a default name for each AU, based on the plugin name and the plugin parameters. The rule is used to generate a name for the AU if it is not listed in the title database (AU inventory).

Conventionally, this is made of a comma-separated list of the *Plugin Name* and the display name and value of each of the *Plugin Configuration Parameters*, from more general (e.g. base_url) to more specific (e.g. volume_name).

5.2.6 Required Daemon Version

Plugin Key
required_daemon_version

Plugin Value Type
String

Sample

```
<entry>
  <string>required_daemon_version</string>
  <string>1.74.7</string>
</entry>
```

Description
The release number of the earliest version of the LOCKSS software that supports all the features required by the plugin.

In the classic LOCKSS system (1.x), this is a string like *1.75.9* (version 1.75.9 or higher) or *1.75.0* (version 1.75.x or higher). In the rearchitected LOCKSS system (2.x), this is currently only *2.0-alpha1* through *2.0-alpha5*.

5.3 Crawl Control

This section introduces plugin features related to the definition and behavior of content crawls.

5.3.1 Start URLs

Plugin Key
`au_start_url`

Plugin Value Type
One of:

- *String*
- *List* of *String*

Plugin Value Format
The strings are `printf` format strings, that expand to URLs. The `printf` format strings accept expressions made of plugin configuration parameter keys and a small language of functions modifying them (e.g. `url_host(...)` applied to a plugin configuration parameter of type URL, resulting in the host portion of the URL).

Sample

```
<entry>
  <string>au_start_url</string>
  <string>"%s%s/vol%s/index.html", base_url, journal_id, volume_name</string>
</entry>
```

Description
One or more URLs from which the crawl of an AU begins.

5.3.2 Crawl Seed

Note: This page is under construction.

Plugin Key
`plugin_craw_seed_factory`

Plugin Value Type
String

The string is the fully-qualified name of a Java class implementing the `org.lockss.crawler.CrawlSeedFactory` interface.

Sample

```
<entry>
  <string>plugin_craw_seed_factory</string>
  <string>edu.example.plugin.publisherx.PublisherXCrawlSeedFactory</string>
</entry>
```

Description
In lieu of a list of start URLs, code called a crawl seed can compute the starting points of the crawl of an AU, for instance by interacting with an API.

5.3.3 Permission URLs

Plugin Key
`au_permission_url`

Plugin Value Type
One of:

- *String*
- *List* of *String*

Plugin Value Format
The strings are `printf` format strings, that expand to URLs. The `printf` format strings accept expressions made of plugin configuration parameter keys and a small language of functions modifying them (e.g. `url_host(...)` applied to a plugin configuration parameter of type URL, resulting in the host portion of the URL).

Sample

```
<entry>
  <string>au_permission_url</string>
  <string>"%slockss.txt", base_url</string>
</entry>
```

Description
One or more URLs giving the LOCKSS software permission to crawl an AU, if permission is not given on the start URLs.

5.3.4 Per-Host Permission Path

Plugin Key
`plugin_per_host_permission_path`

Plugin Value Type
String

Sample

```
<entry>
  <string>plugin_per_host_permission_path</string>
  <string>/lockss.txt</string>
</entry>
```

Description
Relative path where a permission statement may be found on hosts not listed in start URLs or permission URLs.

Useful for sites that have banks of similar hosts with unpredictable names, but with a predictable path to the permission URL on each.

5.3.5 Permitted Host Pattern

Note: This page is under construction.

Plugin Key
`au_permitted_host_pattern`

Plugin Value Type
One of:

- *String*
- *List* of *String*

Plugin Value Format
The strings are `printf` format strings, that expand to regular expressions used to match against host names. The `printf` format strings accept expressions made of plugin configuration parameter keys and a small language of functions modifying them (e.g. `url_host(...)` applied to a plugin configuration parameter of type URL, resulting in the host portion of the URL).

Example

```
<entry>
  <string>au_permitted_host_pattern</string>
  <string>"cdnjs\.cloudflare\.com|fast\.fonts\.net"</string>
</entry>
```

Description
Pattern rules to allow collection from hosts that cannot explicitly grant permission, such as content distribution network hosts used to distribute standard components used by Web sites like Javascript libraries and Web fonts.

5.3.6 Crawl Rules

Plugin Key
`au_crawlrules`

Plugin Value Type
List of *String*

Plugin Value Format
The strings consist of:

- An integer crawl rule code,
- A comma,
- A `printf` format string that expands into a regular expression used to match against URLs. The `printf` format string accepts expressions made of plugin configuration parameter keys and a small language of functions modifying them (e.g. `url_host(...)` applied to a plugin configuration parameter of type *URL*, resulting in the host portion of the URL).

Sample

```
<entry>
  <string>au_crawlrules</string>
  <list>
    <string>4, "^%s", base_url</string>
    <string>1, "^%s.*\.(css|js|gif|jpg|png)$", base_url</string>
    <string>2, "^%s%s/vol%s/iss[^/]+/art[^/]+/citedby", base_url, journal_id,␣
↪volume_name</string>
    <string>1, "^%s%s/vol%s/", base_url, journal_id, volume_name</string>
    <string>1, "^%spdf/.*\.pdf$", base_url</string>
  </list>
</entry>
```

Description

Sequential rules determining if a URL discovered during the crawl of an AU should in turn be fetched as part of the AU or not.

Given a URL, the crawler tries each crawl rule in the order of the list, until one of them produces an outcome for the URL. If none of the crawl rules result in an outcome for the URL, the default outcome is *Exclude* (the URL is excluded from the AU).

Crawl Rule Types

The crawl rule codes are:

Crawl Rule Code	Crawl Rule Type
1	*Include*
2	*Exclude*
3	*Include No Match*
4	*Exclude No Match*
5	*Include Match Else Exclude*
6	*Exclude Match Else Include*

Include

Crawl Rule Code

1

Description

If the URL matches the regular expression, include the URL in the AU; otherwise, this rule produces no outcome for the URL.

Exclude

Crawl Rule Code

2

Description

If the URL matches the regular expression, exclude the URL from the AU; otherwise, this rule produces no outcome for the URL.

Include No Match

Crawl Rule Code
3

Description
If the URL does not match the regular expression, include the URL in the AU; otherwise, this rule produces no outcome for the URL.

Exclude No Match

Crawl Rule Code
4

Description
If the URL does not match the regular expression, exclude the URL from the AU; otherwise, this rule produces no outcome for the URL.

Include Match Else Exclude

Crawl Rule Code
5

Description
If the URL matches the regular expression, include the URL in the AU; otherwise, exclude the URL from the AU.

Exclude Match Else Include

Crawl Rule Code
6

Description
If the URL matches the regular expression, exclude the URL from the AU; otherwise, include the URL in the AU.

5.3.7 Crawl Window

Note: This page is under construction.

Plugin Key
`au_crawlwindow`, `au_crawlwindow_ser`

Plugin Value Type
`au_crawlwindow` value type: *String* representing the fully-qualified name of a Java class implementing the `org.lockss.plugin.definable.DefinableArchivalUnit.ConfigurableCrawlWindow` interface, which is a factory for the `org.lockss.daemon.CrawlWindow` interface.

`au_crawlwindow_ser` value type: a serialized `org.lockss.daemon.CrawlWindow` object.

Description

A crawl window controls what times of day or days of the week crawls against the preservation target are allowed; by default an AU is eligible to crawl at any time.

5.3.8 Recrawl Interval

Plugin Key

`au_def_new_content_crawl`

Plugin Value Type

Long Integer

Sample

```
<entry>
  <string>au_def_new_content_crawl</string>
  <long>1209600000</long>
</entry>
```

Description

The amount of time (in milliseconds) before an AU that has previously been crawled successfully is eligible to attempt crawling again.

Sample values:

- `31449600000` for 52 weeks (364 days).
- `1209600000` for 2 weeks (the most typical value in the Global LOCKSS Network).
- `604800000` for 1 week.
- `86400000` for 24 hours.

5.3.9 Refetch Depth

Note: This page is under construction.

Plugin Key

`au_refetch_depth`

Plugin Value Type

Integer

Sample

```
<entry>
  <string>au_refetch_depth</string>
  <int>2</int>
</entry>
```

Description

Number of links away from a start URL that will be fetched by normal crawls. Deep crawls may be used to cause all URLs in an AU to be refetched (subject to `If-Modified-Since`).

5.3.10 Fetch Pause Time

Plugin Key
`au_def_pause_time`

Plugin Value Type
Long Integer

Sample

```
<entry>
  <string>au_def_pause_time</string>
  <long>3000</long>
</entry>
```

Description
The minimum amount of time (in milliseconds) between two fetches of consecutive URLs in the crawl of an AU.

The most common value in the Global LOCKSS Network is `3000` for no more frequently than every 3 seconds.

5.3.11 Crawl Rate Limiter

Note: This page is under construction.

Plugin Key
?

Plugin Value Type
?

Description
Fine-grained control of the maximum rate at which URLs may be fetched, based on media type, URL pattern, day of week or time of day.

5.3.12 Crawl Pool

Plugin Key
`plugin_fetch_rate_limiter_source`

Plugin Value Type
String

Description
A specification for how AUs are split into crawl pools. Only one new content crawl from a given crawl pool can happen at once. In the default crawl pool scheme, there is one crawl pool per plugin, but this plugin key can be used to define a new behavior.

The possible values are:

- `au`: Each AU has its own crawl pool.
- `plugin`: The pool name is the *Plugin Identifier*.
- `key:str`: The pool name is the string *str*.

- `host:`*`urlparamkey`*: The pool name is the string `host:` followed by the value of the AU's parameter named *`urlparamkey`*. If there is no such parameter, or if the parameter is not of type *URL*, the default crawl pool scheme applies.
- `title_attribute:`*`auattrkey`* and `title_attribute:`*`auattrkey`*`:`*`dflt`*: The pool name is the string `attr:` followed by the value of the AU's attribute named *`auattrkey`*. When the attribute is unset, if the longer form is used then use *`dflt`* as the value instead; otherwise the default crawl pool scheme applies.

5.3.13 Response Handler

Note: This page is under construction.

Plugin Key
`plugin_cache_result_list`

Plugin Value Type
List of *String*

The list represents a mapping; each string is of the form *`x`*=*`y`*, mapping from the left-hand side *`x`* which can be:

- An HTTP response code (integer).
- One of a finite set of Java exceptions (string).

to the right-hand side *`y`* which can be:

- The fully-qualified name of a Java class extending the `org.lockss.util.urlconn.CacheException` class.
- The fully-qualified name of a Java class implementing the `org.lockss.util.urlconn.CacheResultHandler` interface.

Sample

```
<entry>
  <string>plugin_cache_result_list</string>
  <list>
    <string>500=edu.example.plugin.publisherx.PublisherXHttpResponseHandler</string>
    <string>java.io.IOException=org.lockss.util.urlconn.CacheException
↪$RetryableNetworkException_3_30S</string>
  </list>
</entry>
```

5.3.14 URL Normalizer

Note: This page is under construction.

Plugin Key
`au_url_normalizer`

Plugin Value Type
String

Plugin Value Format

The value is the fully-qualified name of a Java class that implements the `org.lockss.plugin.UrlNormalizer` interface.

Sample

```
<entry>
  <string>au_url_normalizer</string>
  <string>edu.example.plugin.publisherx.PublisherXUrlNormalizer</string>
</entry>
```

Description

A URL normalizer maps URL variants to a canonical representation, for example by re-arranging equivalent URL query strings into a canonical order, removing extraneous URL substrings (for instance session IDs), canonicalizing the case (for instance to lowercase), etc. so that equivalent variants are stored only once under the canonical name.

5.3.15 Link Extractor

Note: This page is under construction.

Plugin Key

mediatype`_link_extractor_factory`, where *mediatype* is a media type like *text/html*

Plugin Value Type

String

Plugin Value Format

The value is the fully qualified name of a Java class implementing the `org.lockss.plugin.LinkExtractorFactory` interface.

Sample

```
<entry>
  <string>text/html_link_extractor_factory</string>
  <string>edu.example.plugin.publisherx.PublisherXHtmlLinkExtractorFactory</string>
</entry>
```

Description

The LOCKSS software comes with built-in code to extract URLs from HTML and CSS files encountered during the crawl of an AU. A URL extracted in this manner is then subject to the *URL Normalizer*, then the *Crawl Rules* determine if it should in turn be included in the AU. If URLs need to be extracted from other file types, or if the extraction behavior for built-in types like HTML and CSS needs to be extended or customized, this plugin feature can be used to point the plugin at new link extraction code.

5.3.16 Crawl Filter

Note: This page is under construction.

Plugin Key
*mediatype*_crawl_filter_factory, where *mediatype* is a media type like *text/html*

Plugin Value Type
String

Plugin Value Format
The value is the fully qualified name of a Java class implementing the `org.lockss.plugin.FilterFactory` interface.

Sample

```
<entry>
  <string>text/html_crawl_filter_factory</string>
  <string>edu.example.plugin.publisherx.PublisherXHtmlCrawlFilterFactory</string>
</entry>
```

Description
If files of a given media type need to be pre-processed (filtered) before URLs are extracted by the crawler using a *Link Extractor*, this plugin feature can be used to point at custom filtering code.

Crawl filters are somewhat related to *hash filters*.

5.3.17 URL Fetcher

Note: This page is under construction.

Plugin Key
`plugin_url_fetcher_factory`

Plugin Value Type
String

Plugin Value Format
The value is the fully qualified name of a Java class implementing the `org.lockss.plugin.UrlFetcherFactory` interface, which is a factory for the `org.lockss.plugin.UrlFetcher` interface.

Sample

```
<entry>
  <string>plugin_url_fetcher_factory</string>
  <string>edu.example.plugin.publisherx.PublisherXUrlFetcherFactory</string>
</entry>
```

Description
Under construction

5.3.18 URL Consumer

Note: This page is under construction.

Plugin Key
`plugin_url_consumer_factory`

Plugin Value Type
String

Plugin Value Format
The value is the fully qualified name of a Java class implementing the `org.lockss.plugin.UrlConsumerFactory` interface, which is a factory for the `org.lockss.plugin.UrlConsumer` interface.

Sample

```
<entry>
  <string>plugin_url_consumer_factory</string>
  <string>edu.example.plugin.publisherx.PublisherXUrlConsumerFactory</string>
</entry>
```

Description
Under construction

5.4 Crawl Validation

This section introduces features related to content validation in the context of a crawl.

5.4.1 Redirect to Login URL Pattern

Plugin Key
`au_redirect_to_login_url_pattern`

Plugin Value Type
String

Plugin Value Format
The string is a `printf` format string that expands to a regular expression used to match against URLs. The `printf` format string accepting expressions made of plugin configuration parameter keys.

Sample

```
<entry>
  <string>au_redirect_to_login_url_pattern</string>
  <string>"^%ssignup-login.*", base_url</string>
</entry>
```

Description
Determines whether an HTTP redirect returned by the crawled site is actually a redirect to a login page.

During the crawl of an AU, a redirect from a URL that is accepted by the crawl rules to a URL that is not is a non-fatal crawl error, meaning it gets reported at the end of the crawl rather than immediately stopping it.

If a Web site issues an HTTP redirect to a login page URL when an IP address accesses a URL it is not authorized to view, it is often desirable to recognize this situation differently than other redirects to URLs outside the crawl rules. This key is used to identify such redirects and treat them as fatal crawl errors instead of non-fatal crawl errors.

5.4.2 Login Page Checker

Note: This page is under construction.

Plugin Key
`au_login_page_checker`

Plugin Value Type
String

Plugin Value Format
The value is the fully qualified name of a Java class implementing the `org.lockss.daemon.LoginPageChecker` interface.

Sample

```
<entry>
  <string>au_login_page_checker</string>
  <string>edu.example.plugin.publisherx.PublisherXLoginPageChecker</string>
</entry>
```

Description
When fetching a URL that the requesting IP address is not authorized to retrieve, some Web sites respond with HTTP success code accompanied by an HTML page that is really a login page or error page, rather than with an HTTP error code. A login page checker is a general interface for checking the contents of a URL and determine if it is a login page rather than the intended content.

5.4.3 Content Validator

Note: This page is under construction.

Plugin Key
mediatype`_content_validator_factory`, where *mediatype* is a media type like *application/pdf*

Plugin Value Type
String

Plugin Value Format
The value is the fully qualified name of a Java class implementing the `org.lockss.plugin.ContentValidatorFactory` interface, which is a factory for the `org.lockss.plugin.ContentValidator` interface.

Sample

```
<entry>
  <string>application/pdf_content_validator_factory</string>
```

(continues on next page)

(continued from previous page)

```
  <string>edu.example.plugin.publisherx.PublisherXPdfValidatorFactory</string>
</entry>
```

Description

A content validator can be used to reject files of a given media type, by examining the headers and/or the content of a crawled URL.

5.4.4 Substance Patterns

Note: This page is under construction.

Plugin Key

`au_substance_url_pattern` and `au_non_substance_url_pattern`

Plugin Value Type

String

Plugin Value Type

The strings are `printf` format strings that expand to regular expressions used to match against URL. The `printf` format strings accepting expressions made of plugin configuration parameter keys.

Sample

```
<entry>
  <string>au_substance_url_pattern</string>
  <string>"^%s/%s/(fulltext|pdf)/", base_url, journal_id</string>
</entry>
```

Description

It can be useful to test whether at least one URL in an AU is "of substance", meaning that it is one of the objects of preservation interest and not simply navigation pages and ancillary files (icons, CSS stylesheets, Javascript code). The substance pattern can be used to characterize the URLs of an AU that are of substance. Conversely, the non-substance pattern can be used to characterize the URLs of an AU that are not of substance.

5.4.5 Substance Predicate

Note: This page is under construction.

Plugin Key

`plugin_substance_predicate_factory`

Plugin Value Type

String

Plugin Value Type

The value is the fully qualified name of a Java class implementing the `org.lockss.plugin.SubstancePredicateFactory` interface, which is a factory for the `org.lockss.plugin.SubstancePredicate` interface.

Sample

```
<entry>
  <string>plugin_substance_predicate_factory</string>
  <string>edu.example.plugin.publisherx.PublisherXSubstancePredicateFactory</string>
</entry>
```

Description
An alternative to the regular expressions in the *Substance Patterns* is a substance predicate, which is implemented as Java code.

5.5 Poll Control

This section introduces plugin features that influence the operation of the LOCKSS audit and repair protocol.

5.5.1 Exclude URLs From Polls Pattern

Plugin Key
`au_exclude_urls_from_polls_pattern`

Plugin Value Type
One of:

- *String*
- *List* of *String*

Plugin Value Type
The strings are `printf` format strings that expand to regular expressions used to match against URLs. The `printf` format strings accept expressions made of plugin configuration parameter keys.

Sample

```
<entry>
  <string>au_exclude_urls_from_polls_pattern</string>
  <list>
    <string>"^%scss/.*\.css\?version=", base_url</string>
    <string>"^%sfiles/[0-9]+/.*\.js", base_url</string>
  </list>
</entry>
```

Description
URLs characterized by the regular expressions expanded from the `printf` strings are excluded (ignored) during polls of the AU.

5.5.2 Poll Result Weight

Plugin Key
`au_url_poll_result_weight`

Plugin Value Type

- *List* of *String*

Plugin Value Format
Each string in the list consists of:

- A regular expression,
- A comma,
- A weight between 0.0 and 1.0.

Sample

```
<entry>
  <string>au_url_poll_result_weight</string>
  <list>
    <string>\.(css|js|png|jpe?g|png|gif|tiff)\?ver=.*$, 0</string>
  </list>
</entry>
```

Description
This mechanism is a generalization of *Exclude URLs From Polls Pattern*, which can only exclude URLs from a poll (equivalent of assigning the corresponding URLs a weight of 0.0). Here, URLs can be assigned a partial weight, to reduce the importance of certain families of URLs in polls.

5.5.3 Repair From Publisher When Too Close

Note: This page is under construction.

Plugin Key
`plugin_repair_from_publisher_when_too_close`

Plugin Value Type
?

Description
Under construction

5.5.4 Repair From Peer If Missing

Note: This page is under construction.

Plugin Key
`au_repair_from_peer_if_missing_url_pattern`

Plugin Value Type
One of:

- *String*
- *List* of *String*

Plugin Value Format

The strings are `printf` format strings that expand to regular expressions used to match against URLs. The `printf` format strings accept expressions made of plugin configuration parameter keys.

Sample

```
<entry>
  <string>au_repair_from_peer_if_missing_url_pattern</string>
  <list>
    <string>"^%s.*\.(css|js|png|jpe?g|png|gif|tiff)\?ver=.*$", base_url</string>
    <string>"^https://code\.jquery\.org/"</string>
  </list>
</entry>
```

Description

When this feature is set, if during a poll the poller determines it is missing a URL that matches one of the patterns, it will seek a repair from a peer. This can be useful when a family of URLs keeps getting new variants (with unique names) rapidly, to allow the variants to spread among the network of peers.

5.6 Hash Filtering

This section introduces plugin features related to content canonicalization for inter-node comparison purposes.

5.6.1 Hash Filter

Plugin Key

mediatype`_filter_factory`, where *mediatype* is a media type like *text/html*

Value Type

String

Plugin Value Type

The string is the fully-qualified name of a Java class implementing the `org.lockss.plugin.FilterFactory` interface.

Sample

```
<entry>
  <string>text/html_filter_factory</string>
  <string>edu.example.plugin.publisherx.PublisherXHtmlHashFilterFactory</string>
</entry>
```

Description

To canonicalize content before comparison between nodes in the LOCKSS audit and repair protocol, a plugin can define a hash filter for each affected media type. The goal is to pre-process content so that it is fit for a **logical comparison** between nodes, even if different nodes do not have byte-identical versions. This occurs frequently in HTML content that has personalizations ("You are logged in as..."), advertising, and other variable content ("You may also be interested in...", "Top 10 viewed articles this week...", "Recently added articles...") other than the main content. It can be needed for other media types like PDF and RIS because of timestamping, watermarking, and other dynamic server behaviors.

The `org.lockss.plugin.FilterFactory` interface defines a `createFilteredInputStream` method that accepts an `org.lockss.plugin.ArchivalUnit` object, an `InputStream` of the URL's raw content, and a string representing the encoding, and returns an `InputStream` of the canonicalized byte stream, which does not need to be a valid object of that media type (it is only used to compute a checksum).

As part of its general content filtering framework, the LOCKSS plugin framework offers a variety of utility classes specifically for *HTML Filters* and *PDF Filters*.

5.6.2 HTML Filters

Note: This page is under construction.

HTML is the media type most frequently in need of transformation and canonicalization, as part of a *Hash Filter* or *Crawl Filter*. To this end, the LOCKSS software contains a variety of utility classes that can be used as building blocks to construct effective HTML filters.

`HtmlFilterInputStream`

The `org.lockss.filter.html.HtmlFilterInputStream` class is a way to parse an `InputStream` of HTML content and apply a transform of type `org.lockss.filter.html.HtmlNodeFilterTransform` to it, resulting in a new `InputStream`.

The `org.lockss.filter.html.HtmlNodeFilterTransform` class provides two kinds of transforms, that either filter out all HTML nodes that match a predicate ("exclude transform", very common use pattern) or collect only HTML nodes that match a predicate ("include transform", less common use pattern). The code of the library used to implement `org.lockss.filter.html.HtmlFilterInputStream` and `org.lockss.filter.html.HtmlNodeFilterTransform` refers to these predicates as `org.htmlparser.NodeFilter`.

Whether an "exclude" transform or an "include" transform, most often the predicate is really the union of many smaller predicates, which can be grouped together by the `org.htmlparser.filters.OrFilter` class (and other similar boolean operators).

Although some complex cases call for the definition of a custom predicate, most typical situations can be handled by predicates predefined in the `org.lockss.filter.html.HtmlNodeFilters` utility class:

- `org.lockss.filter.html.HtmlNodeFilters.tag(String tag)`: predicate that matches a tag with the given name. (Many LOCKSS plugins make direct use of the underlying `org.htmlparser.filters.TagNameFilter.TagNameFilter(String)` class, but using the utility method is recommended.)
- `org.lockss.filter.html.HtmlNodeFilters.tagWithAttribute(String tag, String attr)`: predicate that matches a tag with the given name, that defines an attribute with the specified key (regardless of the value).
- `org.lockss.filter.html.HtmlNodeFilters.tagWithAttribute(String tag, String attr, String val)`: predicate that matches a tag with the given name, that defines an attribute with the specified key and value. The variant `org.lockss.filter.html.HtmlNodeFilters.divWithAttribute(String attr, String val)` assumes a tag.
- `org.lockss.filter.html.HtmlNodeFilters.tagWithAttributeRegex(String tag, String attr, String regex)`: predicate that matches a tag with the given name, that defines an attribute with the specified key, whose value matches the given regular expression. The variant `org.lockss.filter.html.HtmlNodeFilters.tagWithAttributeRegex(String tag, String attr, String regex, boolean ignoreCase)` adds a flag for whether the regular expression is case-insensitive.
- `org.lockss.filter.html.HtmlNodeFilters.comment()`: predicate that matches any HTML comment.

- `org.lockss.filter.html.HtmlNodeFilters.commentWithString(String str)`: predicate that matches HTML comments containing the given string. The variant `org.lockss.filter.html.HtmlNodeFilters.commentWithString(String str, boolean ignoreCase)` adds a flag for whether the test is case-insensitive.
- `org.lockss.filter.html.HtmlNodeFilters.commentWithRegex(String regex)`: predicate that matches HTML comments matching the given regular expression. The variant `org.lockss.filter.html.HtmlNodeFilters.commentWithRegex(String regex, boolean ignoreCase)` adds a flag for whether the regular expression is case-insensitive.

WhiteSpaceFilter

The `org.lockss.filter.WhiteSpaceFilter` class is a `Reader` implementation that canonicalizes whitespace by collapsing consecutive whitespace characters to a single one. This is useful because many transformations may leave different numbers of consecutive whitespace characters (perhaps zero) depending on the whitespace surrounding the outermost tags of various removed nodes.

In many LOCKSS plugins, the typical way to use `WhiteSpaceFilter` is to turn an `InputStream` into a `Reader`, apply `WhiteSpaceReader`, and turn the result back into an `InputStream` for further processing:

```
InputStream i1 = ...; // an input stream
String e1 = ...; // the encoding of the input stream
Reader r1 = FilterUtil.getReader(i1, e1);
Reader r2 = new WhiteSpaceFilter(r1);
InputStream i2 = new ReaderInputStream(r2);
```

5.6.3 PDF Filters

Note: This page is under construction.

Increasingly, some content providers generate PDF files dynamically, making each fetch of the same URL slightly different, just like HTML pages in many cases. To allow the nodes in a LOCKSS network to canonicalize PDF files for comparison purposes, the LOCKSS software contains a PDF processing and filtering framework that can be used as building blocks to construct PDF filters[1].

The interface of this framework and general tools are in the `org.lockss.pdf` package, with an implementation based on Apache PDFBox 1.8 in the `org.lockss.pdf.pdfbox` package.

Under construction.

[1] Additionally, there is legacy code based on (pre-Apache) PDFBox 0.7.3 which is **deprecated**, and an experimental library based on Apache PDFBox 2.0 which is in active development.

5.7 Metadata Extraction

This section introduces plugin features related to the extraction and interpretation of metadata from preserved content.

5.7.1 Introduction to Metadata Extraction

Once content is successfully crawled into an archival unit (AU) in a LOCKSS node, optionally with the help of hash filters and related plugin features, the AU is preserved by polling and repairing with other nodes in the network holding the same AU. If metadata extraction from preserved data is desired beyond the preservation of the data itself and the **metadata database** is enabled, the plugin needs to specify **metadata extraction** features.

Metadata extraction relies on a trio of related concepts:

- An *Article Iterator* groups an AU's URLs into one cluster per article ("article" in the sense of "object" or "item").

  The URLs are represented internally as `org.lockss.plugin.CachedUrl` objects. The object representing an article's cluster of URLs is of type `org.lockss.plugin.ArticleFiles`, and is essentially a mapping from string **roles** to URLs.

  An article iterator is merely an object implementing `java.util.Iterator<ArticleFiles>`, that comes from a factory implementing the `org.lockss.plugin.ArticleIteratorFactory` interface.

- A media type-specific *File Metadata Extractor* parses the contents of a URL and emits any number of **intermediate metadata records**.

  A file metadata extractor is an object implementing the `org.lockss.extractor.FileMetadataExtractor` interface and emitting metadata records of type `org.lockss.extractor.ArticleMetadata` through an object implementing the `org.lockss.extractor.FileMetadataExtractor.Emitter` interface. The latter is called for each (`CachedUrl`, `ArticleFiles`) pair, creating a one-to-many relationship from `CachedUrl` to `ArticleFiles`.

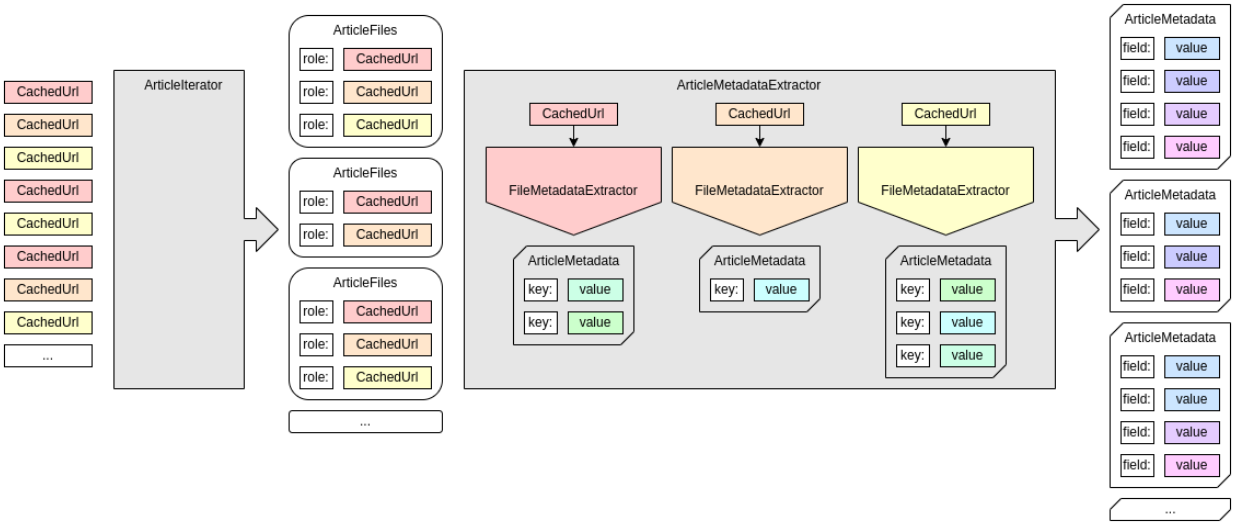
- An *Article Metadata Extractor* receives each article's `ArticleFiles` object, and emits any number of **processed metadata records** (of the same type `ArticleMetadata`).

  An article metadata extractor implements the `org.lockss.extractor.ArticleMetadataExtractor` interface and emits `ArticleMetadata` objects through an object implementing the `org.lockss.extractor.ArticleMetadataExtractor.Emitter` interface. The latter is called for each (`ArticleFiles`, `ArticleMetadata`) pair, creating a one-to-many relationship from `ArticleFiles` to `ArticleMetadata`.

  The article metadata extractor picks and chooses URLs of interest from the `ArticleFiles` instance, invokes the file metadata extractors for the corresponding media types yielding intermediate `ArticleMetadata` objects, and emits appropriate final `ArticleMetadata` objects from them.

Although in principle there are file metadata extractors for multiple media types, a one-to-many relationship from `CachedUrl` to `ArticleFiles` in file metadata extractors, and a one-to-many relationship from `ArticleFiles` to `ArticleMetadata` in article metadata extractors, in many situations plugins derive all the metadata they need from a single media type, there is a one-to-one-to-one correspondence between a `CachedUrl`, `ArticleFiles` and `ArticleMetadata` triple, and the intermediate metadata records can often be emitted as final metadata records.

This process can be summarized in the following diagram:



5.7.2 Article Iterator

Plugin Key

`plugin_article_iterator_factory`

Plugin Value Type

String

Plugin Value Type

The string is the fully-qualified name of a Java class implementing the *org.lockss.plugin.ArticleIteratorFactory* interface.

Sample

```
<entry>
  <string>plugin_article_iterator_factory</string>
  <string>edu.example.plugin.publisherx.PublisherXArticleIteratorFactory</string>
</entry>
```

Description

The article iterator is part of the *metadata-extraction* pipeline. Its function is enumerate the articles (where "article" is meant as "item" or "object") in the archival unit (AU). Each article is represented by an *ArticleFiles* instance.

Rather than traversing the AU's URLs manually through the `ArchivalUnit` interface and implementing typical inner workings of a Java `Iterator`, many article iterators make use of utility classes available in the LOCKSS software, such as `SubTreeArticleIterator` and `SubTreeArticleIteratorBuilder`.

ArticleFiles

An `org.lockss.plugin.ArticleFiles` instance groups the main URLs of an article (item) together and labels them. It is a mapping from **roles** to URLs[1] (which are internally represented as objects implementing `org.lockss.plugin.CachedUrl`).

Roles are arbitrary strings, but many typical role strings are defined as constants in the `ArticleFiles` class, for example:

- `ArticleFiles.ROLE_FULL_TEXT_HTML`: a URL for the full text of a work, in HTML form.
- `ArticleFiles.ROLE_FULL_TEXT_PDF`: a URL for the full text of a work, in PDF form.
- `ArticleFiles.ROLE_FULL_TEXT_EPUB`: a URL for the full text of a work, in EPUB form.
- `ArticleFiles.ROLE_FULL_TEXT_XML`: a URL for the full text of a work, in XML form.
- `ArticleFiles.ROLE_ABSTRACT`: a URL for a work's abstract.
- `ArticleFiles.ROLE_REFERENCES`: a URL for a work's list of works cited.
- `ArticleFiles.ROLE_FIGURES`: a URL for a landing page of the work's figures and illustrations.
- `ArticleFiles.ROLE_TABLES`: a URL for a landing page of the work's tables.
- `ArticleFiles.ROLE_SUPPLEMENTARY_MATERIALS`: a URL for a landing page of the work's supplementary materials.
- `ArticleFiles.ROLE_CITATION`: a URL for a landing page or the work's citation files.
- `ArticleFiles.ROLE_CITATION_BIBTEX`: a URL for a BibTeX citation file for the work.
- `ArticleFiles.ROLE_CITATION_ENDNOTE`: a URL for an EndNote citation file for the work.
- `ArticleFiles.ROLE_CITATION_RIS`: a URL for a RIS citation file for the work.
- `ArticleFiles.ARTICLE_METADATA`: a URL from which metadata for the work can be found.

In addition to the mapping from roles to URL, one URL has a special status in the `ArticleFiles` instance as the designated, "best" full text URL for the work. It is referred to as the **full text URL** or **full text CU** (for `CachedUrl`) of the article, and is set via the `setFullTextCu(...)` method. In plugins written by the LOCKSS Program, for articles with multiple full text representations, HTML is favored above all, then PDF, then EPUB, and lastly XML (in subjective order of richness of rendering experience in a Web browser).

SubTreeArticleIterator

The `org.lockss.plugin.SubTreeArticleIterator` class implements `Iterator<ArticleFiles>` and can be returned by an article iterator factory. It traverses an AU's URLs, restricting them in various ways according to a specification. These restrictions include considering only certain subdirectory trees (hence the name), applying regular expressions, selecting a media type, or applying a custom condition.

The `SubTreeArticleIterator.Spec` class contains the specification:

- The specification identifies **root URLs**, limiting which AU URLs are enumerated to those under these root URLs only. This restriction applies to the directory structure, not the URL strings -- in effect, root URLs end with a slash. Root URLs can be specified directly with `setRoot(...)` or `setRoots(...)`, or via `printf` templates (which expand to URLs) expressed as single Java strings with `setRootTemplate(...)` or `setRootTemplates(...)`. The `printf` templates are expressed as single Java strings, which can be tricky to read, for example: `"\"%s%s/%d/\", base_url, journal_id, year"` (with the convention that the base URL value ends with a slash). If the specification specifies no root URL, all URLs in the AU are enumerated.

[1] Technically a mapping from roles to arbitrary Java objects.

- The specification can optionally have an **include pattern** or an **exclude pattern**. With an include pattern, URLs below each root URL that match the include pattern are considered (but others are ignored). With an exclude pattern, URLs below each root URL that match the exclude pattern are ignored (but others are considered). No error will happen if both are specified, but the exclude pattern will be ignored in favor of the include pattern. These patterns (regular expressions) can be specified directly with `setIncludeSubTreePattern(...)` or `setExcludeSubTreePattern(...)`, or via `printf` templates (which expand to regular expressions) expressed as single Java strings with `setIncludeSubTreePatternTemplate(...)` or `setExcludeSubTreePatternTemplate(...)`.
- The specification can optionally have a **general pattern**, which is a regular expression applied to any URL still under consideration. It can be specified directly with `setPattern(...)`, or via a `printf` template (which expands to a regular expression) expressed as a single Java string with `setPatternTemplate(...)`.
- The specification can have an optional media type (for example `text/html`), and only URLs under consideration that have this media type will be enumerated. There is currently no way to specify multiple media types, which can be problematic for media types with multiple common representations, like `text/xml` and `application/xml`. The class could be enhanced in the future to allow multiple media types.

The logic for which URLs are enumerated is found in the `isArticleCu(...)` method, which can be customized in a subclass.

By default, for each successful URL, this iterator makes one `ArticleFiles` instance that has its designated full text URL set, and no roles set. This behavior can be customized in `visitArticleCu(...)` and `createArticleFiles(...)` in a subclass.

SubTreeArticleIteratorBuilder

The `org.lockss.plugin.SubTreeArticleIteratorBuilder` class assists in the creation of a `SubTreeArticleIterator` instance under circumstances where the URLs of various aspects of an article (for example its abstract URL, its full text HTML URL, its full text PDF URL, etc.) can all be derived from one another through **mutually compatible regular expressions** and replacement strings.

An example of such mutual compatibility would be a journal where articles have full text HTML URLs that look like this: `http://www.example.com/vol12/iss3/art45` and full text PDF URLs that look like this: `http://www.example.com/pdf/article_12_3_45.pdf` (assuming these URLs represent volume 12, issue 3, page 45). A regular expression for the full text HTML URLs (expressed as a Java string) could be `"/vol(\\d+)/iss(\\d+)/art(\\d+)$"`, and the replacement string `"/pdf/article_$1_$2_$3.pdf"` would yield the corresponding full text PDF URL from a match; likewise a regular expressin for the full text PDF URL could be `"/pdf/article_(\\d+)_(\\d+)_(\\d+)\\.pdf$"`, and the replacement string `"/vol$1/iss$2/art$3"` would yield the corresponding full text HTML URL from a match.

The `SubTreeArticleIteratorBuilder` class has convenience methods to:

- Create a `SubTreeArticleIterator.Spec` specification. See `setSpec(...)`, or use `newSpec()` to manipulate an empty `Spec` from scratch.
- Define **major aspects** with one or more regular expressions matching the aspect's URLs, one or more replacement strings yielding the aspect's URLs from matchers for URLs of other major aspects, and one or more roles for the aspect.
- Define **minor aspects** with one or more replacement strings yielding the aspect's URLs from matchers for URLs of major aspects, and one or more roles for the aspect.

The key differences between major and minor aspects are:

- URLs enumerated by the `SubTreeArticleIterator` are tried against the regular expressions of the major aspects only.
- The earliest URL for a major aspect to match for a given article is also designated as the article's full text CU.

All aspects are defined with the variants of the `addAspect(...)` method and associated methods.

When an `ArticleFiles` is complete, methods like `setRoleFromOtherRoles(...)` and `setFullTextFromRoles(...)` can be used to designate additional roles or the full text CU from the value associated with an ordered list of possibilities from other roles.

The `getSubTreeArticleIterator()` method can finally be used to obtain a `SubTreeArticleIterator` instance behaving in the specified manner.

5.7.3 File Metadata Extractor

Plugin Key
mediatype>_metadata_extractor_factory_map, where *mediatype* is a media type like `text/html`

Plugin Value Type
Map from *String* to *String*

Plugin Value Type
The values are the fully-qualified name of a Java class implementing the `org.lockss.extractor.FileMetadataExtractorFactory` interface.

Sample

```
<entry>
  <string>text/html_metadata_extractor_factory_map</string>
  <map>
    <entry>
      <string>*</string>
      <string>edu.example.plugin.publisherx.PublisherXHtmlMetadataExtractorFactory</
↪string>
    </entry>
  </map>
</entry>
```

If the media type is represented under multiple guises in the plugin's AUs, for example XML represented as both `text/xml` and `application/xml`, you will need multiple entries in the plugin.

Description
File metadata extractors are part of the *metadata extraction* pipeline. Their function is to parse the contents of a particular URL based on its media type and file format, and emit any number of `ArticleMetadata` metadata records, and they are invoked as part of the execution of an *Article Metadata Extractor*.

SimpleFileMetadataExtractor

The `org.lockss.extractor.SimpleFileMetadataExtractor` utility class is used as a base class for the common case where a file metadata extractor produces a single metadata record (or `null`), rather than an arbitrary number of metadata records. It defines one abstract method:

```
public abstract ArticleMetadata extract(MetadataTarget target,
                                        CachedUrl cu)
    throws IOException, PluginException;
```

and its `extract(MetadataTarget target, CachedUrl cu, Emitter emitter)` method simply calls `extract(MetadataTarget target, CachedUrl cu)` and emits the returned `ArticleMetadata` if it is not `null`.

Utility classes based on `SimpleFileMetadataExtractor` include `JsoupTagExtractor` and `RisMetadataExtractor`.

JsoupTagExtractor

The `org.lockss.extractor.JsoupTagExtractor` utility class can be used to build **HTML** or **XML** file metadata extractors that use the jsoup parser.

By default, it maps the value of the `name` attribute of HTML `<meta>` tags to the value of their `content` attribute in the `ArticleMetadata` object's raw multi-map.

However if the media type is `text/xml`, `application/xml` or `application/xhtml+xml`, or if the extractor is created with selector strings, for each selector string, and for each element matched by the selector string, it maps the selector string to the selector value in the raw multi-map. The selector strings are those understood by the `select(...)` method of jsoup's `Document` class.

Subclasses provide the recipe multi-map (cook map) to process raw data into metadata.

Note: The `org.lockss.extractor.JsoupXmlTagExtractor` class exists but its functionality has been absorbed into `org.lockss.extractor.JsoupTagExtractor`, which is capable of handling HTML without selector strings as well as HTML and XML with selector strings. It may be removed in a future version of the LOCKSS system and should not be used for new plugin implementations -- use `JsoupTagExtractor` instead.

The `org.lockss.extractor.SimpleHtmlMetaTagMetadataExtractor` class also exists and scrapes HTML `<meta>` tags using a regular expression-based approach. It is at risk of being deprecated in a future version of the LOCKSS system, and is not recommended for new plugin implementations -- use `JsoupTagExtractor` instead.

RisMetadataExtractor

The `org.lockss.extractor.RisMetadataExtractor` utility class parses **RIS** metadata files (media type `application/x-research-info-systems`).

By default, it maps RIS tags to their values in the `ArticleMetadata` object's raw multi-map, and its recipe map (cook map) maps the following raw keys (RIS tags) to the following `MetadataField` instances:

- `T1` to the article title (`MetadataField.FIELD_ARTICLE_TITLE`)
- `AU` to an author (`MetadataField.FIELD_AUTHOR`)
- `JF` tp the journal title (`MetadataField.FIELD_PUBLICATION_TITLE`)
- `DO` to the DOI (`MetadataField.FIELD_DOI`)
- `PB` to the publisher name (`MetadataField.FIELD_PUBLISHER`)
- `VL` to the journal volume (`MetadataField.FIELD_VOLUME`)
- `IS` to the journal issue (`MetadataField.FIELD_ISSUE`)
- `SP` to the start page (`MetadataField.FIELD_START_PAGE`)
- `EP` to the end page (`MetadataField.FIELD_END_PAGE`)
- `DA` to the publication date (`MetadataField.FIELD_DATE`)

- SN to the ISSN (MetadataField.FIELD_ISSN) for a journal (TY tag equal to JOUR) or ISBN (MetadataField.FIELD_ISBN) for a book (TY tag equal to BOOK, CHAP, EBOOK, ECHAP, EDBOOK)

but the behavior is customizable.

SourceXmlMetadataExtractor

Because the LOCKSS Program processes large amounts of bulk content on behalf of the CLOCKSS Archive, which is often in the form of bundles of content with multi-article metadata in XML (for example JATS format), there are utility classes in the org.lockss.plugin.clockss package of the *plugins* tree of the lockss-daemon project to generalize this kind of data processing.

Plugins can only reference classes found in the plugin JAR itself, in lockss-core and in its dependencies (if using the re-architected LOCKSS system), or in the *main* tree of lockss-daemon and in its dependencies (if using the classic LOCKSS system), so these classes in the *plugins* tree of lockss-daemon are not directly accessible to arbitrary plugins (without some manipulation, like injecting additional classes in plugin JARs). However there is growing interest in re-using these utility classes in the broader LOCKSS community, so some of these classes will be "promoted" to lockss-core so they can be used by third-party plugins in a future version of the LOCKSS system.

The org.lockss.plugin.clockss.SourceXmlMetadataExtractorFactory, org.lockss.plugin.clockss.SourceXmlMetadataExtractorFactory.SourceXmlMetadataExtractor and org.lockss.plugin.clockss.SourceXmlSchemaHelper classes define a framework for processing XML metadata in some format, and mapping from XPath expressions to text values in the ArticleMetadata object's raw multi-map. The format-specific logic is confined in the SourceXmlSchemaHelper object.

The SourceXmlSchemaHelper class consists of a *global map* and an *article map*. Both map XPath strings to the corresponding values. The article map, aided by the getArticleNode() method which gives an XPath for the top-level node of each article in the XML file, is used to designate XPaths for each emitted article from the file. The optional global map is used to designate XPaths that apply to all emitted articles from the file, and can be used for XML formats that hoist some data above the level of each article (for instance publication-level or issue-level data).

This framework also offers some features to perform deduplication or recombination, verify some URLs or file paths, and SourceXmlSchemaHelper's getCookMap() method provides the recipe multi-map to produce metadata from the raw multi-map.

There is also an effort underway to define an equivalent framework for similarly structured metadata in JSON, using the Jayway JsonPath library.

5.7.4 Article Metadata Extractor

Plugin Key
plugin_article_metadata_extractor_factory

Plugin Value Type
String

Plugin Value Type
The string is the fully-qualified name of a Java class implementing the org.lockss.extractor.ArticleMetadataExtractorFactory interface.

Sample

```
<entry>
  <string>plugin_article_metadata_extractor_factory</string>
  <string>edu.example.plugin.publisherx.PublisherXArticleMetadataExtractorFactory</
↪string>
</entry>
```

Description

The article metadata extractor is part of the *metadata extraction* pipeline. Its function is to process each article (where "article" is meant as "item" or "object") in the archival unit (AU) and emit any number of metadata records from it. Each metadata record is represented by an `ArticleMetadata` instance. The article metadata extractor is aided in its task by *file metadata extractors*.

ArticleMetadata

An `org.lockss.extractor.ArticleMetadata` object contains two multi-maps (one-to-many mappings, using the Apache Commons Collections *org.apache.commons.collections4.map.MultiValueMap* ` class internally): a **raw multi-map**, and a **metadata multi-map** often called the **cooked multi-map** by analogy. The raw multi-map is for general-purpose storage of data extracted from content, where the keys are arbitrary strings, and the multiple values are either strings or maps from string to string. The cooked multi-map is the final representation of the metadata information contained in the object, where the keys are `MetadataField` instances, and the multiple values are strings.

`org.lockss.extractor.MetadataField` objects represent not only the key of a metadata field but also its cardinality (single or multiple) and a validator. Many are built into the `MetadataField` class itself, including single cardinality fields for journal volume, issue, start page and end page; single cardinality fields for DOI, ISSN, eISSN, ISBN that accept a string potentially prefixed with doi:, issn:, eissn: and isbn: (as is often found on publisher websites); multiple cardinality fields for authors; etc.

The plugin-dependent way to populate the cooked multi-map from the raw multi-map is given by a **recipe multi-map** sometimes confusingly referred to as the **cook map**, which maps raw keys (string) to one or more cooked keys (`MetadataField`). For each raw key-cooked key pair, each raw multi-value corresponding to the raw key is validated and stored by the cooked field into a cooked multi-value.

BaseArticleMetadataExtractor

Most plugins do not implement arbitrary logic in the article metadata extractor, but simply use the `org.lockss.extractor.BaseArticleMetadataExtractor` class.

This utility article metadata extractor is parameterized with a single target `ArticleFiles` role, parses the URL in the `ArticleFiles` that has the target role using the appropriate file metadata extractor for its media type, and post-processes each emitted `ArticleMetadata` object by adding bibliographic metadata drawn from the AU's listing in the title database (AU inventory) if such data is not extracted from the content already.

The logic for what gets post-processed from the title database into the `ArticleMetadata` instance is in the `addTdbDefaults(...)` method, which can be overridden to customize. If the AU is not labeled as bulk content (disparate content from many sources), the publication type, series name, ISSN, eISSN, ISBN, eISBN, publication name, volume, issue, and publication date are set from the corresponding value in the title database, if available and if not already set from the actual metadata extraction.

Additionally, the `checkAccessUrl(...)` method ensures that the URL set under the `MetadataField.FIELD_ACCESS_URL` metadata key for the article is actually in the AU, and if not, it is reset to the full text URL for the article.

The implementation of `ArticleMetadataExtractorFactory` is often simply to return a new `BaseArticleMetadataExtractor` for a given target role (usually `ArticleFiles.ARTICLE_METADATA`), with no further code:

```
@Override
public ArticleMetadataExtractor createArticleMetadataExtractor(MetadataTarget target)
    throws PluginException {
  return new BaseArticleMetadataExtractor(ArticleFiles.ROLE_ARTICLE_METADATA);
}
```

Because of this, many plugins do not define a separate Java class for the article metadata extractor factory but simply let their article iterator factory also implement `ArticleMetadataExtractorFactory`:

```
public class PublisherXArticleIteratorFactory
    implements ArticleIteratorFactory, ArticleMetadataExtractorFactory {

  @Override
  public Iterator<ArticleFiles> createArticleIterator(ArchivalUnit au,
                                                      MetadataTarget target)
      throws PluginException {
    // ...
  }

  @Override
  public ArticleMetadataExtractor createArticleMetadataExtractor(MetadataTarget target)
      throws PluginException {
    return new BaseArticleMetadataExtractor(ArticleFiles.ROLE_ARTICLE_METADATA);
  }

}
```

5.8 Web Replay

This section introduces plugin features related to supporting the replay of Web content.

5.8.1 Link Rewriter

Note: This page is under construction.

Plugin Key
mediatype`_link_rewriter_factory`, where *mediatype* is a media type like `text/html`

Plugin Value Type
String

Plugin Value Type
The value is the fully qualified name of a Java class implementing the `org.lockss.extractor.LinkRewriterFactory` interface.

Sample

```
<entry>
  <string>text/html_link_rewriter_factory</string>
  <string>edu.example.plugin.publisherx.PublisherXHtmlLinkRewriterFactory</string>
</entry>
```

Description
When content is replayed through the LOCKSS system's ServeContent Web replay engine, links have to be rewritten so that they point to other ServeContent URLs where applicable. ServeContent contains logic to handle typical cases in HTML and CSS, but some specific use cases may require additional or custom link rewriting. To accomplish this, the plugin defines link rewriters for the affected media types.

For example, a Web site could have image tags for journal article figures that look like this: `<img src="fig1_small.jpg" data-target="fig1_large.jpg" />`, and Javascript code in the page such that when the small version of the image is clicked, an image viewer widget is displayed with the large version of the image instead. ServeContent has internal logic that knows to look for the `src` attribute of `<img>` tags, but would not know to also process this non-standard `data-target` attribute so the image viewer widget works with a preserved copy of the large version of the image. Depending on the situation, this might require a custom rewriter for just HTML, or for HTML plus Javascript.

5.8.2 Rewrite HTML Meta URLs

Note: This page is under construction.

Plugin Key
`plugin_rewrite_html_meta_urls`

Plugin Value Type
List of *String*

Plugin Value Type
Each string in the list is a value of the `name` attribute of HTML `<meta>` tags.

Sample

```
<entry>
  <string>plugin_rewrite_html_meta_urls</string>
  <list>
    <string>citation_abstract_url</string>
    <string>citation_pdf_url</string>
  </list>
</entry>
```

Description
This plugin feature enables a canned HTML *Link Rewriter* that seeks out `<meta name="..." content="...">` tags where the value of the `name` attribute matches one of the specified names, and rewrites the URL that is the value of their `content` attribute.

5.9 Inheritance

This section introduces plugin features related to sharing similar behavior among a set of plugins.

5.9.1 Parent Plugin

Plugin Key
`plugin_parent`

Plugin Value Type
String

Plugin Value Format
The value is the *Plugin Identifier* of this plugin's parent.

Sample

```
<entry>
  <string>plugin_parent</string>
  <string>edu.example.plugin.platformx.PlatformXPlugin</string>
</entry>
```

Description

Declares that this plugin uses the specified plugin as its parent. This plugin inherits the key-value pairs from the parent plugin, and additionally adds values for keys not found in the parent or redefines the value for a key found in the parent.

If the parent plugin maps a key to a specific value and this plugin wishes to undo the effect and simply use whatever the default is for the key in the system, this plugin can map the key to the special value `<org.lockss.util.Default />`.

5.9.2 Parent Plugin Version

Plugin Key

`plugin_parent_version`

Plugin Value Type

String, *not* integer

Plugin Value Format

The value is the *Parent Plugin*'s *Plugin Version*.

Sample

```
<entry>
  <string>plugin_parent_version</string>
  <string>2</string>
</entry>
```

Description

Declares the intended version of this plugin's *Parent Plugin*.

5.10 Appendix

This appendix gives a brief overview of *printf Format Strings* and *Regular Expressions*.

5.10.1 `printf` Format Strings

Note: This page is under construction.

A `printf` format string is a template for generating a string of text from variable parts. It consists of a **format specification** containing literal elements and **format specifiers** (placeholders with formatting hints), and a list of **expressions** for each placeholder's value.

A `printf` interpreter is a template engine that accepts a `printf` format string as input and produces an output string from it, based on its implementation-dependent capabilities to compute the value of expressions and on context-dependent variables.

The LOCKSS software contains a `printf` interpreter where the expressions are names of *Plugin Configuration Parameters* and AU attributes, optionally modified by a small number of processing functions.

`printf` Format String Format

A `printf` format specification starts and ends with a quotation mark (") and follows the `printf` specification syntax.

For each format specifier in the format specification, the value of an **expression** is computed and substituted for the format specifier. The successive expressions correspond to the format specifiers in the order they appear in the format specification.

The format specification and the expressions are comma-separated, with whitespace surrounding the elements trimmed.

Anything that is not recognized as a format specifier is interepreted to be a **literal** part of the output string.

`printf` Format Specifiers

Format specifiers begin with a **percent sign** (`%`) and end with a **type field**, separated by optional characters further constraining the formatting of the placeholder's value. The most important type fields are:

- `%s`: for **string**-valued expressions.
- `%d`: for **integer**-valued expressions.
- `%%` for a **literal percent sign**.

String

The most common format specifier is `%s` for string-valued expressions. Example:

Input:

```
"The base URL is %s", base_url
```

Output in a context where `base_url` is `http://www.example.com/`:

```
The base URL is http://www.example.com/
```

Integer

The format specifier `%d` is used for an integer-valued expression. Example:

Input:

```
"The year is %d", year
```

Output in a context where `year` is the integer `2022`:

```
The year is 2022
```

Percent Sign

Because the percent sign is used to introduce format specifiers, there is type field to indicate that a placeholder is for a literal percent sign, and that type field is also the percent sign. Examples:

Input:

```
"100%% of the time"
```

Output:

```
100% of the time
```

Input:

```
"100% of the time"
```

Output: error

References

- printf format string on Wikipedia

5.10.2 Regular Expressions

Note: This page is under construction.

A **regular expression** is a character string that specifies a **text search pattern**. The search pattern can then be applied to an input string. If a portion of the input string **matches** the search pattern, the matching portions or designated subparts of it can be extracted.

A **regular expression engine** is a software library capable of applying search patterns specified in an implementation-dependent regular expression language to arbitrary input strings.

The LOCKSS software uses the Java regular expression engine, but because Java did not always have built-in regular expression support, in some contexts the LOCKSS software uses the older Apache ORO regular expression engine. Typical use of regular expressions in LOCKSS plugins should not result in noticeable differences between the two engines' capabilities.

References

- Regular expression on Wikipedia

CHAPTER

SIX

LOCKSS NETWORK ADMINISTRATOR GUIDE

Note: The LOCKSS Network Administrator Guide is under construction.

Welcome to the LOCKSS Network Administrator Guide. This section of the LOCKSS Documentation Portal contains information aimed at administrators of LOCKSS networks.

INDEX