

IBM Cúram Social Program Management  
Version 7.0.10

*Data extractor*



**Note**

Before using this information and the product it supports, read the information in [“Notices” on page 95](#)

**Edition**

This edition applies to IBM® Cúram Social Program Management v7.0.10 and to all subsequent releases unless otherwise indicated in new editions.

Licensed Materials - Property of IBM.

© **Copyright International Business Machines Corporation 2019, 2020.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

- Figures..... V**
- Tables..... vi**
- Chapter 1. IBM Social Program Management Data Extractor..... 1**
- Chapter 2. Determining case eligibility and entitlement..... 2**
- Chapter 3. Components of the IBM Social Program Management Data Extractor..... 5**
- Chapter 4. Downloading and installing the IBM Social Program Management Data Extractor..... 8**
- Chapter 5. Batch jobs.....10**
  - Batch jobs: job types and their purposes.....10
    - How determinations are read by the IBM Social Program Management Data Extractor batch jobs..... 12
  - XML Flow job..... 13
  - Filter Flow CSV job ..... 17
    - The output of the Filter Flow CSV job..... 17
    - The output directory for the Filter Flow CSV job..... 26
  - Filter Flow database job..... 27
    - Filter Flow compaction limit..... 27
  - Monitoring job progress in the log ..... 30
  - Functions of the IBM Social Program Management Data Extractor that relate to the batch jobs .....31
    - Scheduling a job execution ..... 31
    - Restarting a job execution..... 31
    - Per determination retry..... 33
    - Stopping a job execution..... 34
    - Reporting a job execution status..... 34
    - Abandoning a job execution..... 34
- Chapter 6. Other functions of the IBM Social Program Management Data Extractor..... 36**
  - Searching for the Filter Flow attribute paths ..... 36
  - Generating the Filter Flow attribute value database tables creation and drop DDL..... 37
    - Issues associated with the length of a generated table name..... 39
  - Generating schema definitions for Determination BLOB XML .....39
    - Generating schema definitions ..... 40
- Chapter 7. Setup steps to run the IBM Social Program Management Data Extractor..... 44**
  - Prerequisite setup steps.....44
  - Creating and dropping the database tables..... 44
    - Required database access..... 45
  - Editing the Spring Profile..... 46
    - Filter Flow job configuration properties..... 47
    - Job configuration properties (applies to all job types)..... 48

Editing the Social Program Management database data source properties.....	51
Editing the Data Warehouse database data source properties.....	51
The ShortNames.properties file .....	52
Getting the database drivers.....	54
<b>Chapter 8. Running the IBM Social Program Management Data Extractor.....</b>	<b>55</b>
Starting the IBM Social Program Management Data Extractor in server mode.....	55
Starting the IBM Social Program Management Data Extractor in non-server mode.....	58
The console log output when started in non-server mode: running a batch job.....	59
The console log output when the tool is started in non-server mode: searching for the valid Filter Flow attribute paths .....	60
Running the IBM Social Program Management Data Extractor's functions in server mode and non- server mode.....	62
Scheduling jobs .....	62
Generating the Filter Flow database tables DDL.....	65
Searching for the Filter Flow attribute paths.....	66
Generating XSDs for XML Flow ingested determination XML.....	66
Batch jobs.....	69
Retry Determination Extraction operations.....	73
Parameters for the IBM Social Program Management Data Extractor's functions.....	75
Parameters for scheduling jobs.....	75
Parameters for retrying Determination extraction operations.....	81
Parameters for generating the Filter Flow database tables DDL.....	82
Parameters for searching for the valid Filter Flow attribute paths.....	82
Parameters for generating XSDs for XML Flow- ingested determination XML.....	82
Parameters for batch jobs.....	83
Stopping the IBM Social Program Management Data Extractor.....	85
<b>Chapter 9. Appendixes.....</b>	<b>86</b>
Appendix A: Determination BLOB XML structure.....	86
Appendix B: Logging.....	87
Appendix C: An extract of the rule class that declares the attributes that are requested.....	87
Appendix D: Securing a connection to the Social Program Management and Data Warehouse databases over TLS 1.2.....	88
Appendix E: Configuring the IBM Social Program Management Data Extractor JMX/MBean API for remote access.....	90
Appendix F: How to perform profiling by using J9 VM tracing.....	92
<b>Notices.....</b>	<b>95</b>
Privacy Policy considerations.....	96
Trademarks.....	96

---

# Figures

1. BLOB data structure..... 3

---

# Tables

- 1. The process and write operations used by the three functions..... 10
- 2. .... 15
- 3. Columns that are in the Determinations, Decisions, and Attribute value CSV files..... 19
- 4. STREAMLINEMEDICAIDDISPLAYRULESET\_STREAMLINEMEDICAIDINCOMECATEGORY.csv sample.. 24
- 5. STREAMLINEMEDICAIDDISPLAYRULESET\_STREAMLINEMEDICAIDINCOMECATEGORY\_LIST\_EL  
IGIBLEMEMBERSNAME.csv sample..... 25
- 6. The number of tables that are written for each Filter Flow table..... 28
- 7. The number of database write operations for a chunk size of 20 with compaction disabled..... 29
- 8. The number of database write operations for a chunk size of 20 and a compaction limit of 10..... 29
- 9. .... 29
- 10. The XML format and the XSD files used. .... 40

---

# Chapter 1. IBM Social Program Management Data Extractor

The IBM Social Program Management Data Extractor makes Determinations data available in a format that can be used in a data warehouse with analytic tools. The IBM Social Program Management Data Extractor can be used to extract complex determination BLOB data into a format that is more easily queried.

Determinations data stores the display rules data that the caseworker application then uses to present and explain eligibility outcomes to caseworkers. Determinations data is also used for eligibility and entitlement reports. Determinations data is stored in a compressed format in the Social Program Management database, which makes it difficult to extract for use in reports. However, the IBM Social Program Management Data Extractor makes it easier to retrieve the display rule attributes of a product rule set from the Determinations data.

To use the IBM Social Program Management Data Extractor, customers must be familiar with Cúram Express Rules (CER), specifically with display rules and the role that the rules play in relation to determination data.

Operators must be familiar with eligibility and entitlement rules. For more information about eligibility and entitlement rules, see the *Eligibility and Entitlement Rules Getting Started* related link. Operators must also be familiar with how to use Cúram Rules to develop eligibility and entitlement. For more information about Cúram Rules, see the *Developing with Eligibility and Entitlement by using Cúram Rules* related link.

## **Related information**

[Eligibility and Entitlement Getting Started](#)

[Developing with Eligibility and Entitlement by using Cúram Rules](#)

---

## Chapter 2. Determining case eligibility and entitlement

The IBM Social Program Management Data Extractor extracts complex determination BLOB data into a format that is more easily queried. The IBM Social Program Management Data Extractor currently supports only the extraction of the data for determinations that are associated with product delivery cases.

### Eligibility and entitlement engine

In IBM SPM, the eligibility and entitlement engine uses Cúram Express Rules (CER) to apply rules to real-world data to determine eligibility and entitlement. The following list outlines the main characteristics of the eligibility and entitlement engine:

- The starting point for case eligibility and entitlement is the product.
- A product contains all the configuration details that specify the CER rules to use for the determination of eligibility and entitlement.
- When a customer configures a product, the product's configuration can be used to calculate and store a determination result that is based on the input data. The determination result is used to generate financials and is retrieved when a caseworker views eligibility and entitlement details for the case.
- Decision display rules are used to display information about decisions in the application. For example, the decision display rules can be used to display the key criteria that is used to determine the income assistance payable to a family in need.
- The information to display comes from the display rules XML output or determination results. The results are stored in the Social Program Management database, in a BLOB type field called CREOLECASEDETERMINATIONDATA . CREOLESNAPSHOTDATA as a compressed, XML document.

### Determinations

A determination is a collection of one or more decisions. Each decision persists the following BLOB models:

- Display rules; for more information about display rules, see the *Display rules* related link.
- CER session data; for more information about CER session data, see the *CREOLECaseDeterminationData* related link.

The display rules model is a summary of important data that was used in the calculation or that was calculated in the determination. The purpose of storing the data is to permit the caseworker to better understand decisions and determinations.

For every new determination that is calculated, a new CREOLECASEDETERMINATION row is inserted. The following list applies where the determination is the first determination for a case or where the results of the determination differ from the current determination for the case:

- A new CREOLECASEDETERMINATIONDATA is created to store the display rules data for the determination. The stored data is compressed.
- The DETERMINATIONRESULTDATAID column of the CREOLECASEDETERMINATION row points to the CREOLECASEDETERMINATIONDATA row.

Where the option to persist CER session data is enabled, another CREOLECASEDETERMINATIONDATA row or rows are created to store the session data and the following list applies:

- The stored data is also compressed.
- The RULEOBJECTSNAPSHOTDATAID column of the CREOLECASEDETERMINATION row points to the other CREOLECASEDETERMINATIONDATA row. If the result of the determination does not differ from

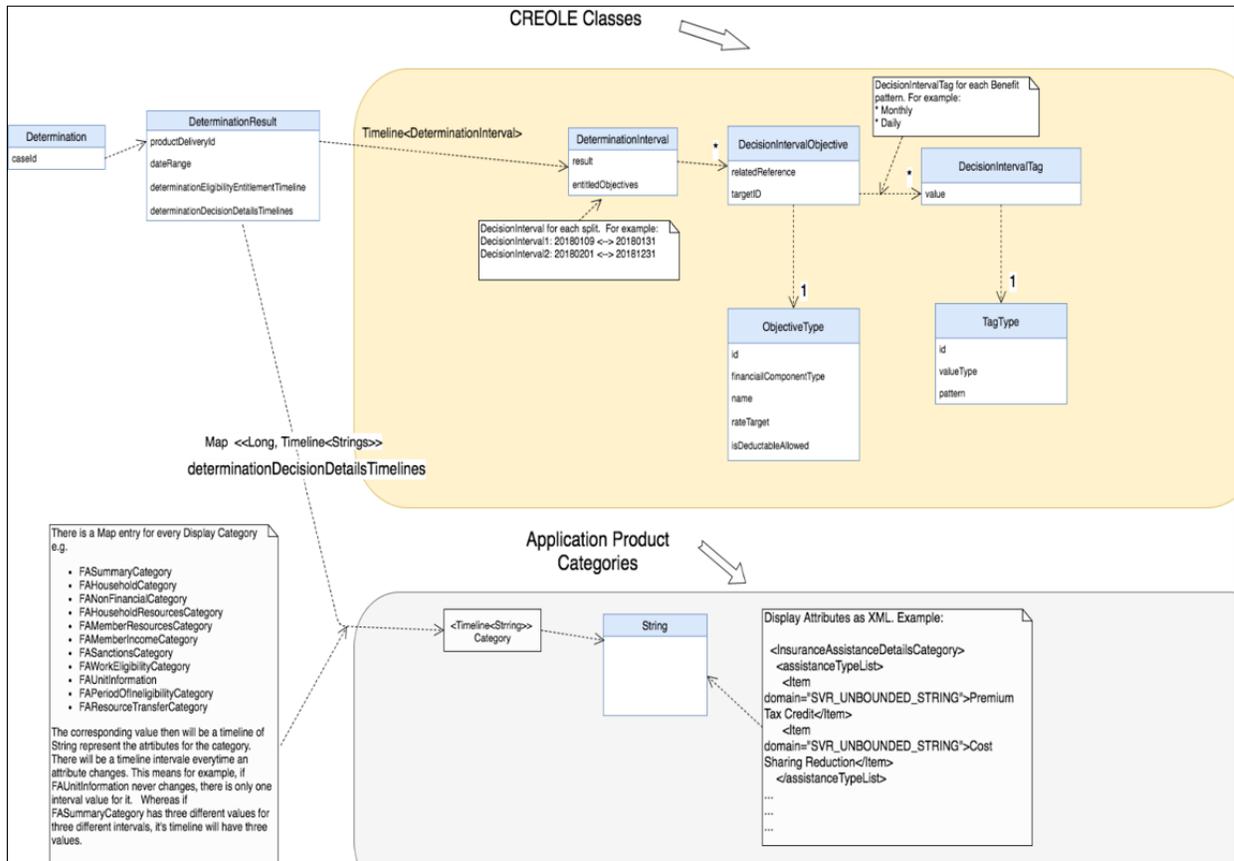
the current determination, the CREOLECASEDETERMINATION.DETERMINATIONRESULTDATAID points to the existing CREOLECASEDETERMINATIONDATA row with the display rule data.

If the option to persist CER session data is enabled, the CREOLECASEDETERMINATION.RULEOBJECTSNAPSHOTDATAID points to the existing CREOLECASEDETERMINATIONDATA row with the rule session data.

### Extracted BLOB data format

When the BLOB for the display rule data is extracted and parsed, the XML structure is as shown in the following figure.

Figure 1. BLOB data structure



The complex data structure consists of multiple parts. The first part of the data is structured as illustrated in the top box of the figure. The data follows the structure for all determinations.

The second part of the data is structured as illustrated by the bottom box of the figure. The second part of the data is the display rules data. As the displayable categories and the displayable attributes vary from product to product, so does the structure of the display rules data.

The extracted BLOB is a collection of XML files. The first XML file is a hierarchical representation that corresponds to the CREOLE rule classes. After the first XML, there is a collection of XML files that are all escaped. Each escaped XML file corresponds to a category for a specific period. For more information about the BLOB XML structure, see the *Appendix A* related link.

### Related concepts

[Appendix A: Determination BLOB XML structure](#)

When the operator opens a determination in the Social Program Management application, a set of decisions for different periods is displayed. When the operator selects a period, a list of categories is displayed. Under each category, a set of values is displayed.

**Related information**

[Display rules](#)

[CREOLECaseDeterminationData](#)

---

## Chapter 3. Components of the IBM Social Program Management Data Extractor

The components of the IBM Social Program Management Data Extractor are the Spring technology stack, the Social Program Management data source, the Data Warehouse data source, the Extraction tables, and the SPM Tools tables.

The IBM Social Program Management Data Extractor is developed for Java 8 and it requires a Java 8 Standard Development Kit (JDK). For information about how to start the IBM Social Program Management Data Extractor, see the *Running the IBM Social Program Management Data Extractor* related link.

### Spring technology stack

The IBM Social Program Management Data Extractor tool was developed as a Spring® Boot application. For more information about Spring Boot, see the *Spring Boot* related link. The tool uses other frameworks from the Spring stack, for example, Spring Data, Spring JMX, and Spring Batch. For more information about the Spring stack, see the *Spring* related link.

As a Spring Boot application, the IBM Social Program Management Data Extractor supports the use of multiple profiles. A profile is used to segregate application configuration. The tool uses the Java properties to store the application configuration. By default, the name of the properties file, which is commonly referred to as the Spring Profile, is `application.properties`.

The IBM Social Program Management Data Extractor tool can run in two modes: server and non-server mode. The tool's functions can be called in either mode. Selecting the mode that the tool runs in is based on the Java command that is used to start the tool. For more information, see the *Running the IBM Social Program Management Data Extractor* related link. For a full list of the Data Extractor's functions, see the *Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode* related link.

The following list outlines the differences between the two modes:

- When the operator starts the tool in server mode, the operator does not specify the function to start. The tool starts an MBean server and registers the MBeans that provide APIs for calling the tool functions by using JConsole. The operator can then call the tool's functions by using an MBean API. The tool runs until the operator explicitly shuts down the Java virtual machine (JVM). The following notes apply to running the tool in server mode:
  - By default, the tool limits access to the MBean server to the local computer only.
  - IBM does not recommend exposing remote access to the MBeans. However, customers can choose to open the MBeans to remote access where customers follow specific steps. For information about the steps, see the *Appendix E: Configuring the Data Extractor JMX/MBean API for remote access* related link.
  - The operator who wants to use the MBean APIs can use JConsole to connect to the MBeans that are registered by the tool. JConsole is packaged as part of a Java Development Kit (JDK). For information about JConsole, see the *Using JConsole* related links.
- When the tool is run in non-server mode, the operator must specify the tool function to run. The tool starts, runs that one function, and then exits. In non-server mode, the IBM Social Program Management Data Extractor MBeans are not registered, so the MBean API is unavailable.

The tool includes three batch job types that extract display rules determination data from the source Social Program Management database. The operator can schedule the batch jobs in server mode by using the MBean APIs or in non-server mode by using the command line.

Operators can schedule the following three jobs to run:

## XML Flow

- For information about how to schedule the job to run by using the MBean API, see the *Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode* related link.
- For information about how to schedule the job to run by using the command line in non-server mode, see the *Starting the IBM Social Program Management Data Extractor in non-server mode* and the *Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode* related links.

## Filter Flow CSV job

- For information about how to schedule the job to run by using the MBean API, see the *Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode* related link.
- For information about how to schedule the job to run by using the command line in non-server mode, see the *Starting the IBM Social Program Management Data Extractor in non-server mode* and the *Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode* related links.

## Filter Flow database job

- For information about how to schedule the job to run by using the MBean API, see the *Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode* related link.
- For information about how to schedule the job to run by using the command line in non-server mode, see the *Starting the IBM Social Program Management Data Extractor in non-server mode* and the *Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode* related links.

The tool accesses two data sources: the SPM data source (the source, from which data is extracted) and the Data Warehouse data source (the target, to which data is written).

## SPM data source

The SPM data source is used to connect to the source Social Program Management database. The database must be a copy or a replica of the database that is used by the SPM application and must not be the live production database. The IBM Social Program Management Data Extractor tool requires read-only access to a subset of the tables in the Social Program Management database. For more information, see the *Required database access* related link.

## Data Warehouse data source

The Data Warehouse data source is used to connect to two sets of tables that are collocated in the same schema of a second database, referred to as the Data Warehouse database. The first set of tables is referred to as the Extraction tables. The second set of tables is referred to as the SPM Tools tables.

For information about how to configure the application to connect to the two data sources, see the *Editing the Spring Profile* related link.

## Extraction tables

The Extraction tables are where the tool's batch jobs write the data that the tables extracted and processed from the Social Program Management database. The following list is included in the tables:

- The DETERMINATIONXML table, which is written to by the XML Flow job. For every determination that is read and processed by the execution of an XML Flow job, a row is written to the DETERMINATIONXML table.
- The Filter Flow database tables, which are used by the Filter Flow database job.

- The DETERMINATIONS table, which is written to by the Filter Flow database job. For every determination that is read and processed by the execution of a Filter Flow database job, a row is written to the DETERMINATIONS table.
- The DECISIONS table, which is written to by the Filter Flow database job. For every decision split in a determination that is read and processed by the execution of a Filter Flow database job, a row is written to the DECISIONS table.
- Filter Flow attribute value tables, which is written to by the Filter Flow database job, where the data items that the operator specifies from the display rules determination data are written.

The Data Definition Language (DDL) scripts to create the DETERMINATIONXML, DECISIONS, and DETERMINATIONS tables are packaged as part of the IBM Social Program Management Data Extractor tool. For more information, see the *Downloading and installing the IBM Social Program Management Data Extractor* related link.

However, the DDL for the Filter Flow attribute value tables must be generated by using the tool because the schema of these tables is computed from the Rules Set Definitions in the Social Program Management database and the set of attributes, that is, data items, to be extracted as specified by the operator in `application.properties`. Both inputs can vary between configurations. So, the DDL must be generated by using the tool because the schema cannot be known without knowing these two inputs first. For more information, see the *Generating the Filter Flow attribute value database tables creation and drop DDL* related link.

### **SPM Tools tables**

The SPM Tools tables are infrastructure tables that are used in the running of the IBM Social Program Management Data Extractor tool. For the first release, the SPM Tools tables include tables that are written to by the tool's Spring Batch infrastructure. The tables store information about the job instances that are scheduled and the status of job executions. The script for creating the tables is packaged as part of the tool. For more information, see the *Downloading and installing the IBM Social Program Management Data Extractor* related link. Create the tables in the same database and the schema as the Extraction tables.

### **Related concepts**

[Generating the Filter Flow attribute value database tables creation and drop DDL](#)

Generating the creation and drop DDL for the Filter Flow attribute value tables is a function of the IBM Social Program Management Data Extractor.

### **Related tasks**

[Appendix E: Configuring the IBM Social Program Management Data Extractor JMX/MBean API for remote access](#)

Customers can perform a series of steps to expose the MBean API. However, to expose the MBean API customers must secure the connection over TLS v1.2 and configure client certificate-based authentication as a requirement.

### **Related information**

[Running the IBM Social Program Management Data Extractor](#)

[Spring Boot](#)

[Spring](#)

[Using JConsole with Oracle Java](#)

[Using JConsole with IBM Java](#)

[Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode](#)

[Starting the IBM Social Program Management Data Extractor in non-server mode](#)

[Required database access](#)

[Editing the Spring Profile](#)

[Downloading and installing the IBM Social Program Management Data Extractor](#)

# Chapter 4. Downloading and installing the IBM Social Program Management Data Extractor

Extract the contents of the compressed file and run the installer jar.

Use the following steps to download and install the IBM Social Program Management Data Extractor:

1. Extract the contents of the downloaded compressed file to a local drive location.
2. Run the installer jar, which can be found in the **INSTALLER** folder at that location.

The installer extracts the tool binary, the sample properties files, and the SQL scripts. The installation location contains the following contents:

```
DataExtractor
├── DataExtractor-<version_no>.jar
├── samples
│   ├── application-SAMPLE.properties
│   ├── log4j2.properties
│   └── ShortNames-SAMPLE.properties
├── sql
│   ├── db2
│   │   ├── CreateDataWarehouseTables.sql
│   │   ├── CreateSPMToolsTables.sql
│   │   ├── DropDataWarehouseTables.sql
│   │   └── DropSPMToolsTables.sql
│   ├── h2
│   │   ├── CreateDataWarehouseTables.sql
│   │   ├── CreateSPMToolsTables.sql
│   │   ├── DropDataWarehouseTables.sql
│   │   └── DropSPMToolsTables.sql
│   └── oracle
│       ├── CreateDataWarehouseTables.sql
│       ├── CreateSPMToolsTables.sql
│       ├── DropDataWarehouseTables.sql
│       └── DropSPMToolsTables.sql
├── Installer
│   ├── CuramInstaller.log
│   └── Installhistory.txt
├── license
│   └── IBM\ Social\ Program\ Management\ Data\ Extractor
│       └── license
│           ├── Chinese_TW.txt
│           ├── Chinese.txt
│           ├── Czech.txt
│           ├── English.txt
│           ├── French.txt
│           ├── German.txt
│           ├── Greek.txt
│           ├── Indonesian.txt
│           ├── Italian.txt
│           ├── Japanese.txt
│           ├── Korean.txt
│           ├── Lithuanian.txt
│           ├── notices.txt
│           ├── Polish.txt
│           ├── Portuguese.txt
│           ├── Russian.txt
│           ├── Slovenian.txt
│           ├── Spanish.txt
│           ├── status.dat
│           └── Turkish.txt
└── Uninstaller
    └── uninstaller.jar
```

The InstallLocation/DataExtractor/DataExtractor-<version\_no>.jar is the application binary. In Spring Boot terminology, the JAR file is a fat or uber JAR. The Uber jar is the binary that delivers the tool's functionality.

## The samples folder

The **samples** folder contains the following files:

- The `application-SAMPLE.properties` file. The file is a sample Spring Profiles File, the main configuration file for the tool. For information about how to use the `application-SAMPLE.properties` file to create your own Spring Profile, see the *Running the IBM Social Program Management Data Extractor* related link.
- The `ShortNames-SAMPLE.properties` file. The file is a sample "Short-Names" substitution file. The customer is advised to create a copy and rename this copy to `ShortNames.properties` and move the file to the same folder on the `loader.path`.
- The `ShortNames.properties` is required to solve the issues with the length of filter flow database table names. For more information, see the *Issues associated with the length of a generate table name* related link. For information about how to use the `ShortNames.properties` file, see the *The ShortNames.properties file* related link.
- The `log4j2.properties` file. The file does not have a `-SAMPLE` suffix because it is usable as it is. For information about how to use the `log4j2.properties` file, see the *Appendix B: Logging* related link.

### The sql folder

The **sql** folder contains database vendor-specific versions of the following scripts:

- `CreateSPMToolsTables.sql`: The script is used to create the SPM Tools tables. This includes the DDL to create the tables that are used by the Spring Batch infrastructure.
- `DropSPMToolsTables.sql`: The script is used to drop the SPM Tools tables.
- `CreateDataWarehouseTables.sql`: The script is used to create the Data Warehouse tables. This script contains the DDL to create the `DETERMINATIONXML`, `DETERMINATIONS`, and `DECISIONS` tables. For more information, see the *Creating the database tables* related link.
- `DropDataWarehouseTables.sql`: This script is used to drop the Data Warehouse tables.

### Related concepts

[Appendix B: Logging](#)

The IBM Social Program Management Data Extractor uses Apache Log4j 2 as its logging service.

### Related information

[Running the IBM Social Program Management Data Extractor](#)

[Issues associated with the length of a generated table name](#)

[The ShortNames.properties file](#)

[Creating the database tables](#)

---

## Chapter 5. Batch jobs

The batch jobs for the IBM Social Program Management Data Extractor run asynchronously. The jobs all read a set of determinations that is based on criteria that are specified by the operator. The jobs process these determinations by extracting and transforming the data. Finally, the jobs are used to write the transformed data to a destination.

### Batch jobs: job types and their purposes

---

The batch jobs that the IBM Social Program Management Data Extractor runs are Spring Batch jobs. Spring Batch uses a chunk-oriented processing approach.

#### Chunk-oriented processing

In a chunk-oriented processing approach, items are handled in the following ways:

- Items are read and processed individually.
- Items are then written out in chunks up to a fixed size.

The Spring Batch *Configuring a step* documentation includes the following description for chunk-oriented processing: "Chunk oriented processing refers to reading the data one at a time and creating 'chunks' that will be written out, within a transaction boundary." After a transaction boundary, a commit operation is performed.

#### Single step job configuration

The IBM Social Program Management Data Extractor performs single step jobs. The following list outlines how single step jobs are configured:

- An ItemReader implementation for reading input items of type *I*, individually
- An ItemProcessor implementation from processing items of type *I* and creating output items of type *O*
- An ItemWriter implementation for writing out chunks of items of type *O*

#### Batch job implementation

The following table outlines the procedures that are used by the XML Flow, the Filter Flow database job, and the Filter Flow CSV when the functions process and write jobs. The following list applies to the XML Flow, the Filter Flow database job, and the Filter Flow CSV:

- A generic ItemReader is used to read all functions.
- A generic ItemProcessor is used to convert IDs to Plain Old Java Objects (POJOs).

Function	Process	Write
XML Flow	XML Flow ItemProcessor	XML Flow Writer
Filter Flow database job	Filter Flow ItemProcessor	Filter Flow Database Writer
Filter Flow CSV	Filter Flow ItemProcessor	Filter Flow CSV Writer

In Java, a singleton is a class that provides only one instance of itself and is used as a point of access. For the IBM Social Program Management Data Extractor, the singleton steps from all three job types use the same ItemReader implementation. The implementation runs a reader query in a paged way. For more

information, see the *How determinations are read by the IBM Social Program Management Data Extractor batch jobs* related link.

The implementation initially runs a database query to read  $N$  number of items into an in-memory buffer. In the IBM Social Program Management Data Extractor jobs, the items are CREOLECASEDETERMINATION.CREOLECASEDETERMINATIONIDs. The implementation serves the  $N$  number of items individually until the buffer is empty. Then, the reader runs a second query to fetch the next  $N$  items into the buffer. The process repeats until no results are returned by the query.

The singleton steps also use chained ItemProcessors, that is, a pair of ItemProcessors where the output of the first ItemProcessor of the pair is piped to the input of the second ItemProcessor of the pair. The first ItemProcessor is used to resolve a CREOLECASEDETERMINATION.CREOLECASEDETERMINATIONID to a Plain Old Java Object (POJO) that contains all the data that is needed by the second ItemProcessor of the pair.

All three jobs use the same implementation of the first ItemProcessor in the pair. The Filter Flow jobs use the same ItemProcessor implementation for the second processor of the pair. The XML Flow job uses a dedicated ItemProcessor implementation for the second processor. All three jobs use different ItemWriter implementations.

For the XML Flow and Filter Flow database jobs, a job can optionally be scaled by having the step for that job as a partitioned step. So, several step executions can run in parallel. As a result, a single execution of a job starts multiple step executions in parallel. For example, with **gridSize**  $P$ ,  $P$  step executions are started with their own dedicated ItemReader instance and run the execution cycle that is shown in Table 1.

A dedicated partitioner creates a separate step execution context that effectively divides the range of CREOLECASEDETERMINATION.CREOLECASEDETERMINATIONID into  $P$  partitions. Each execution step is then used to read, process, and write the determination data in the partition for which it is responsible.

In the release of the tool, a thread-based grid execution fabric is used. So, each step execution has a corresponding thread in the same JVM as the IBM Social Program Management Data Extractor tool itself.

A larger grid size  $P$  means that instead of a job execution that reads-processes-writes items serially, the job execution is performed in parallel over  $P$  # threads. So, the job execution typically takes less time than with a single thread. When specified to be a value greater than one, the tool creates partitioned step executions.

### Criteria for processing determinations

Other than criteria that is fixed, the operator can specify some of the criteria for matching determinations. The jobs only process determinations that meet the following criteria:

1. Determinations that belong to one specific product type OR to a specific product. Operators must set a product type code or a product ID in the Spring Profile (`applications.properties`). For more information, see the *Job configuration properties (applies to all job types)* related link.
2. Determinations of type Manual Check Eligibility (CDT1) can optionally be excluded. By default, the determinations are included. For more information, see the *Job configuration properties (applies to all job types)* related link.
3. Determinations that were made within a specified period. By default, the period is assumed to be the last three days from the moment the job was scheduled. However, the period can be overridden by a configuration property or by a parameter in the MBean API operation for scheduling the job, or as job parameter when the operator schedules the job from the command line, that is, in non-server mode. For more information, see the *Scheduling a job execution*, *Job configuration properties (applies to all job types)*, and *Starting the IBM Social Program Management Data Extractor in non-server mode* related links.

### Related concepts

[How determinations are read by the IBM Social Program Management Data Extractor batch jobs](#)

The reader that is used by all three job types reads the set of CREOLECASEDETERMINATIONIDS for the rows of the CREOLECASEDETERMINATION table in the Social Program Management database that match the preceding criteria.

### Related information

[Spring Batch: Configuring a Step](#)

[Job configuration properties \(applies to all job types\)](#)

[Scheduling a job execution](#)

[Starting the IBM Social Program Management Data Extractor in non-server mode](#)

## How determinations are read by the IBM Social Program Management Data Extractor batch jobs

The reader that is used by all three job types reads the set of CREOLECASEDETERMINATIONIDS for the rows of the CREOLECASEDETERMINATION table in the Social Program Management database that match the preceding criteria.

### Criteria for processing determinations

Other than criteria that is fixed, the operator can specify some of the criteria for matching determinations. The jobs only process determinations that meet the following criteria:

1. Determinations that belong to one specific product type OR to a specific product. Operators must set a product type code or a product ID in the Spring Profile (`applications.properties`). For more information, see the *Job configuration properties (applies to all job types)* related link.
2. Determinations of type Manual Check Eligibility (CDT1) can optionally be excluded. By default, the determinations are included. For more information, see the *Job configuration properties (applies to all job types)* related link.
3. Determinations that were made within a specified period. By default, the period is assumed to be the last three days from the moment the job was scheduled. However, the period can be overridden by a configuration property or by a parameter in the MBean API operation for scheduling the job, or as job parameter when the operator schedules the job from the command line, that is, in non-server mode. For more information, see the *Scheduling a job execution*, *Job configuration properties (applies to all job types)*, and *Starting the IBM Social Program Management Data Extractor in non-server mode* related links.

The job reads the IDs in a paged way,  $N$  IDs at a time, where  $N$  is the value of the `pageSize` job parameter where specified. Otherwise,  $N$  defaults to the value of the `spm.extract.pagesize` property.

### Running a job with a single partition

- Use the following query to read the first page of IDs (assuming  $N=10$ ) from a Social Program Management database on Db2:

```
SELECT CCDID FROM (
    SELECT CCD.CREOLECASEDETERMINATIONID AS CCDID
    FROM PRODUCTDELIVERY PD INNER JOIN CREOLECASEDETERMINATION CCD ON PD.CASEID
    = CCD.CASEID
    WHERE
        (0=:USEPRODUCTTYPECODE OR PD.PRODUCTTYPE=:PRODUCTTYPECODE) AND
        (1=:USEPRODUCTTYPECODE OR PD.PRODUCTID=:PRODUCTID) AND
        (1=:INCLUDEMANUALELIGIBILITYCHECK OR CCD.TYPE!:=EXCLUDEDETERMINATIONTYPE)
    AND
        CCD.DETERMINATIONDATETIME >=:STARTDATE AND CCD.DETERMINATIONDATETIME
    <:ENDDATE ) ORDER BY CCDID ASC FETCH FIRST 10 ROWS ONLY
```

- Use the following query to read the second and later page of IDs (assuming  $N=10$ ) from a Social Program Management database on Db2:

```
SELECT CCDID FROM (
    SELECT CCD.CREOLECASEDETERMINATIONID AS CCDID
    FROM PRODUCTDELIVERY PD INNER JOIN CREOLECASEDETERMINATION CCD ON PD.CASEID
    = CCD.CASEID
    WHERE
        (0=:USEPRODUCTTYPECODE OR PD.PRODUCTTYPE=:PRODUCTTYPECODE) AND
```

```

(1=:USEPRODUCTTYPECODE OR PD.PRODUCTID=:PRODUCTID) AND
(1=:INCLUDEMANUALELIGIBILITYCHECK OR CCD.TYPE!:=:EXCLUDEDETERMINATIONTYPE)
AND
CCD.DETERMINATIONDATETIME >=:STARTDATE AND CCD.DETERMINATIONDATETIME
<:ENDDATE)WHERE ((CCDID > :_CCDID)) ORDER BY CCDID ASC FETCH FIRST 10 ROWS ONLY

```

### Running a job with multiple partitions

- Use the following query to read the first page of IDs (assuming N=10) from a Social Program Management database on Db2:

```

SELECT CCDID FROM (
  SELECT CCD.CREOLECASEDETERMINATIONID AS CCDID
  FROM PRODUCTDELIVERY PD INNER JOIN CREOLECASEDETERMINATION CCD ON PD.CASEID
  = CCD.CASEID
  WHERE
  (0=:USEPRODUCTTYPECODE OR PD.PRODUCTTYPE=:PRODUCTTYPECODE) AND
  (1=:USEPRODUCTTYPECODE OR PD.PRODUCTID=:PRODUCTID) AND
  (1=:INCLUDEMANUALELIGIBILITYCHECK OR CCD.TYPE!:=:EXCLUDEDETERMINATIONTYPE)
AND
  CCD.DETERMINATIONDATETIME >=:STARTDATE AND CCD.DETERMINATIONDATETIME
<:ENDDATE)WHERE ((CCDID > :_CCDID)) ORDER BY CCDID ASC FETCH FIRST 10 ROWS ONLY

```

- Use the following query to read the second and later page of IDs (assuming N=10) from a Social Program Management database on Db2:

```

SELECT CCDID FROM (
  SELECT CCD.CREOLECASEDETERMINATIONID AS CCDID
  FROM PRODUCTDELIVERY PD INNER JOIN CREOLECASEDETERMINATION CCD ON PD.CASEID
  = CCD.CASEID
  WHERE
  (0=:USEPRODUCTTYPECODE OR PD.PRODUCTTYPE=:PRODUCTTYPECODE) AND
  (1=:USEPRODUCTTYPECODE OR PD.PRODUCTID=:PRODUCTID) AND
  (1=:INCLUDEMANUALELIGIBILITYCHECK OR CCD.TYPE!:=:EXCLUDEDETERMINATIONTYPE)
AND
  CCD.DETERMINATIONDATETIME >=:STARTDATE AND CCD.DETERMINATIONDATETIME
<:ENDDATE AND
  CCD.CREOLECASEDETERMINATIONID>=:LOWERBOUND AND
  CCD.CREOLECASEDETERMINATIONID<=:UPPERBOUND ) WHERE ((CCDID > :_CCDID)) ORDER BY CCDID ASC
  FETCH FIRST 10 ROWS ONLY

```

### Related information

[Job configuration properties \(applies to all job types\)](#)

[Scheduling a job execution](#)

[Starting the IBM Social Program Management Data Extractor in non-server mode](#)

## XML Flow job

Operators can use the XML Flow job to read the determinations that match the proceeding criteria.

### Criteria for processing determinations

Other than criteria that is fixed, the operator can specify some of the criteria for matching determinations. The jobs only process determinations that meet the following criteria:

1. Determinations that belong to one specific product type OR to a specific product. Operators must set a product type code or a product ID in the Spring Profile (`applications.properties`). For more information, see the *Job configuration properties (applies to all job types)* related link.
2. Determinations of type Manual Check Eligibility (CDT1) can optionally be excluded. By default, the determinations are included. For more information, see the *Job configuration properties (applies to all job types)* related link.
3. Determinations that were made within a specified period. By default, the period is assumed to be the last three days from the moment the job was scheduled. However, the period can be overridden by a configuration property or by a parameter in the MBean API operation for scheduling the job, or as job parameter when the operator schedules the job from the command line, that is, in non-server mode. For more information, see the *Scheduling a job execution, Job configuration properties (applies to all*

job types), and *Starting the IBM Social Program Management Data Extractor in non-server mode* related links.

For each determination, use the XML Flow job to perform the following actions:

- Ingest the determination BLOB from CREOLECASEDETERMINATIONDATA.CREOLESNAPSHOTDATA.
- Write the transformed XML to the DETERMINATIONXML.XMLVALUE column in the data warehouse.

The DETERMINATIONXML.XMLVALUE column is of the XML data type for Db2 or XMLType data type for Oracle. For more information about the XML data type for Db2, see the *XML data type* related link. For more information about the XMLType data type for Oracle, see the *XMLTYPE* related link. The data type supports XQuery functionality. For more information about the XQuery functionality, see the *An introduction to XQuery* related link.

The SQL script for creating the DETERMINATIONXML table is supplied as part of the tooling. For more information, see the *Components of the IBM Social Program Management Data Extractor* and the *Downloading and installing the IBM Social Program Management Data Extractor* related links.

When the job transforms the determination of the XML, the job performs the following five tasks:

1. The job extracts the compressed data to its original XML form.
2. The job unescapes the XML so that the sections that pertain to decisions are now part of the XML document structure. The step is to permit the XPath to data query the decisions.
3. The job scrubs the XML of any XML declarations. The step is to permit the XPath to data query the decisions.
4. The job augments the XML with the following data that is taken from the corresponding CREOLECASEDETERMINATION.

The XML uses the following seven attributes:

- ASSESSMENTSTATUS
- ASSESSMENTREASON
- TYPE
- DETERMINATIONDATETIME
- CREOLECASEDETERMINATIONID
- CREATEDBYUSER
- RULEOBJECTSNAPSHOTDATAID

The information appears as attributes of an IngestedData element as the last child of the root Determination element. The nested decision nodes are links from the decision split dates in the determination XML to the case decision record on the database. The nested determinationResultDataId nodes are used to list the CREOLECASEDETERMINATIONDATAID identifiers for CREOLECASEDETERMINATIONDATA records that hold the raw display rules data. There might be more than one for a determination. The CREOLECaseDeterminationData page of the IBM Cúram Social Program Management Knowledge Center includes the following description: "In the unlikely event that the XML data is too long to fit onto a single CREOLECaseDeterminationData, the data will be truncated to fit, and the extra data stored on an "overflow" CREOLECaseDeterminationData row (or chain of overflow rows)."

```
<IngestedData
  assessmentReason="CADR6"
  assessmentStatus="CDAS1"
  type="CDT3"
  determinationDateTime="2019-01-21"
  creoleCaseDeterminationId="-6587007036238594048"
  createdByUser="SYSTEM"
  ruleObjectSnapshotDataId="5878956732322938880">
  <Decisions>
    <Decision caseDecisionId="5673232293888045101" endDate="20140531" startDate="20140425"/>
    <Decision caseDecisionId="5673232293888045102" endDate="20141026" startDate="20140601"/>
    <Decision caseDecisionId="5673232293888045102" endDate="20141031" startDate="20141027"/>
    <Decision caseDecisionId="5673232293888045103" endDate="20141130" startDate="20141101"/>
    <Decision caseDecisionId="5673232293888045103" endDate="20141231" startDate="20141201"/>
  </Decisions>
</IngestedData>
```

```

</Decisions>
<DeterminationResultDataIds>
  <DeterminationResultDataId>6527475078664290304</DeterminationResultDataId>
  <DeterminationResultDataId>-1470917859545710592</DeterminationResultDataId>
  <DeterminationResultDataId>3068710564843749376</DeterminationResultDataId>
</DeterminationResultDataIds>
</IngestedData>

```

5. The resulting XML is written to the XMLVALUE column of the DETERMINATIONXML table.

### Columns in DETERMINATIONXML

The columns JOBINSTANCEID, JOBEXECUTIONID, and EXTRACTIONDATE are included in all Extraction tables. The columns have the same meaning in each table.

<i>Table 2.</i>	
Column name	Column description
created	The time the row was created.
caseid	The case identifier for the case for which the original determination was created.
xmlvalue	The transformed XML.
jobinstanceid	<p>The value in the column corresponds to the unique identifier for the job instance of the XML Flow job that was used to write the row.</p> <p>The <i>The Domain Language of Batch</i> documentation includes the following description for a job instance: "A JobInstance refers to the concept of a logical job run."</p> <p>The job instance ID is the same one that is displayed in the logs in the printed summary after a job completes.</p> <pre> Job Name: 'XmlFlow' Instance: 23 Execution: 43 Finished Step: 'xmlFlowStep0' Step Summary: 'StepExecution: id=43, version=7, name=step,  status=COMPLETED, exitStatus=COMPLETED, readCount=53,  filterCount=0, writeCount=53 readSkipCount=0,  writeSkipCount=0, processSkipCount=0, commitCount=6, rollbackCount=0' </pre>

Table 2. (continued)

Column name	Column description
jobexecutionid	<p>The value in the column corresponds to the unique identifier for the job execution of the XML Flow job that was used to write the row.</p> <p>The <i>The Domain Language of Batch</i> documentation includes the following description for a job execution: "A JobExecution refers to the technical concept of a single attempt to run a Job."</p> <p>The job execution ID is the same one that is displayed in the logs in the printed summary after a job completes.</p> <pre>Job Name: 'XmlFlow' Instance: 23 Execution: 43 Finished Step: 'xmlFlowStep0'</pre> <pre>Step Summary: 'StepExecution: id=43, version=7, name=step, status=COMPLETED, exitStatus=COMPLETED, readCount=53, filterCount=0, writeCount=53 readSkipCount=0, writeSkipCount=0, processSkipCount=0, commitCount=6, rollbackCount=0'</pre> <p>A job that is restarted has the same job instance ID when it restarts. However, the new attempt to run the job has a new job execution ID.</p>
extractiondate	<p>The value in the column corresponds to the start time of the job execution that was used to write the row.</p>

**Related information**

[Job configuration properties \(applies to all job types\)](#)

[Scheduling a job execution](#)

[Starting the IBM Social Program Management Data Extractor in non-server mode](#)

[XML data type](#)

[XMLTYPE](#)

[An introduction to XQuery](#)

[Components of the IBM Social Program Management Data Extractor](#)

[Downloading and installing the IBM Social Program Management Data Extractor](#)

[CREOLECaseDeterminationData](#)

[The Domain Language of Batch](#)

## Filter Flow CSV job

---

Use the Filter Flow job to match a set of determinations and specify the attributes.

Determinations for a product can have many display attributes. In many scenarios, customers are interested in only a few of these attributes. Additionally, customers often prefer such data in a relational format rather than in a hierarchical format.

The purpose of the Filter Flow job is to enable customers to perform the following two tasks:

- Match a set of determinations on the same set of criteria that is used by the XML Flow job.
- Specify the attributes in the matching determinations that the customer is interested in and extract this data to a destination that supports a relational format. For the Filter Flow CSV job, the data is a set of CSV files.

### How do I specify the attributes that I am interested in?

To specify the attributes that the Filter Flow job is to extract, customers must configure the `spm.extract.creole.determinations.displayrules.extractlistattributes` property in the Spring Profile properties file.

Specify the attributes as a comma-delimited list of attribute paths. An attribute path is a reference to an attribute in the determination XML that uses the format `{RuleSet} . {RuleClass} . {RuleAttribute}`. For example: `StreamlineMedicaidDisplayRuleSet.StreamlineMedicaidIncomeCategory.householdnonfinancialnoeligibletimeline`.

For information about how to search for all valid attribute paths, see the *Searching for the Filter Flow attribute paths* related link.

### Related information

[Searching for the Filter Flow attribute paths](#)

## The output of the Filter Flow CSV job

A successful Filter Flow CSV batch job produces several CSV files. Typically, the CSV includes the files `DETERMINATIONS.csv`, `DECISIONS.csv`, and `Attribute value` CSVs.

### DETERMINATIONS.csv

The `DETERMINATIONS.csv` is a single CSV file where the individual rows correspond to the determinations that were extracted and that reflect the state of the determination at the time of the extraction. For more information about the columns that are included in the `DETERMINATIONS.csv` files, see Table 1.

### DECISIONS.csv

`DECISIONS.csv` is a single CSV file. The file's individual rows correspond to the decision periods that are stored in the display rules data for the extracted determinations. In the `DECISIONS.csv` file, coverage periods and decision period are important concepts.

### Coverage periods

The rows in `DECISIONS.csv` correspond to the decision periods that are stored in the display rules data for the extracted determinations. In the display rules data representation of the decision period, there might be two consecutive periods where the client is eligible but is eligible for different reasons. In the display rules determination XML data, the two consecutive periods are classified as distinct coverage periods. Coverage periods is the terminology that is used in the caseworker application in Social Program Management. When the caseworker views the **Current Determination** page on a product delivery case, the caseworker is viewing a visual representation of the coverage period view of decisions within a determination.

## Decision periods

The other concept or model of decision period that also exists in the SPM product is described in the **Decision Periods** page. For more information about decision periods, see the *Decision Periods* related link. The decision period concept is where the eligibility and entitlement engine is used to split a determination into decision periods of constant eligibility and entitlement. The eligibility and entitlement engine is used to store each of those periods as a row on the CASEDECISION table and then links to the corresponding row in the CREOLECASEDETERMINATION table.

The IBM Cúram Social Program Management Knowledge Center *CaseDecision* documentation includes the following description for CaseDecision: "Sometimes it is possible for a determination result to contain key decision factors and/or decision details which change on a particular date where there is no accompanying change in eligibility and entitlement, for example where the case continues to be eligible but for a different business reason than previously. In these circumstances, the Engine will *not* store a CaseDecision record effective from the change date, because there has not been a change in eligibility and entitlement."

This CSV file links the individual determination decision period to the corresponding CASEDECISION row through the caseDecisionId column.

## Attribute value CSVs

The Filter Flow CSV job writes the extracted data to attribute values CSVs that correspond to the set of specified display rule attributes in the set of matching determinations.

The following rules determine the mapping of the attributes that are exported to the corresponding attribute CSV files:

1. Attributes that are not defined as lists in their corresponding rule class are always exported to a destination that corresponds to that rule class. The file names are in the format: `{RuleSet}_{RuleClass}.csv`. An example is `STREAMLINEMEDICAIDDISPLAYRULESET_STREAMLINEMEDICAIDINCOMECATEGORY.csv`.
2. Attributes that are defined as lists in their corresponding rule class are always exported to a destination that corresponds to that attribute only. The file names are in the format: `{RuleSet}_{RuleClass}_LIST_{RuleAttribute}.csv`. An example is `STREAMLINEMEDICAIDDISPLAYRULESET_STREAMLINEMEDICAIDINCOMECATEGORY_LIST_ELIGIBLEMEMBERSNAMES.csv`.

The following table lists and describes the applicable Filter Flow table columns. Some of the columns exist only in one of the CSV file types, that is, the DETERMINATIONS.csv, DECISIONS.csv, and Attribute value CSVs. In other cases, the applicable Filter Flow table columns exist in two or more of the CSV file types. In such instances, it is indicated in the left column.

Table 3. Columns that are in the Determinations, Decisions, and Attribute value CSV files.

Applicable Filter Flow tables	Column name	Description
All	caseref	<ul style="list-style-type: none"> <li>A reference ID, which has a path-like structure, that uniquely identifies a case.</li> <li>The structure of a caseref value is case/{caseID}.</li> </ul>
	determinationref	<ul style="list-style-type: none"> <li>A reference ID, which has a path-like structure, that uniquely identifies a determination.</li> <li>The structure of a determinationref value is case/{caseID}/determination/{determinationId}.</li> </ul>
	jobinstanceid	<ul style="list-style-type: none"> <li>The value of the jobinstanceid column corresponds to the unique identifier for the job instance of the Filter Flow that was used to write the row.</li> <li>The <i>The Domain Language of Batch</i> documentation includes the following description for a job instance:                      "A JobInstance refers to the concept of a logical job run."</li> <li>The job instance ID is the same one that is displayed in the logs in the printed summary after a job finishes:  <pre> Job Name: 'FilterFlowCsv' Instance: 21 Execution: 41 Finished Step: 'filterFlowCSVStep0' Step Summary: 'StepExecution: id=41,version=7, name=step, status=COMPLETED, exitStatus=COMPLETED, readCount=53, filterCount=0, writeCount=53 readSkipCount=0, writeSkipCount=0, processSkipCount=0, commitCount=6, rollbackCount=0'                     </pre> </li> </ul>
	jobexecutionid	<ul style="list-style-type: none"> <li>The jobexecutionid corresponds to the unique identifier for the job execution of the Filter Flow that was used to write the row.</li> <li>The <i>The Domain Language of Batch</i> documentation includes the following description for a job execution: "A JobExecution</li> </ul>

Table 3. Columns that are in the Determinations, Decisions, and Attribute value CSV files. (continued)

Applicable Filter Flow tables	Column name	Description
All except DETERMINATIONS	decisionref	<ul style="list-style-type: none"> <li>• A reference ID, which has a path-like structure, that uniquely identifies a decision.</li> <li>• In the Decisions CSV file, the values in the decisionref column on any row are unique within that file.</li> <li>• The structure of decisionref value is case/{caseID}/determination/{determinationId}/decision/{decisionDate} where decisionDate is in the date format yyyyMMdd.</li> </ul>
All except attribute value tables	productid	<ul style="list-style-type: none"> <li>• The unique reference number that is assigned by SPM to the product offering, for the product delivery case for which the determination was created.</li> </ul>
	producttypecode	<ul style="list-style-type: none"> <li>• The code table entry for the product type for the product offering. Multiple products can share the same product type code.</li> </ul>
DECISIONS	casedecisionid	<ul style="list-style-type: none"> <li>• The caseDecisionId for CASEDECISION that corresponds to the decision period.</li> </ul>
	startdate	<ul style="list-style-type: none"> <li>• The start date of the decision period of eligibility or ineligibility that is based on a set of criteria.</li> <li>• In the CSV, it is a date formatted string with the format yyyyMMdd</li> </ul>
	enddate	<ul style="list-style-type: none"> <li>• The end date of the decision period of eligibility or ineligibility that is based on a set of criteria.</li> <li>• In the CSV, it is a date formatted string with the format yyyyMMd</li> </ul>

Table 3. Columns that are in the Determinations, Decisions, and Attribute value CSV files. (continued)

Applicable Filter Flow tables	Column name	Description
DETERMINATIONS	status	<ul style="list-style-type: none"> <li>• The assessment status of the determination at the time of extraction.</li> <li>• It corresponds to the CREOLECASEDETERMINATION.STATUS of the determination at the time of the extraction.</li> </ul>
	type	<ul style="list-style-type: none"> <li>• The type of the determination.</li> <li>• It corresponds to the CREOLECASEDETERMINATION.TYPE of the determination at the time of the extraction.</li> </ul>
	determinationdatetime	<ul style="list-style-type: none"> <li>• The date and time that the determination was made.</li> <li>• It corresponds to the CREOLECASEDETERMINATION.DETERMINATIONDATETIME of the determination at the time of the extraction.</li> <li>• In the CSV, it is a date formatted string with the format yyyyMMdd.</li> </ul>

Table 3. Columns that are in the Determinations, Decisions, and Attribute value CSV files. (continued)

Applicable Filter Flow tables	Column name	Description
Attribute value tables only	attributeref	<ul style="list-style-type: none"> <li>• A hierarchically name-spaced reference that uniquely identifies a Rule Class Attribute instance, across all determinations.</li> <li>• A Filter Flow extraction flattens hierarchical XML data and maps it a relational format in the following ways:               <ul style="list-style-type: none"> <li>– attributeref can be used to reconnect data items that had a relationship that is represented in the Determination XML in a hierarchical way.</li> <li>– Either attributeref or ruleclasref, but not both, is present in an attribute value CSV.</li> <li>– For a full explanation with examples, see Scenario A and Scenario B.</li> </ul> </li> <li>• The attributeref uses the following path-like structure:               <ul style="list-style-type: none"> <li>– case/{caseID}/determination/{determinationId}/decision/{decisionDate}/{RuleClass}{index_1}/{RuleAttribute}/{index_2}/Item/{index_3}</li> </ul> </li> <li>• The indexes are one-based, that is, the indexing starts with a one, rather than zero-based. The following characteristics apply to the indexes:               <ul style="list-style-type: none"> <li>– index_1 identifies the specific rule class instance within given decision period.</li> <li>– index_2 identifies the specific rule attribute instance within a specific rule class instance.</li> <li>– index_3 identifies the specific item within a specific rule attribute list.</li> </ul> </li> </ul>

Table 3. Columns that are in the Determinations, Decisions, and Attribute value CSV files. (continued)

Applicable Filter Flow tables	Column name	Description
	ruleclassref	<ul style="list-style-type: none"> <li>• A hierarchically name-spaced reference that uniquely identifies a Rule Class instance, across all determinations.</li> <li>• A Filter Flow extraction flattens hierarchical XML data and maps it to a relational format in the following ways:               <ul style="list-style-type: none"> <li>– ruleclassref can be used to reconnect data items that had a relationship that is represented in the Determination XML in a hierarchical way.</li> <li>– Either attributeref or ruleclassref, but not both, is present in an attribute value CSV.</li> <li>– For a full explanation with examples, see Scenario A and Scenario B.</li> </ul> </li> <li>• The ruleclassref uses the following path-like structure:               <ul style="list-style-type: none"> <li>– case/{caseID}/determination/{determinationId}/decision/{decisionDate}/{RuleClass}/{index}</li> <li>– The index value is one-based, not zero-based.</li> <li>– The index identifies the specific rule class instance for the decision period. It supports the scenario of a decision period where there is more than one instance of a specific rule class.</li> </ul> </li> </ul>

There are always reference columns in these CSVs for Case, Determination, Decision. The references follow a hierarchical name spacing pattern and are used to make it easy to join across the CSVs when required. The concept also applies to the Filter Flow database job. In the case of the Filter Flow database job, the CSV file is a database table.

Scenario A and Scenario B and *Appendix C* illustrate where attribute data that is extracted by a Filter Flow job is written. The process is how the extracted data corresponding to an attribute path A is written to file F. For more information about Appendix C, see the *Appendix C: An extract of the rule class that declares the attributes that are requested* related link.

The CSV to use, that is, its name and its structure, is based on how the requested rule attribute is declared in the display rule class.

Scenario A and Scenario B show example attribute paths, the name of the corresponding CSVs, and how the corresponding CSVs are structured.

For a sample of the rule class that declares the attributes that are requested, see the *Appendix C: An extract of the rule class that declares the attributes that are requested* related link.

### Scenario A example attribute paths and CSV

In scenario A, the following attribute paths are specified:

- `StreamlineMedicaidDisplayRuleSet.StreamlineMedicaidIncomeCategory.membersEligibleOrNotMessage`
- `StreamlineMedicaidDisplayRuleSet.StreamlineMedicaidIncomeCategory.householdNonFinancialNotEligibleTimeline`

In the `StreamlineMedicaidIncomeCategory` rule class, these attributes are declared to be non-list values. If both these attribute paths were specified, the value is extracted to the same CSV. The name of the file is `STREAMLINEMEDICAIDDISPLAYRULESET_STREAMLINEMEDICAIDINCOMECATEGORY.csv`. In the file, `caseref`, `determinationref`, and `decisionref` do not apply. The following table shows the structure of the file.

<i>Table 4. STREAMLINEMEDICAIDDISPLAYRULESET_STREAMLINEMEDICAIDINCOMECATEGORY.csv sample</i>		
<b>ruleclassref</b>	<b>memberseligibleornotmessage</b>	<b>householdnonfinancialnoeligibletimeline</b>
case/649/determination/152/decision/20180401/StreamlineMedicaidIncomeCategory/1	Income eligibility was not determined.	True
case/649/determination/152/decision/20180429/StreamlineMedicaidIncomeCategory/1	No members in the household are found to be eligible.	True
case/649/determination/152/decision/20180501/StreamlineMedicaidIncomeCategory/1	No members in the household are found to be eligible.	False

The file `STREAMLINEMEDICAIDDISPLAYRULESET_STREAMLINEMEDICAIDINCOMECATEGORY.csv` has a column `ruleclassref`. The characteristics of the `ruleclassref` column apply in the following ways:

- A hierarchically name-spaced reference to the immediate rule class instance.
- It can be used to identify an instance of the rule class instance across all determinations.
- The `ruleclassref` uses the structure `case/{caseID}/determination/{determinationId}/decision/{decisionDate}/{RuleClass}/{index}`.
- The index is one-based, not zero-based.
- The index identifies the specific rule class instance for the decision period.
- It supports the scenario of a decision period where there is more than one instance of a rule class.
- The Filter Flow flattens the hierarchical XML data.
- The `ruleclassref` can be used to reconnect related hierarchical data.
- The `ruleclassref` provides a simple way for the operator to manually look up values in context in the XML Flow job's CREOLE snapshot data.

## Scenario B example attribute paths and CSV

In scenario B, the following attribute path is specified:

`StreamlineMedicaidDisplayRuleSet.StreamlineMedicaidIncomeCategory.eligibleMembersName`.

In the `StreamlineMedicaidIncomeCategory` rule class, the `eligibleMembersName` rule attribute is defined as list, which means that it can have multiple values within a single coverage period. So, it is extracted into

`STREAMLINEMEDICAIDDISPLAYRULESET_STREAMLINEMEDICAIDINCOMECATEGORY_LIST_ELIGIBLEMEMBERSNAME.csv`. In the file, `caseref`, `determinationref`, and `decisionref` do not apply. The following table shows the structure of the file.

<i>Table 5. STREAMLINEMEDICAIDDISPLAYRULESET_STREAMLINEMEDICAIDINCOMECATEGORY_LIST_ELIGIBLEMEMBERSNAME.csv sample</i>	
<b>attributeref</b>	<b>eligiblemembersname</b>
case/649/determination/152/decision/20180401/StreamlineMedicaidIncomeCategory/1/eligibleMembersName/1/Item/2/0 Mary Mitchell	Mary Mitchell
case/649/determination/152/decision/20180429/StreamlineMedicaidIncomeCategory/1/eligibleMembersName/1/Item/2/1 John Stephens	John Stephens
case/649/determination/152/decision/20180429/StreamlineMedicaidIncomeCategory/1/eligibleMembersName/1/Item/2/0 Mary Mitchell	Mary Mitchell
case/649/determination/152/decision/20180429/StreamlineMedicaidIncomeCategory/1/eligibleMembersName/1/Item/2/2 Peter Brown	Peter Brown

The file

`STREAMLINEMEDICAIDDISPLAYRULESET_STREAMLINEMEDICAIDINCOMECATEGORY_LIST_ELIGIBLEMEMBERSNAME.csv` has an `attributeref` column. The column stores a hierarchical name that is spaced-referenced for the attribute. The structure `case/{caseID}/determination/{determinationId}/decision/{decisionDate}/{RuleClass}/{index_0}/{RuleAttribute}/{index_1}/Item/{index_2}` is used. The following lists the characteristics of the `attributeref` column:

- It can be used to identify an instance of an attribute across all determinations.
- The `attributeref` uses the structure `case/{caseID}/determination/{determinationId}/decision/{decisionDate}/{RuleClass}{index_1}/`
- The indexes are one-based, not zero-based. The following characteristics apply to the indexes:
  - `index_1` identifies the specific rule class instance within a decision period.
  - `index_2` identifies the specific rule attribute instance within a rule class instance.
  - `index_3` identifies the specific item within a rule attribute list.
- The Filter Flow flattens the hierarchical XML data.
- The `attributeref` can be used to reconnect related hierarchical data.
- The `attributeref` provides a simple way for the operator to manually look up values in context in the XML Flow job's CREOLE snapshot data.

In the Spring Profile properties file, to extract the following attribute the operator must specify the `spm.extract.creoledeterminations.displayrules.extractlistattributes` property in the following way:

```
spm.extract.creoleDeterminations.displayrules.extractlistattributes=\
    StreamlineMedicaidDisplayRuleSet.StreamlineMedicaidIncomeCategory.eligibleMembersName
```

## Related concepts

[Appendix C: An extract of the rule class that declares the attributes that are requested](#)

The sample in this topic shows a snippet of a `StreamlineMedicaidIncomeCategory` rule class.

## Related information

[Decision Periods](#)

[CaseDecision](#)

[The Domain Language of Batch](#)

## The output directory for the Filter Flow CSV job

The path to the directory where the Filter Flow CSV job writes the CSVs is configurable. It is a configuration property `spm.extract.csvs.output.dir` in the Spring Profile (`applications.properties`).

For information about all the properties of the Spring Profile, see the [Editing the Spring Profile \(application.properties\)](#) related link.

An example of the directory structure that is beneath the output directory is:

```
├── 66
│   ├── DECISIONS.csv
│   ├── DETERMINATIONS.csv
│   ├── STREAMLINEMEDICAIDDISPLAYRULESET_STREAMLINEMEDICAIDINCOMECATEGORY.csv
│   ├── STREAMLINEMEDICAIDDISPLAYRULESET_STREAMLINEMEDICAIDINCOMECATEGORY_LIST_ELIGIBLEMEMBERSINCOMEPERCENTAGE.csv
│   ├── STREAMLINEMEDICAIDDISPLAYRULESET_STREAMLINEMEDICAIDINCOMECATEGORY_LIST_ELIGIBLEMEMBERSNAME.csv
│   ├── STREAMLINEMEDICAIDDISPLAYRULESET_STREAMLINEMEDICAIDINCOMECATEGORY_LIST_SMINCOMEELIGIBILITYSUBSCREEN.csv
│   └── STREAMLINEMEDICAIDDISPLAYRULESET_STREAMLINEMEDICAIDINCOMECATEGORY_LIST_SMMEMINCOMECATEGORYSUBSCREENS.csv
├── 67
│   ├── DECISIONS.csv
│   ├── DETERMINATIONS.csv
│   ├── STREAMLINEMEDICAIDDISPLAYRULESET_STREAMLINEMEDICAIDINCOMECATEGORY.csv
│   ├── STREAMLINEMEDICAIDDISPLAYRULESET_STREAMLINEMEDICAIDINCOMECATEGORY_LIST_ELIGIBLEMEMBERSINCOMEPERCENTAGE.csv
│   ├── STREAMLINEMEDICAIDDISPLAYRULESET_STREAMLINEMEDICAIDINCOMECATEGORY_LIST_ELIGIBLEMEMBERSNAME.csv
│   ├── STREAMLINEMEDICAIDDISPLAYRULESET_STREAMLINEMEDICAIDINCOMECATEGORY_LIST_SMINCOMEELIGIBILITYSUBSCREEN.csv
│   └── STREAMLINEMEDICAIDDISPLAYRULESET_STREAMLINEMEDICAIDINCOMECATEGORY_LIST_SMMEMINCOMECATEGORYSUBSCREENS.csv
```

Beneath the `spm.extract.csvs.output.dir` directory, there is a set of directories with names that correspond to the job instance IDs. For example, in the listing above, the directory 66 contains files that are written by executions of the Filter Flow CSV job with Job Instance ID 66.

**Note:** The IBM Social Program Management Data Extractor does not encrypt the data that is written to the CSV files. For customers who use Filter Flow CSV jobs, customers can configure the `spm.extract.csvs.output.dir` directory to point to a location that is either file system encrypted or that corresponds to a mounted volume that uses block level encryption.

## Related information

[Editing the Spring Profile](#)

## Filter Flow database job

---

The Filter Flow CSV job and the Filter Flow database job are similar and are used for the same purpose. The key difference between the Filter Flow database job and the Filter Flow CSV job is the data that is extracted by the Filter Flow database job is written to a set of database tables that the customer creates in the customer's data warehouse.

A successful Filter Flow database batch job writes to several database tables. The Filter Flow database batch job includes the following characteristics:

- One or more attribute value database tables to which the extracted data, corresponding to the attributes that are specified by the operator, is exported. The way that attributes are exported is determined in the following ways:
  1. Attributes that are not defined as lists in their corresponding rule class are always exported to a destination that corresponds to that rule class. The table names are in the format: `{RuleSet}_{RuleClass}`. An example is `STREAMLINEMEDICAIDDISPLAYRULESET_STREAMLINEMEDICAIDINCOMECATEGORY`.
  2. Attributes that are defined as lists in their corresponding rule class are always exported to a destination that corresponds to that attribute only. The table names are in the format `{RuleSet}_{RuleClass}_LIST_{RuleAttribute}`. An example is `STREAMLINEMEDICAIDDISPLAYRULESET_STREAMLINEMEDICAIDINCOMECATEGORY_LIST_ELIGIBLEMEMBERSNAMES`.
- Attributes consist of a single database table that contains data that relates to the determinations that are extracted. The data that is written to the table is independent of the specified attribute paths. The database table name is `DETERMINATIONS`.
- Attributes consist of a single table that contains data that relates to the decisions in the determinations that are extracted. The data that is written to the table is independent of the specified attribute paths. The table name is `DECISIONS`.

The naming specifications that apply to the CSV files and their schemas also apply to the naming of output tables and their schema for the Filter Flow database job. For more information about the Filter Flow CSV job, see the *Filter Flow CSV job* related link.

**Note:** The Filter Flow CSV job can be used to create and write to files on the local file system. However, the Filter Flow database jobs are used to write to pre-existing database tables in the data warehouse. The tool cannot be used to create the tables in the data warehouse. However, the tool can be used to generate the DDL to create the tables and to drop the tables. For more information, see the *Generating the Filter Flow attribute value database tables creation and drop DDL* related link.

### Related concepts

[Generating the Filter Flow attribute value database tables creation and drop DDL](#)

Generating the creation and drop DDL for the Filter Flow attribute value tables is a function of the IBM Social Program Management Data Extractor.

### Related information

[Filter Flow CSV job](#)

## Filter Flow compaction limit

Compaction is a configuration setting to enable a behavior for Filter Flow database job executions. Use compaction to reduce the number of database write operations to the Filter Flow extraction tables.

Extraction jobs have different ItemWriter implementations for writing out chunks of items of type `O`. For more information, see the *Batch jobs: job types and their purposes* related link.

For the Filter Flow database jobs, the number of calls to its ItemWriter's `write` operation is not equal to the number of database calls to write data to the Data Warehouse for the following reasons:

- The process operation of the ItemProcessor for the Filter Flow jobs is used to take in a single determination object of type `I` and produces a single result object of type `O`.

- A single *O* object is used to model all the relational data that is due to be written to two or more Filter Flow tables.
- When Filter Flow database job calls its ItemWriter to write out the data for a single *O* item, it performs database write operations against two or more Filter Flow tables in the following ways:
  - The job writes data to the DETERMINATION and DECISIONS filter tables.
  - Data is written to the attribute value tables only if the determination's XML BLOB contained the requested data attributes for one or more decision periods in that determination.

The ItemWriter's `write` operation for a Filter Flow job, however, operates on a collection or 'chunk' of these *O* objects under the following conditions:

- The size of the chunks is up to `chunkSize`.
- The call to the ItemWriter's `write` operation is contained within a transaction boundary. At the end of chunk, the writes to the database are committed if there were no unskippable errors or rolled back if there were unskippable errors. This chunk-oriented processing is used to enable a behavior that is known as compaction for the Filter Flow database job.

The IBM Social Program Management Data Extractor is used to exploit the chunk-oriented approach to reduce the number of round-trip calls that write data to the Data Warehouse database, by enabling the compaction behavior.

If compaction is not enabled when the Filter Flow Database ItemWriter's `write` operation is called, then for each item in a chunk, the ItemWriter makes separate database write calls to every Filter Flow table for which that item mapped data. So, for a call to the ItemWriter's `write` operation for each Filter flow table, there is a maximum `#chunkSize` database calls to write data to it.

If compaction is enabled, then the `write` operation the Filter Flow database ItemWriter is used to buffer the data on a per Filter Flow table basis. A database write operation to a table is performed only for every *N* rows of data that is buffered for that table. Immediately before it completes a chunk, the `write` operation for the ItemWriter performs one extra database write per Filter Flow table for any rows that remain in that table's buffer, if any. The number *N* is called the compaction limit. For example, if there was a chunk size of 25 and a compaction limit of 10, then there are, on average, three calls per chunk, to write data to the DETERMINATIONS table. That is, there are two writes of 10 rows of data, and one write for the remaining five rows.

### Scenario examples to illustrate the benefit of using the compaction limit

#### Scenario 1

The following conditions apply to this scenario:

- Only one attribute path (a.p) is specified, the a.p. corresponds to rule attribute that is a list.
- The a.p. is used to attribute data when it is extracted. The data is written to an attribute value (AV) table.
- On average, there are three decision periods per determination.
- On average, three items are extracted for the decision period that corresponds to the specified a.p.
- The `chunkSize` is 20.

So, it is expected that a call to the ItemWriter's `write` operation generates the results that are shown in the following table. The numbers in the table are the same regardless of whether compaction is enabled.

<i>Table 6. The number of tables that are written for each Filter Flow table</i>	
<b>Table</b>	<b>Rows written</b>
DETERMINATIONS	20
DECISIONS	60 (20*3)
AV	180 (20*3*3)

## Scenario 2

The benefits of compaction are that with a properly tuned compaction limit the Filter Flow Database ItemWriter is used to write out the same volume of data over fewer database calls. As a result, the time that is spent writing data to the Data Warehouse is reduced. If compaction is disabled, the ItemWriter's write operation that is operating on a chunk size of 20 produces the metrics in the following table.

*Table 7. The number of database write operations for a chunk size of 20 with compaction disabled*

Table name	Number of database writes	Average number of rows that are written for each database write
DETERMINATIONS	20	1
DECISIONS	20	3
AV	20	9
Total number of database writes	60	N/A

**Note:** There can be less than 20 database writes to the AV table because there might be determinations that do not have the specified attribute set for any period in that determination.

## Scenario 3

If compaction is enabled with chunk size 20 and compaction limit 10, the metrics in the following table are produced.

*Table 8. The number of database write operations for a chunk size of 20 and a compaction limit of 10*

Table name	Number of database writes	Average number of rows that are written for each database write
DETERMINATIONS	2	10
DECISIONS	6	10
AV	18	10
Total number of database writes	26	N/A

## Scenario 4

If compaction is enabled with chunk size 20 and compaction limit 20, the metrics in the following table are produced.

*Table 9.*

Table name	Number of database writes	Average number of rows that are written for each database write
DETERMINATIONS	1	20
DECISIONS	3	20
AV	9	20
Total number of database writes	13	N/A

The compaction limit is set as a property in the Spring Profile. The property name is `spm.extract.filterflow.compactionlimit`. For more information about the `spm.extract.filterflow.compactionlimit` property, see the *Filter Flow job configuration properties* related link.

## Related information

[Batch jobs: job types and their purposes](#)

[Filter Flow job configuration properties](#)

## Monitoring job progress in the log

---

When the operator runs the IBM Social Program Management Data Extractor in either server or non-server mode, the log output is written to both the standard output and to the `dataExtractor.log` file in the same directory where the IBM Social Program Management Data Extractor was started.

For more information about the log output, see the *Running the IBM Social Program Management Data Extractor* related link. The tool's log displays a message when the operator starts the tool and the tool logs output when tool functionality is called.

### Job executions log output when the job runs

A job execution logs a progress message every  $N$  chunks. The  $N$  is configurable through the `spm.extract.chunkloginterval` property in the Spring Profile (`application.properties`). For more information about the properties of the Spring Profile, see the *Editing the Spring Profile (application.properties)* related link.

The following samples show examples of progress messages:

```
2019-07-21 02:50:54.332 [INFO ] [SimpleAsyncTaskExecutor-1] DeterminationStepLifeCycleListener
- >>logProgressAtInfo:Read Count=1, Read Skip Count=0, Process Skip Count=0, Write Count=1,
Write Skip Count=0, Commit Count=1, Roll Back Count=0, Calculated Read Progress=25.0
```

```
2019-07-21 02:50:55.381 [INFO ] [SimpleAsyncTaskExecutor-1] DeterminationStepLifeCycleListener
- >>logProgressAtInfo:Read Count=2, Read Skip Count=0, Process Skip Count=0, Write Count=2,
Write Skip Count=0, Commit Count=2, Roll Back Count=0, Calculated Read Progress=50.0
```

```
2019-07-21 02:50:56.416 [INFO ] [SimpleAsyncTaskExecutor-1] DeterminationStepLifeCycleListener
- >>logProgressAtInfo:Read Count=3, Read Skip Count=0, Process Skip Count=0, Write Count=3,
Write Skip Count=0, Commit Count=3, Roll Back Count=0, Calculated Read Progress=75.0
```

```
2019-07-21 02:50:57.502 [INFO ] [SimpleAsyncTaskExecutor-1] DeterminationStepLifeCycleListener
- >>logProgressAtInfo:Read Count=4, Read Skip Count=0, Process Skip Count=0, Write Count=4,
Write Skip Count=0, Commit Count=4, Roll Back Count=0, Calculated Read Progress=100.0
```

When a job execution finishes, it prints a summary. The message is a step execution summary rather than a job execution summary, which is a known issue. For jobs that support parallelization, that is, XML Flow and Filter Flow database, each parallel step execution logs a summary message. The log message includes statistics that relate to the step execution. The following sample shows an example summary:

```
2019-07-21 02:50:58.458 [INFO ] [SimpleAsyncTaskExecutor-1] DeterminationStepLifeCycleListener
- >>logStepFinished:
```

```
-----
Job Name: 'FilterFlowCsv'
Instance: 4
Execution: 4
Finished Step: 'filterFlowCSVStep0'
Step Summary: 'StepExecution: id=4, version=6, name=filterFlowCSVStep0, status=COMPLETED,
exitStatus=COMPLETED, readCount=4, filterCount=0, writeCount=4 readSkipCount=0,
writeSkipCount=0, processSkipCount=0, commitCount=5, rollbackCount=0'
```

```
2019-07-21 02:50:58.462 [INFO ] [SimpleAsyncTaskExecutor-1] DeterminationStepLifeCycleListener
- >>logReaderExhaustedMessages:We have finished querying the backlog.The number of items
successfully read was 4.We were within the limit that was explicitly set for the reader 250.
```

## Related information

[Running the IBM Social Program Management Data Extractor](#)

[Editing the Spring Profile](#)

## Functions of the IBM Social Program Management Data Extractor that relate to the batch jobs

---

The operator can call the batch-related functions. The batch-related functions can be used to, for example, schedule a job execution, restart a job execution, stop a job execution, or report a job execution status.

### Scheduling a job execution

In server mode or non-server mode, tool operators can schedule an execution of one of the three batch job types.

For more information about the scheduling a job execution, see the *Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode* related link.

**Note:** One of the inputs for scheduling a job, is a parameter **name**. The parameter is intended as an easy way for operators to identify a job instance. If the jobs are of the same type, operators cannot schedule two job instances with the same value for the **name** parameter because the **name** job parameter is an identifying parameter in the domain language of Spring Batch. For more information about an identifying parameter, see the *The Domain Language of Batch* related link.

#### Related information

[Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode](#)

[The Domain Language of Batch](#)

### Restarting a job execution

In server mode or non-server mode, tool operators can use the failed job's execution ID to restart a failed execution of one of the three batch job types.

#### Jobs that can be restarted

- Only a job with a FAILED status can be restarted. Jobs that fail are ones that encounter  $N$  number of failures that are caused by a failure to process determinations. The number  $N$  is configurable by the `spm.extract.skiplimit` property in the Spring Profile. For more information about the Spring Profile, see the *Editing the Spring Profile (application.properties)* related link.

#### Execution ID

- For each new job that a tool operator schedules, both an instance ID and an execution ID are generated. When an operator restarts a failed job, the new job execution uses the same job instance ID as the original failed execution but uses a different execution ID. The new job execution resumes where the failed execution ended. For example, a job execution that appears to fail after the execution reads 100 determinations even though there are more determinations that matched that job's criteria. A new execution for the same job instance does not try to read or process those first 100 determinations. Instead, the new execution starts at that 101st determination that the original execution was due to process before that attempt failed.
- If a job execution fails, the `dataExtractor` log file lists a printed summary of the job execution:

```
Job Name: 'XmlFlow'  
Instance: 43  
Execution: 43  
Finished Step: 'xmlFlowStep0'  
Step Summary: 'StepExecution: id=43, version=3, name=xmlFlowStep0, status=FAILED,  
exitStatus=FAILED, readCount=30, filterCount=0, writeCount=20 readSkipCount=0,  
writeSkipCount=0, processSkipCount=0, commitCount=2, rollbackCount=2'
```

In the example, the execution ID is shown as 43. The operator can restart the job execution by calling the `restart a job` function. For more information, see the *Parameters for the IBM Social Program Management Data Extractor's functions* related link.

If the restarted job succeeds, a dataExtractor log file is displayed:

```
Job Name: 'XmlFlow'  
Instance: 43  
Execution: 44  
Finished Step: 'xmlFlowStep0'  
Step Summary: 'StepExecution: id=44, version=4, name=xmlFlowStep0, status=COMPLETED,  
exitStatus=COMPLETED, readCount=22, filterCount=0, writeCount=22 readSkipCount=0,  
writeSkipCount=0, processSkipCount=0, commitCount=3, rollbackCount=0'
```

As the log file shows, the summary of the restarted job uses the same job instance ID as the failed execution, but shows a new job execution ID.

### Rows that are written by the same execution have the same value in the JOBEXECUTIONID column

- The Extraction database tables, that is DETERMINATIONXML, DETERMINATIONS, DECISIONS, and the Filter Flow attribute value tables, have JOBINSTANCEID and JOBEXECUTIONID columns. For more information, see the *Creating the database tables* related link. A characteristic of restarted job executions is that rows that are written to the Extraction tables by executions that belong to the same job instance have the same value in the JOBINSTANCEID column. Rows that are written by the same execution have the same value in the JOBEXECUTIONID column. Using the preceding example, a customer might use the DETERMINATIONXML table to identify the following characteristics of the job:
  - It took more than one execution to complete job instance 43.
  - The execution IDs for the executions, that is, 43 and 44.
  - The rows that each execution wrote to that table.

```
SELECT JOBEXECUTIONID,COUNT(*) AS WRITECOUNT FROM DW.DETERMINATIONXML  
WHERE JOBINSTANCEID=43  
GROUP BY JOBEXECUTIONID;  
JOBEXECUTIONID WRITECOUNT  
-----  
43                20  
44                22
```

### Directories and job instance IDs

- Beneath the directory that is specified as the value of the `spm.extract.csvs.output.dir` property in the Spring Profile, there is a set of one or more directories with names that correspond to the job instance IDs. For example, the directory 1 contains files that are written by executions of the Filter Flow CSV job with Job Instance ID 1.

For restarted job executions, the new job execution appends to the same CSVs written to the previous failed job execution with which they share a job instance ID. The rows that are written by the new job execution have the same value in the JOBINSTANCEID column as the rows that were written by the failed execution but have a different value in the JOBEXECUTIONID column.

### Scheduling a job of the same type with the same name as the job that failed

- To restart a failed job execution, IBM recommends that operators use the function that accepts a job execution ID. However, if the operator tried to schedule a job of the same type with the same name as the job that failed, the effect is to restart the failed job by creating a new job execution with the same instance ID but a different execution ID. In such a scenario, the new job execution runs by using the job parameters that were specified the second time rather than the parameters that were specified the first time. It is problematic when the job parameters that were specified the second time are used because it can change the criteria that is used to select the determinations to extract.

For example, an operator schedules a job with **MyJob101** as the name parameter, but the operator does not specify the **startDate** and **endDate** parameters. So, only determinations that were created within the last *N* days are to be extracted, that is, when the **startDate** or the **endDate** are not specified, the **startDate** and the **endDate** are inferred by the tool.

Where the first attempt to run the job fails and the operator uses a restart by name option instead of the recommended restart by execution ID option, the **startDate** and **endDate** values are used by the new execution. The values are used regardless of whether the operator omits the **startDate**

and **endDate** or the operator specifies a **startDate** and **endDate** on the restart. The values from the original execution are not used, which might lead to undesirable effects. For example, the new execution skips determinations because the selection criteria are changed. So, IBM recommends that operators use the restart by execution ID method.

### Related information

[Editing the Spring Profile](#)

[Parameters for the IBM Social Program Management Data Extractor's functions](#)

[Creating the database tables](#)

## Per determination retry

Operators can use the retry operation to explicitly specify the determinations that the operator wants to extract by specifying the set of CREOLECASEDETERMINATION.CREOLECASEDETERMINATIONIDs that is required.

When the operator is extracting data from an individual determination, the operator can encounter errors. The error might be regarded as a tolerable failure to the extraction job. A job execution can tolerate *X* such errors, where *X* is the effective value of the `spm.extract.skiplimit` configuration property. For more information about errors, see the *Restarting a job execution* and the *Editing the Spring Profile (application.properties)* related links.

The failed extractions can be identified by using the following information:

- For the Filter Flow CSV or Filter Flow database, operators use the following code:

```
[WARN ] 2019-08-06 12:20:22.552 [SimpleAsyncTaskExecutor-1] FilterFlowJobItemListener -  
>>onProcessError 45018 Failed to parse data for CREOLEdetermination with ID: 45018
```

- For the XML Flow, operators use the following code:

```
[WARN ] 2019-08-06 12:22:23.483 [SimpleAsyncTaskExecutor-1] XMLFlowJobItemListener -  
>>onProcessError 45018 Failed to parse data for CREOLEdetermination with ID: 45018
```

As the log message shows, the CREOLECASEDETERMINATION.CREOLECASEDETERMINATIONID for the problem determination is logged to the `dataExtractor.log` file and to the standard output of the Data Extractor Java process.

Certain issues can be rectified. For example, bad test data on a test database. Where the test data is rectified, the determination can be extracted successfully if the operator schedules a new job instance of the same type and with the same parameters except for the **name** parameter. The **name** parameter must be different. After the operator fixes the cause of the error, the operator might want to extract the only applicable determinations.

However, IBM recommends that customers avoid performing either of the following steps just to extract, for example, 10 determinations that were skipped:

- Scheduling a job execution with a new name, but the same criteria as the original execution.
- Scheduling a job execution with a new name, but with changed criteria like a narrower date range.

In both scenarios, scheduling such jobs results in extra load in processing hundreds, and possibly thousands, of determinations, most of which were already extracted to get the 10 determinations that were skipped.

Operators can use the retry operation to explicitly specify the determinations that the operator wants to extract by specifying the set of CREOLECASEDETERMINATION.CREOLECASEDETERMINATIONIDs. The operator can use the `dataExtractor.log` to identify the set of CREOLECASEDETERMINATION.CREOLECASEDETERMINATIONIDs to extract.

The functionality can be used for all three types of extraction jobs.

In server mode, the retry operations are displayed as separate operations on the **JobLaunchingController**. In non-server mode, the command that is used is similar to the command that is used to schedule new jobs, but it accepts a different job parameter. While the operation is listed as a

separate function, the operation reuses the existing batch job types. However, there are six key differences. The six distinctions between an XML Flow retry job and a standard XML Flow job are:

1. The retry job does not use an SQL query to identify the set of determinations to extract.
2. The retry job ignores the **gridSize** parameter and the `spm.extract.gridsize` configuration property. A grid-size of one is assumed.
3. The retry job ignores the **maxNumItems** parameter and the `spm.extract.maxitemcount` configuration property.
4. The retry job ignores the **pageSize** parameter and the `spm.extract.pagesize` configuration property.
5. The retry job ignores the **fetchSize** parameter and the `spm.extract.fetchsize` configuration property.
6. While the IBM Social Program Management Data Extractor does not prevent the restarting of a retry job, IBM does not support the operation.

### Related information

[Restarting a job execution](#)

[Editing the Spring Profile](#)

## Stopping a job execution

In server mode or non-server mode, tool operators can use the running job's execution ID to stop a running execution of one of the three batch job types.

For more information about how to stop a job execution, see the *Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode* related link.

### Related information

[Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode](#)

## Reporting a job execution status

In server mode or non-server mode, tool operators can use the running job's execution ID to get the status of a running execution of one of the three batch job types.

The following summary looks like the summary that is returned:

```
JobExecution: id=288, version=1, startTime=2019-07-17 11:19:15.644, endTime=null,
lastUpdated=2019-07-17 11:19:15.736, status=STARTED,
exitStatus=exitCode=UNKNOWN;exitDescription=, job=[JobInstance: id=228, version=0,
Job=[FilterFlowDb]], jobParameters=[{includeManualEligibilityCheck=true, gridSize=1,
chunkSize=10, endDate=87804100800000, fetchSize=10, fieldsNotExplicitlySet=chunkLog
gingInterval,chunkSize,endDate,fetchSize,gridSize,includeManualEligibilityCheck,pageSize,product
TypeCode,name=FilterFlowDb_17-07-2019_11-19-14.09, pageSize=10, maxNumItems=100,
chunkLoggingInterval=2, startDate=1404100800000, productTypeCode=PT26304}]
```

For more information about how to call the function, see the *Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode* related link.

### Related information

[Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode](#)

## Abandoning a job execution

In server mode or non-server mode, operators can use the execution ID of the running job to abandon a job execution of one of the three batch job types.

The Spring Batch *Configuring and Running a Job* documentation includes the following description for aborting a job: "If the process died ("kill -9" or server failure) the job is, of course, not running, but the JobRepository has no way of knowing because no-one told it before the process died. You have to tell it manually that you know that the execution either failed or should be considered aborted."

For more information about how to call the function to abandon a job execution, see the *Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode* related link.

**Related information**

[Spring Batch: Configuring and Running a Job](#)

[Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode](#)

---

## Chapter 6. Other functions of the IBM Social Program Management Data Extractor

The IBM Social Program Management Data Extractor can be used for functions other than batch jobs. The other functions include searching for the Filter Flow attribute paths, generating the Filter Flow attribute value database tables creation and drop DDL, and generating schema definitions for Determination BLOB XML.

---

### Searching for the Filter Flow attribute paths

Search the set of display rule attributes in the rule set definitions that can be queried. The output of the function is a listing of attribute paths.

The IBM Social Program Management Data Extractor computes the list by querying the rules meta model, an in-memory data structure that models the set of rule set definitions that are stored in the Social Program Management database.

The same data structure and the properties `spm.extract.creoledeterminations.displayrules.extractlistattributes`, `spm.extract.dw.datasource.platform` in the Spring Profile, and the `ShortNames.properties` determine the Filter Flow attribute value table outputs. For more information, see the *Generating the Filter Flow attribute value database tables creation and drop DDL* related link.

Spring Profiles are a way to segregate application configurations so that the configuration is available only in certain environments. Operators can use the attributes paths from the output of the search function as the value of the `spm.extract.creoledeterminations.displayrules.extractlistattributes` property in the Spring Profile. For more information, see the *Filter Flow CSV job* related link.

The input to the search function is a regular expression, for example `[.]*ncomePer[^\.]+$`.

The response to a query is a text output that shows the results, as shown in the following example:

```
Filtered Attribute Paths matching pattern '[.]*ncomePer[^\.]+$':-
-----
InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceDetailsCategory.fplTierForIncomePercentage,
InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceDetailsCategory.incomePercentageofFPL,
InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceIncomeCategory.incomePercentageFPLTimeline,
StreamlineMedicaidDisplayRuleSet.StreamlineMedicaidIncomeCategory.eligibleMembersIncomePercentage
-----
Found 4 Attribute Paths
-----
```

For more information about how to call the function and for details about how to specify the parameter or parameters of the function, see the *Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode* and the *Parameters for the IBM Social Program Management Data Extractor's functions* related links.

#### **Related concepts**

[Generating the Filter Flow attribute value database tables creation and drop DDL](#)

Generating the creation and drop DDL for the Filter Flow attribute value tables is a function of the IBM Social Program Management Data Extractor.

[Filter Flow CSV job](#)

[Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode](#)

[Parameters for the IBM Social Program Management Data Extractor's functions](#)

## Generating the Filter Flow attribute value database tables creation and drop DDL

Generating the creation and drop DDL for the Filter Flow attribute value tables is a function of the IBM Social Program Management Data Extractor.

For more information about generating the creation and drop DDL for the Filter Flow attribute value tables, see the *Creating the database tables* related link. For information about how to call the functionality in server mode and non-server mode, see the *Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode* related link.

The schema of the tables can be generated only when the following criteria is specified:

1. The rule set definitions that are configured for that Social Program Management system.
2. The attributes that the customer wants to extract from the determinations as part of a Filter Flow job, which requires that the customer edit a property in `application.properties`.

The DDL that is generated varies depending on the database vendor. The tool generates the DDL that is specific for the database vendor type of the Data Warehouse database.

For example, the operator might edit their Spring Profile (`application.properties`) according the following criteria:

1. The Data Warehouse data source is pointing to an Oracle database.
2. The operator is using the default version of the INSURANCEASSISTANCEDISPLAYRULESET Rule Set Definition.
3. The operator configured their Spring Profile as:

```
spm.extract.product.typecode=PT26304
spm.extract.creole.determinations.displayrules.extractlistattributes=\
InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceDetailsCategory.caseParticipantRoleID,\
InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceDetailsCategory.isPassedNonFinancialRules,\
```

Consequently, the output of the Generate Filter Flow attribute value tables Creation DDL function is as defined in the following sample:

```
CREATE TABLE INSURANCEASSISTANCEDISPLAYRULESET_INSURANCEASSISTANCEDetailSCATEGORY (
CASEPARTICIPANTROLEID DOUBLE,
CASEREF VARCHAR(1024),
DECISIONREF VARCHAR(1024),
DETERMINATIONREF VARCHAR(1024),
EXTRACTIONDATE DATE,
ISPASSEDNONFINANCIALRULES BOOLEAN,
JOBEXECUTIONID BIGINT,
JOBINSTANCEID BIGINT,
RULECLASSREF VARCHAR(1024)
);
```

The output of the Generate Filter Flow attribute value tables Drop DDL function is:

```
DROP TABLE INSURANCEASSISTANCEDISPLAYRULESET_INSURANCEASSISTANCEDetailSCATEGORY CASCADE
CONSTRAINT PURGE;
```

### Alternative implementation for Oracle databases

Issues with some versions of Oracle that use the preceding DDL can occur. For more information about the likely issues, see the *Issues associated with the length of a generated table name* related link. However, customers can shorten the table name length and column name length by configuring a `ShortNames.properties` and placing it in a directory on the `loader.path`.

By adding to the example scenario of the preceding steps 1-3, then configuring the `ShortNames.properties` is:

```
INSURANCEASSISTANCEDISPLAYRULESET=IA
INSURANCEASSISTANCEDENIED=DND
INSURANCEASSISTANCEDetails=DTLS
INSURANCEASSISTANCEINCOME=INCOME
INSURANCEASSISTANCE=IA
PROGRAMNAMETIMELINE=PRGMNAMETL
COUNTEDMEMBERMAGITIMELINE=COUNTMEMTL
ASSISTANCETYPELIST=ASTLIST
DENIED=DND
PROGRAM=PRGM
CATEGORY=CAT
DETAILS=DTLS
INSURANCEASSISTANCEDENIED=DND
INSURANCEASSISTANCEDetails=DTLS
INSURANCEASSISTANCEINCOME=INCOME
INSURANCEASSISTANCE=IA
PROGRAMNAMETIMELINE=PRGMNAMETL
COUNTEDMEMBERMAGITIMELINE=COUNTMEMTL
ASSISTANCETYPELIST=ASTLIST
DENIED=DND
PROGRAM=PRGM
CATEGORY=CAT
DETAILS=DTLS
```

The output of the Generate Filter Flow attribute value tables Creation DDL function is:

```
CREATE TABLE IA_DTLSCATEGORY(
CASEPARTICIPANTROLEID DOUBLE,
CASEREF VARCHAR(1024),
DECISIONREF VARCHAR(1024),
DETERMINATIONREF VARCHAR(1024),
EXTRACTIONDATE DATE,
ISPASSEDNONFINANCIALRULES BOOLEAN,
JOBEXECUTIONID BIGINT,
JOBINSTANCEID BIGINT,
RULECLASSREF VARCHAR(1024)
);
```

The output of the Generate Filter Flow attribute value tables Drop DDL function is: `DROP TABLE IA_DTLSCATEGORY CASCADE CONSTRAINT PURGE;`

### Drop DDL function output

Generating the drop DDL for the Filter Flow database tables is a function of the IBM Social Program Management Data Extractor.

If the operator configured the Spring Profile (`application.properties`) and the `ShortNames.properties` as specified in the preceding section, then the drop DDL functionality returns:

```
DROP TABLE IA_DTLSCATEGORY CASCADE CONSTRAINT PURGE;
```

For more information about how to call the functionality, see the *Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode* related link.

### Related information

[Creating the database tables](#)

[Running the Data Extractor's functions in server mode and non-server mode](#)

[Issues associated with the length of a generated table name](#)

## Issues associated with the length of a generated table name

Tool operators might see an issue where the table names in the generated Filter Flow creation DDL reaches the limit for table name length for their selected database vendor. As an interim solution, the Short Names Substitution functionality is used on table names and column names.

The algorithm for generating the name of an output database table uses a structured naming scheme, based on the attribute path. For more information, see the [Generating the Filter Flow attribute value database tables creation and drop DDL](#) related link. As the attribute path becomes longer, so does the generated table name. Social Program Management permits the following rule set and rule class specifications:

- Rule set names up to 255 characters.
- Rule class names up to 255 characters.
- Rule set attributes up to 255 characters.

Therefore, tool operators might see an issue where the table names in the generated Filter Flow creation DDL reaches the limit for table name length for their selected database vendor. For example:

- Before Oracle 12cR2, the limit for database object names is 30 characters.
- After Oracle 12cR2, the limit is 128 characters. For more information, see the [Increase maximum identifier length from 30 characters to 60 or more](#) related link.
- For Db2 11.1, the limit is also 128 characters. For more information, see the [SQL and XML limits](#) related link.

To enable the Filter Flow database job and the Filter Flow Tables DDL API to conform to the short name substitution policy, the operator can include the `ShortName.properties` file on the `loader.path` when the operator runs the tool. For more information, see the [The ShortNames.properties file](#) related link. For more information about the `loader.path`, see the [Running the IBM Social Program Management Data Extractor](#) related link.

### Related concepts

[Generating the Filter Flow attribute value database tables creation and drop DDL](#)

Generating the creation and drop DDL for the Filter Flow attribute value tables is a function of the IBM Social Program Management Data Extractor.

[Increase maximum identifier length from 30 characters to 60 or more](#)

### Related information

[SQL and XML limits](#)

[The ShortNames.properties file](#)

[Running the IBM Social Program Management Data Extractor](#)

## Generating schema definitions for Determination BLOB XML

---

Use the XSD Generation functionality to provide a definition of a particular Determination XML for a specific product on a specific date.

The operator can use the XML Flow jobs to extract the CREOLE Case Determination Data BLOB and convert it into a readable XML format. However, it can still be difficult to anticipate the structure of the XML. It can also be difficult to anticipate what the operator can either query by using the XQuery on the DETERMINATIONXML or can extract by using the Filter Flow jobs. For more information about XQuery, see the [XQuery](#) related link.

For the extracted CREOLE Case Determination Data BLOB that is converted into a readable XML format, the XML differs for every product and for every period of that product.

The XSD Generation functionality provides the definition by using the following XSD files that together define the full structure of the Determination XML:

- Statics XSD

- Product Period XSD
- RuleSet XSD

The following table lists the readable XML format and the XSD files that are used to define the structure of the Determination XML.

Table 10. The XML format and the XSD files used.	
XML	XSD
<pre>&lt;Determination&gt; &lt;...&gt;   &lt;...&gt;   &lt;/...&gt; &lt;/...&gt; &lt;...&gt;   &lt;...&gt;   &lt;...&gt;</pre>	<p>Statics</p> <ul style="list-style-type: none"> <li>• Portions of the XML do not change. So, the portions have their own XSDs that can be downloaded one time and reused.</li> </ul>
<pre>&lt;DecisionDetails&gt;</pre>	<p>Product period</p> <ul style="list-style-type: none"> <li>• The product period XSDs define the single element that connects the Statics to RuleSets XSD. The single element, DecisionDetails can contain any of the configured Display Categories.</li> </ul>
<pre>&lt;CustomDisplayCategory&gt;   &lt;...&gt;   &lt;/...&gt; &lt;/...&gt; &lt;CustomDisplayCategory&gt;</pre>	<p>RuleSet</p> <ul style="list-style-type: none"> <li>• Three RuleSet XSDs define each display category that is configured on the rule set, that is, the superset of all periods.</li> </ul>

### Related information

[An introduction to XQuery](#)

## Generating schema definitions

For a product and period, actual XSD content can be generated by using the XSD Generation APIs.

List APIs also exist to support finding the right XSD APIs. List APIs return JSON formatted content that attempts to be self-describing. The `_links` portion of each item does not contain real API references. Instead, it contains an indicative URL representation of the API to call to get further information.

For more information about how to call the functionality, see the *Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode* related link.

In the order that an operator needs to call them, the high-level functionality consists of seven operations.

### 1. Get a list of all CER products

The proceeding operation shows a portion of the output of the API. For each product, it includes the `productID`, the `productTypeCode`, and the `productName`. The `productID` can be used as input to the next operation.

```
[
  {
    "id": "26301",
    "typeCode": "PT26301",
    "name": "PN26301",
    "_links": {
      "self": {
```

```

    "href": "/xsd/cer/products/26301"
  }
},
{
  "id": "26304",
  "typeCode": "PT26304",
  "name": "PN26304",
  "_links": {
    "self": {
      "href": "/xsd/cer/products/26304"
    }
  }
}
]

```

## 2. List the details of one CER product

The proceeding operation shows the output of the API. It includes the same details as the previous operation for just one product, but it adds all the configured product periods.

For each period, it includes the **periodId**, **startDate**, and **endDate** and a list of XSD files that are required to fully describe the Determination XML for the product period, including rule set schemas, static schemas, and a schema for the product period itself.

The details that are presented can be used to call the next set of operations. When these operations are called, the outputs must be saved to the same location in the operator's file system that adheres to the file names provided here.

```

[
  {
    "id": "26304",
    "typeCode": "PT26304",
    "name": "PN26304",
    "_links": {
      "self": {
        "href": "/xsd/cer/products/26304"
      }
    }
  },
  "periods": [
    {
      "id": "26302",
      "startDate": "2012-01-01",
      "endDate": "null",
      "dependencies": [
        {
          "id": "26302",
          "startDate": "2012-01-01",
          "endDate": "null",
          "file_name": "Product_26302_2012-01-01.xsd",
          "_links": {
            "self": {
              "href": "/xsd/cer/products/26304/periods?date=2012-01-01"
            }
          }
        }
      ]
    },
    {
      "id": "BaseTypes",
      "file_name": "BaseTypes.xsd",
      "_links": {
        "self": {
          "href": "/xsd/cer/statics/BaseTypes"
        }
      }
    },
    {
      "id": "DeterminationToDecisionDetails",
      "file_name": "DeterminationToDecisionDetails.xsd",
      "_links": {
        "self": {
          "href": "/xsd/cer/statics/DeterminationToDecisionDetails"
        }
      }
    },
    {
      "id": "InsuranceAssistanceDisplayRuleSet",

```

```

        "file_name": "InsuranceAssistanceDisplayRuleSet.xsd",
        "_links": {
          "self": {
            "href": "/xsd/cer/rulesets/InsuranceAssistanceDisplayRuleSet"
          }
        }
      }
    ]
  }
]

```

### 3. Get the contents of a CER product period XSD

The operation takes the `periodId` and date from the preceding listing.

### 4. Get the contents of a CER static XSD

The operation takes the schema name from the preceding listing.

### 5. Get the contents of a CER ruleset XSD

The operation takes the ruleset name from the preceding operation.

### 6. List all CER Static XSDs

The operation is listed for completeness, but the operation is not typically needed as the dependencies of a product period always include the static schemas. The proceeding operation lists the output of the API.

```

[
  {
    "id": "BaseTypes",
    "file_name": "BaseTypes.xsd",
    "_links": {
      "self": {
        "href": "/xsd/cer/statics/BaseTypes"
      }
    }
  },
  {
    "id": "DeterminationToDecisionDetails",
    "file_name": "DeterminationToDecisionDetails.xsd",
    "_links": {
      "self": {
        "href": "/xsd/cer/statics/DeterminationToDecisionDetails"
      }
    }
  }
]

```

### 7. List all CER ruleset XSDs

The operation is listed for completeness, but the operation is not typically needed as the dependencies of a product period always include the ruleset schemas. The proceeding operation lists a portion of the output of the API.

```

[
  {
    "id": "InsuranceAssistanceDisplayRuleSet",
    "file_name": "InsuranceAssistanceDisplayRuleSet.xsd",
    "_links": {
      "self": {
        "href": "/xsd/cer/rulesets/InsuranceAssistanceDisplayRuleSet"
      }
    }
  },
  {
    "id": "StreamlineMedicaidDisplayRuleSet",
    "file_name": "StreamlineMedicaidDisplayRuleSet.xsd",

```

```
    "_links": {  
      "self": {  
        "href": "/xsd/ceer/rulesets/StreamlineMedicaidDisplayRuleSet"  
      }  
    }  
  }  
]
```

**Related information**

[Running the Data Extractor's functions in server mode and non-server mode](#)

---

# Chapter 7. Setup steps to run the IBM Social Program Management Data Extractor

The four main setup steps to run the IBM Social Program Management Data Extractor are completing the prerequisites, creating the database tables, editing the Spring Profile application `.properties`, and getting the database drivers.

## Prerequisite setup steps

---

To run the IBM Social Program Management Data Extractor, operators must perform three prerequisites.

### Procedure

1. The operator must install and set up a Java SE Development Kit (JDK) according to the instructions that are provided by the operator's JDK provider.
2. The operator must use the instructions that are provided by the JDK vendor to set the `JAVA_HOME` and `PATH` environment variables.
3. The operator must copy the same `CryptoConfig.jar` that was copied during the setup steps for the customer's SPM deployment to the `${JAVA_HOME}/jre/lib/ext` directory.

## Creating and dropping the database tables

---

Operators must create and drop the Extraction tables and the SPM tables.

### Creating the Extraction tables

The scripts for creating the Extraction tables are installed in the location `{INSTALL_LOCATION}/DataExtractor/sql/<dbvendor>/CreateDataWarehouseTables.sql`.

### Creating the SPM tables

The scripts for creating the SPM Tools tables are packaged and installed in the location `{INSTALL_LOCATION}/DataExtractor/sql/<dbvendor>/CreateSPMToolsTables.sql`.

**Note:** For the default scripts for Db2, the `CreateDataWarehouseTables.sql` is used to create a separate Data Warehouse schema for the Extraction tables and the SPM Tools tables. So, when the default scripts for Db2 are used it is important to run the `CreateDataWarehouseTables.sql` script first.

### Dropping the SPM Tools tables

The scripts for dropping the SPM Tools tables are packaged and installed in the location `{INSTALL_LOCATION}/DataExtractor/sql/<dbvendor>/DropSPMToolsTables.sql`.

### Dropping the Extraction tables

The scripts for dropping the Extraction tables are installed in the location `{INSTALL_LOCATION}/DataExtractor/sql/<dbvendor>/DropDataWarehouseTables.sql`.

**Note:** For the default scripts for Db2, the `DropDataWarehouseTables.sql` drops the Data Warehouse schema that is shared by the Extraction tables and the SPM Tools tables. So, when the default scripts for Db2 are used it is important to run the `DropDataWarehouseTables.sql` script last.

## Required database access

Operators must configure the Social Program Management data source and the Data Warehouse data source for the IBM Social Program Management Data Extractor.

### The data sources

The first data source that must be configured corresponds to the source Social Program Management database. The second data source that must be configured corresponds to the target Data Warehouse. The information in the preceding SPM data source and Data Warehouse data source sections, the information in *Filter Flow job configuration properties*, and the information in *Editing the Spring Profile (application.properties)* list the database objects that are accessed through each data source and the privileges that the tool requires to run successfully. Operators must grant the privileges to the database user accounts that the operator configured in the `application.properties` for the corresponding data source. For more information, see the *Filter Flow job configuration properties* and *Editing the Spring Profile (application.properties)* related links.

### Example

The database administrator of the Social Program Management database grants READ access to the CASEDECISION table to the user account whose name is the assigned value for the `spm.extract.spm.datasources.username` property in the `application.properties`.

### SPM data source

#### Database user (property in `applications.profile`)

- `spm.extract.spm.datasources.username`

#### Database object type

- TABLES

#### Database objects: SPM tables

- CASEDECISION
- CASEHEADER
- CREOLECASEDECISION
- CREOLECASEDETERMINATION
- CREOLECASEDETERMINATIONDATA
- CREOLEPRODUCTDECISIONDISPCAT
- CREOLEPRODUCTPERIOD
- CREOLEPRODUCTPERIODDISPCAT
- CREOLERULECLASSLINK
- CREOLERULESET
- PRODUCT
- PRODUCTDELIVERY

#### Grant privileges

- READ

### Data Warehouse data source

#### Database user (property in `applications.profile`)

- `spm.extract.dw.datasources.username`

#### Database object type

- TABLES

**Database objects: SPM tools tables**

- BATCH\_JOB\_EXECUTION
- BATCH\_JOB\_EXECUTION\_CONTEXT
- BATCH\_JOB\_EXECUTION\_PARAMS
- BATCH\_JOB\_INSTANCE
- BATCH\_STEP\_EXECUTION
- BATCH\_STEP\_EXECUTION\_CONTEXT

**Grant privileges**

- WRITE

**Database objects: Extraction tables**

- DECISIONS
- DETERMINATIONS
- DETERMINATIONXML
- Filter Flow attribute value Tables

**Grant privileges**

- WRITE

**Database object type**

- SEQUENCES

**Database objects: SPM tools sequences**

- BATCH\_JOB\_EXECUTION\_SEQ
- BATCH\_JOB\_SEQ
- BATCH\_STEP\_EXECUTION\_SEQ

**Grant privileges**

- USAGE

**Note:** IBM recommends that you run the tool on the same computer as your databases. However, if you must run the tool remotely to the databases, IBM recommends that you enable the TLSv1.2 connections between the tool and the two databases. For more information, see the *Appendix D: Securing a connection to the Cúram SPM and Data Warehouse databases over TLS 1.2* related link.

**Related concepts**

[Appendix D: Securing a connection to the Social Program Management and Data Warehouse databases over TLS 1.2](#)

The procedure for securing a connection to the Social Program Management and Data Warehouse databases over TLS 1.2 is determined by the database that you use.

**Related information**

[Filter Flow job configuration properties](#)

[Editing the Spring Profile](#)

## Editing the Spring Profile

---

The properties that operators can edit in the Spring Profile application .properties include the Filter Flow job configuration properties and the ShortNames.properties file.

**Note:** Changes to the application.properties, or to the file specified to be the active Spring Profile, are only effective after the application is restarted.

## Filter Flow job configuration properties

Operators can edit three Filter Flow job configuration properties, which are `spm.extract.creoledeterminations.displayrules.extractlistattributes`, `spm.extract.csvs.output.dir`, and `spm.extract.filterflow.compactionlimit`.

### **spm.extract.creoledeterminations.displayrules.extractlistattributes**

- The property is a comma-delimited list of attribute paths to query.
- An attribute path is a reference to a rule attribute that uses the format `RuleSet.RuleClass.RuleAttribute`.
- An example of the property is:  
`StreamlineMedicaidDisplayRuleSet.StreamlineMedicaidIncomeCategory.isInPostPartumPeriodTimeline`
- Where `RuleAttributeA` is of type `RuleClass`, then `RuleSet.RuleClass.RuleAttributeA.RuleAttributeB` is also supported.
- Tip: By using backslashes at the end of each line in the file, the user can put one attribute path per line to make it more readable.
- For example:

```
spm.extract.creoledeterminations.displayrules.extractlistattributes=\
```

```
InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceDeniedPrograms.programNameTimeline,InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceDetailsCategory.assistanceTypeList
```

- For more information about how to search for all valid attribute paths, see the *Searching for the Filter Flow attribute paths* related link.
- Default value when not specified: NONE.

### **spm.extract.csvs.output.dir**

- The property is a string.
- The property is an absolute file path that if explicitly specified must point to an existing directory on the local file system to which the java process has permission to write.
- The property is the location where the CSV output of the Filter Flow CSV job is written.
- The property only applies to the Filter Flow CSV job. When the tool is deployed on a Windows OS, escape the back-slashes in the file path. For example: `C:\\IBM\\DataExtractor\\csvs\\`
- Default value when not specified: `${java.io.tmpdir}spmExtractToolCsvs`. For more information about class files, see the *Class File* related link.

### **spm.extract.filterflow.compactionlimit**

- The property is a positive integer greater than or equal to zero.
- When zero, the property is used to disable the compaction behavior.
- When greater than zero, the property is used to enable the compaction behavior. For more information about compaction behavior, see the *Filter Flow compaction limit* related link.
- The maximum value that can be specified for the property is 500.
- Default value when not specified: 0.

### **Related information**

[Searching for the Filter Flow attribute paths](#)

[Class File](#)

[Filter Flow compaction limit](#)

## Job configuration properties (applies to all job types)

Operators can edit 11 job configuration properties. The configuration properties include `spm.extract.product.typecode` and `spm.extract.product.id`.

### **spm.extract.product.typecode**

Description

- The property is a string.
- The property is a product type code.
- The extract job reads the determinations that belong to the product delivery cases for products that use this type code.
- Set the property `spm.extract.product.typecode` or set the property `spm.extract.product.id`, but do not set both properties. Setting both properties prevents the tool from starting.

Default value when not specified

- NONE

### **spm.extract.product.id**

Description

- The property is a long.
- The property is a product identifier.
- The property corresponds to the primary key of the PRODUCT table in Social Product Management that is PRODUCT.PRODUCTID.
- The extract job reads the determinations that belong to the product delivery cases for the product that uses the `productid`.
- Set the property `spm.extract.product.id` or set the property `spm.extract.product.typecode`, but do not set both properties. Setting both properties prevents the tool from starting.

Default value when not specified

- NONE

### **spm.extract.creoleDeterminations.includemanual**

Description

- `true` indicates that all job types include determinations of type Manual Check Eligibility (CDT1).
- `false` indicates that all job types exclude determinations of type Manual Check Eligibility (CDT1).

Default value when not specified

- `true`

### **spm.extract.maxdateinterval**

Description

- The property is a positive integer *I*, greater than zero.
- The number of days that are permitted between the **startDate** and **endDate** parameters for all extraction job types.

Default value when not specified

- 3

### **spm.extract.maxitemcount**

Description

- The property is a positive integer *J*, greater than zero.
- The maximum number of determinations read by a step execution. The criteria applies to all job types.

- All jobs are configured with a single step. However, XML Flow and Filter Flow database support parallel step executions. For more information about step executions, see the *Batch jobs: job types and their purposes* related link.
- The step execution finishes either after it reads the maximum number of items or the backlog of readable items is exhausted, whichever happens first. For example, where an XML Flow job is started with a **gridSize** of two, a job execution runs with two parallel step executions.
- Each parallel execution can read as many items as set by the **maxItemCount**.

Default value when not specified

- 250

#### **spm.extract.chunkloginterval**

Description

- The property is a positive integer  $K$ , greater than zero.
- A property that is used by all job types.
- When specified to be number  $K$ , XML Flow and Filter job executions log progress every  $K$  number of chunks.

Default value when not specified

- 2

#### **spm.extract.skiplimit**

Description

- The property is a positive integer  $L$ , greater than zero.
- The number of read or process failures that are tolerated per step execution before the step is deemed failed.

Default value when not specified

- 100

#### **spm.extract.pagesize**

Description

- The property is a positive integer  $M$ , greater than zero, less than or equal to 500.
- A property that is used by all batch jobs.
- A property that is used by the paged reader SQL query that is run by all the batch jobs types to read the CREOLECASEDETERMINATION.CREOLECASEDETERMINATIONIDs from those CREOLECASEDETERMINATION rows that match the query criteria,  $M$  IDs at a time. For more information about how determinations are read, see the *How determinations are read by the IBM Social Program Management Data Extractor batch jobs* related link.
- A larger  $M$  means fewer reader queries are run against the database during the execution of a job.

Default value when not specified

- 200

#### **spm.extract.fetchsize**

Description

- The property is a positive integer  $N$ , greater than zero, less than or equal to 500.
- A property that is used to determine the number of rows that are fetched from the database after a reader query is run. It affects the number of network round trips to the database to fetch the results of a query. For example, if reader query page size was 20 and the fetch size was 10 then, after a reader query that yields 20 rows is run, the tool must make two round trips to the database to fetch these rows. For more information about the fetch size, see the *setFetchSize* related link. IBM suggests that operators leave the property the same as the **pageSize**.

Default value when not specified

- 200

### **spm.extract.chunksize**

#### Description

- The property is a positive integer  $O$ , greater than zero, less than or equal to 500.
- For chunk size  $O$ , a batch job buffers every  $O$  items that are processed before the batch job sends them to the associated ItemWriter for the job. The write is then committed.
- For the XML Flow, a processed item or output item corresponds to the number of items that are processed before the resulting items are sent to ItemWriter for that job. A processed item or output item for the XML Flow job corresponds to a row in the DETERMINATIONXML table in the Data Warehouse.
- For the Filter Flow database job, a processed item or output item corresponds to all the data that is extracted from a single determination. As a result, a collection of output rows is written to multiple Filter Flow database tables in the Data Warehouse. A larger chunk size means fewer database calls and fewer database commits to the Data Warehouse.
- For the Filter Flow CSV, a processed item or output item corresponds to the number of items that are processed before the resulting items are sent to ItemWriter for that job.
- For the Filter Flow CSV job, a processed item or output item corresponds to all the data that is extracted from a single determination. As a result, a collection of output rows is written to multiple CSV files on the local file system.
- The chunk size is the commit interval for a job. The bigger the number, the fewer the commit operations there is for a job.
- IBM suggests that operators leave the property the same as the **pageSize**.

#### Default value when not specified

- 200

### **spm.extract.gridsize**

#### Description

- The property is a positive integer,  $P$ .
- The property is used by the Filter Flow database and the XML Flow batch jobs. The property is not used by the Filter Flow CSV batch job.
- When specified as a value greater than 1, the tool creates partitioned step executions. With a grid size that is greater than 1, the following changes then apply:
  - A single execution of a job starts multiple step executions in parallel. For example, with **gridSize**  $P$ ,  $P$  step executions are started with their own dedicated ItemReader instance and run the execution cycle. For more information about the execution cycle, see the *Batch jobs: job types and their purposes* related link.
  - A dedicated partitioner creates a separate step execution context that effectively divides the range of CREOLECASEDETERMINATION.CREOLECASEDETERMINATIONIDs that are to be processed by this job into  $P$  partitions.
  - Each step execution is then used to read and process the determinations that correspond to the IDs in the partition for which it is responsible.
- In the release of the IBM Social Program Management Data Extractor, a thread-based grid execution fabric is used. So, each step execution has a corresponding thread in the same JVM as the IBM Social Program Management Data Extractor tool itself.
- A larger grid size  $P$  means that instead of a job execution that reads-processes-writes items serially, the job execution is performed in parallel over  $\#P$  threads. As a result, the job execution typically takes less time than with a single thread.
- The lower limit of the property is 1. The upper limit of the property is 128.

#### Default value when not specified

- 4

## Related concepts

How determinations are read by the IBM Social Program Management Data Extractor batch jobs  
The reader that is used by all three job types reads the set of CREOLECASEDETERMINATIONIDS for the rows of the CREOLECASEDETERMINATION table in the Social Program Management database that match the preceding criteria.

## Related information

[Batch jobs: job types and their purposes](#)

[setFetchSize](#)

## Editing the Social Program Management database data source properties

Operators must configure properties to specify the data source to connect to the Social Program Management database.

### Social Program Management database properties

The following code explains the Social Program Management database properties.

```
#The Database Vendor type of the SPM database.
#Allowable values are
#Db2|Oracle|h2
spm.extract.spm.datasources.platform=<spm-db-type>
#The JDBC driver class used to connect to the SPM database.
#DB2    -> com.ibm.db2.jcc.DB2Driver
#Oracle -> oracle.jdbc.driver.OracleDriver
#H2     -> org.h2.Driver
spm.extract.spm.datasources.driver-class-name=<jdbc-driver-class>
#The JDBC URL used to connect to the SPM database.
#Db2    -> jdbc:db2://<spm-db-server-name>:<spm-db-port>/<spm-db-name>
#Oracle -> jdbc:oracle:thin:@//<spm-db-server-name>:<spm-db-port>/<spm-db-name>
#H2     -> jdbc:h2:tcp://localhost/file:<file-system--absolute-path>/<spm-db-
name>;schema=<spm-db-schema>
spm.extract.spm.datasources.url=<spm-db-jdbc-url>
#The database schema to which the SPM tables belong.
#Comment out this property if you are configuring this datasources as a H2 datasources
spm.extract.spm.datasources.schema=<spm-db-schema>
#The database user name as whom we connect to the SPM database.
spm.extract.spm.datasources.username=<spm-db-user>
#The Curam encrypted password for the database user.
spm.extract.spm.datasources.password=<spm-db-password>
```

### Example Social Program Management database properties

The following code is an example of the Social Program Management database properties.

```
spm.extract.spm.datasources.driver-class-name=oracle.jdbc.driver.OracleDriver
spm.extract.spm.datasources.url=jdbc:oracle:thin:@//localhost:1521/orcl
spm.extract.spm.datasources.schema=CURAM
spm.extract.spm.datasources.username=curam
spm.extract.spm.datasources.password=GVE/3J2k+3KkoF62aRdUjTyQ/5TVQZ4fI2PuqJ3+4d0=
spm.extract.spm.datasources.platform=Oracle
```

## Editing the Data Warehouse database data source properties

Operators must configure properties to specify the data warehouse for connecting to the Data Warehouse database.

### Data Warehouse database properties

The following code explains the Data Warehouse database properties.

```
#The Database Vendor type of the Data warehouse database.
#Db2|Oracle|h2
```

```

spm.extract.dw.datasource.platform=<dw-db-type>
#The JDBC driver class used to connect to the Data warehouse database.
#DB2 -> com.ibm.db2.jcc.DB2Driver
#Oracle -> oracle.jdbc.driver.OracleDriver
#H2 -> =org.h2.Driver
spm.extract.dw.datasource.driver-class-name=<jdbc-driver-class>
#The JDBC URL used to connect to the Date warehouse database.
spm.extract.dw.datasource.url=<dw-db-jdbc-url>
#Db2 -> jdbc:db2://<dw-db-server-name>:<dw-db-port>/<dw-db-name>
#Oracle -> jdbc:oracle:thin:@//<dw-db-server-name>:<dw-db-port>/<dw-db-name>
#H2 -> jdbc:h2:tcp://localhost/file:<file-system--absolute-path>/<dw-db-name>;schema=<dw-
db-schema>
#The database schema to which the Data warehouse tables belong.
#Comment out this property if you are configuring this datasource as a H2 datasource
spm.extract.dw.datasource.schema=<dw-db-schema>
#The database user name as whom we connect to the SPMTools database.
spm.extract.dw.datasource.username=<dw-db-user>
#The Curam encrypted password for the database user.
spm.extract.dw.datasource.password=<dw-db-password>

```

### Example Data Warehouse properties

The following code is an example of the Data Warehouse database properties.

```

spm.extract.dw.datasource.url=jdbc:oracle:thin:@//localhost:1521/orcl
spm.extract.dw.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spm.extract.dw.datasource.schema=CURAM
spm.extract.dw.datasource.username=curam
spm.extract.dw.datasource.password=GVE/3J2k+3KkoF62aRdUjTyQ/5TVQZ4fI2PuqJ3+4d0=
spm.extract.dw.datasource.platform=Oracle

```

## The ShortNames.properties file

Changes to the `ShortNames.properties` are effective only after an application restart.

The algorithm to generate the names of the Filter Flow database tables might produce table names whose length exceeds the limits on the table name length for a database vendor. For more information about potential table name issues, see the *Issues associated with the length of a generate table name* related link. Operators can use `ShortNames.properties`, the introduced conversion file, to shorten the names produced by the algorithm.

Operators can control whether a conversion file is used by including a `ShortNames.properties` file in a directory on the `loader.path` of the tool. For more information about the `loader.path`, see the *Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode* related link.

### Substitution rules

Substitution is performed according to the following rules:

- The possible substitution is checked at the start position of the word to be converted.
- If the word is found, the word is replaced with the right string of the substitution pair.
- The same procedure is repeated from the beginning of the word and continues until either the shortened name has no more than 18 characters or no possible substitutions are found.
- If the word to be shortened consists of several words that are delimited by the “\_” character, then each component of the word is shortened independently.

The following examples show how the substitution rules apply.

### The ShortNames.properties file example

```

# -----
# File layout:
# - everything to the right of the hash symbol '#' is treated
# as comments and is ignored by the substituter.
# - pairs of strings denote substitutions whereby instances of

```

```

# the left string are replaced with instances of the right
# string.
# -----
# Substitutions
OVERUNDERPAYMENT=OVUNPYMT
INSTRUCTIONLINEITEM=ILI
INSTRUCTLINEITEM=ILI
PAYMENT=PYMNT

```

Based on the file, the names OVERUNDERPAYMENTENTITY becomes OVUNPYMENTENTITY and SAMPLEPAYMENTENTITY becomes SAMPLEPYMNTENTITY.

**Note:** The substitutions are applied in order of their appearance in the ShortNames.properties file.

### Words example

In the proceeding example, the file name PERSONALHOMEFAXTELEPHONENUMBER becomes PRSHOMEFAXPHONE,NUM.

```

# -----
# File layout:
# - everything to the right of the hash symbol '#' is treated
# as comments and is ignored by the substituter.
# - pairs of strings denote substitutions whereby instances of
# the left string are replaced with instances of the right
# string.
# -----
# Substitutions
PERSON=PERS
PERSAL=PRS
NUMBER=NUM
TELEPHONE=PHONE
TELEPH=F

```

The name that consists of several words delimited by an underscore character PERSONALHOMEFAXTELEPHONENUMBER\_PERSONALHOMEFAXTELEPHONENUMBER\_PERSONALHOME FAXTELEPHONENUMBER becomes PRSHOMEFAXPHONENUM\_PRSHOMEFAXPHONENUM\_PRSHOMEFAXPHONENUM.

In the proceeding example, a different ordering is used so that PERSONALHOMEFAXTELEPHONENUMBER becomes PRSHOMEFAXFONENUM.

```

# -----
# File layout:
# - everything to the right of the hash symbol '#' is treated
# as comments and is ignored by the substituter.
# - pairs of strings denote substitutions whereby instances of
# the left string are replaced with instances of the right
# string.
# -----
# Substitutions
PERSON=PERS
PERSAL=PRS
NUMBER=NUM
TELEPH=F
TELEPHONE=PHONE

```

### Related information

[Issues associated with the length of a generated table name](#)

[Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode](#)

## Getting the database drivers

---

The IBM Social Program Management Data Extractor uses the database drivers for the database server that is used by the customer. The database drivers must be copied to the computer where the operator intends to run the IBM Social Program Management Data Extractor.

The `log4j2.properties` and the `ShortNames.properties` can be colocated in the same directory as the database driver JAR files. As a result and providing that the directory is on the `loader.path`, the resources are used by the tool. For more information about the Apache log4j2 logging service, see the [Appendix B: Logging](#) related link. For more information about the `ShortNames.properties` file, see the [The ShortNames.properties file](#) related link.

### Db2 drivers

For more information about how to obtain the appropriate Java Database Connectivity (JDBC) driver jars, see RDBMS official documentation. For more information about the required driver versions and downloads, see the [DB2 JDBC Driver Versions and Downloads](#) related link.

### Oracle drivers

For information about how to obtain the appropriate JDBC driver jars, see RDBMS official documentation. For more information about Oracle 12cR1, see the [Oracle Database 12.1.0.1 JDBC Driver & UCP Downloads](#) related link. For more information about Oracle 12cR2, see the [Oracle 12cR2, see Oracle Database 12.2.0.1 JDBC Driver & UCP Downloads](#) related link.

The three following Oracle jars are required by the tool:

- `ojdbc{X}.jar`
  - For Oracle 12cR1; `ojdbc7.jar`.
  - For Oracle 12cR2; `ojdbc8.jar`.
- `xmlparserv2.jar`
- `xdb.jar`

### Related concepts

[Appendix B: Logging](#)

The IBM Social Program Management Data Extractor uses Apache Log4j 2 as its logging service.

### Related information

[The ShortNames.properties file](#)

[DB2 JDBC Driver Versions and Downloads](#)

[Oracle Database 12.1.0.1 JDBC Driver & UCP Downloads](#)

[Oracle Database 12.2.0.1 JDBC Driver & UCP Downloads](#)

---

## Chapter 8. Running the IBM Social Program Management Data Extractor

To run the IBM Social Program Management Data Extractor, operators must perform mandatory steps. Depending on configuration, operators might need to perform an optional step.

### Procedure

1. Copy the database driver jars for your selected database server to the client computer where the tool is installed. IBM suggests creating a folder here: `{INSTALL_LOCATION}/DataExtractor/lib/`. Then, copy your database driver jars to the folder.
2. Create a copy of the file `sample/application-SAMPLE.properties` and rename the copy to `application-{yourProfileName}.properties`, for example `application-TEST.properties`. IBM suggests that you create a folder here: `{INSTALL_LOCATION}/DataExtractor/profiles/`. Then, place the properties file in the folder.  
Support is only provided for the use of one profile at a time. However, the operator you can use a different profile to start the tool to switch between database systems, job configurations, or both.
3. Change the properties file. For more information about how to change the file, see the *Editing the Spring Profile* related link.
4. Depending on your configuration, you might need to create a copy of the file `ShortNames-SAMPLE.properties` and rename the copy to `ShortNames.properties`. Ensure that the renamed copy is on the `loader.path` when you start the tool in either server or non-server mode. For simplicity, IBM suggests that you copy the file to the `{INSTALL_LOCATION}/DataExtractor/lib/` directory that you created in Step 1. Edit the `ShortNames.properties` file. For more information about how to edit the file, see the *Generating the Filter Flow attribute value database tables creation and drop DDL* related link.
5. Open a command window and change directory to `{INSTALL_LOCATION}/DataExtractor`.

### Results

When the application starts, it prints to the console and to a log file that is named `dataExtractor.log` in the directory where the tool was started.

### Related concepts

[Generating the Filter Flow attribute value database tables creation and drop DDL](#)

Generating the creation and drop DDL for the Filter Flow attribute value tables is a function of the IBM Social Program Management Data Extractor.

### Related information

[Editing the Spring Profile](#)

---

## Starting the IBM Social Program Management Data Extractor in server mode

When the operator starts the tool in server mode, the operator does not specify the function to start. The tool starts an MBean server and registers the MBeans that provide APIs for calling the tool functions by using JConsole. The operator can then call the tool's functions by using an MBean API. The tool runs until the operator explicitly shuts down the Java virtual machine (JVM).

### About starting the IBM Social Program Management Data Extractor in server mode

The `loader.path` is a comma-separated class path, such as `lib,${HOME}/app/lib`. In the Spring Batch *The Executable Jar Format* documentation, the following description of the `loader.path` is provided: "Earlier entries take precedence, like a regular `-classpath` on the `javac` command line."

The command to run the tool is:



```

'spm.extract.spm.datasources' Database Meta Data 'Database Product Name 'Oracle' Database
Product Version 'Oracle Database 12c Enterprise Edition Release 12.2.0.1.0 - 64bit Production'
Database Driver Version '12.2.0.1.0'
2019-07-22 08:03:32.559 [INFO ] [main] DataSourceConfiguration - >>getDataSource: Data Source
'spm.extract.spm.datasources' Database URL
'jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCPS)(HOST=127.0.0.1)(PORT=2484))
(CONNECT_DATA=(SERVICE_NAME=ORCL)))'
2019-07-22 08:03:32.559 [INFO ] [main] DataSourceConfiguration - >>getDataSource: Data Source
'spm.extract.spm.datasources' Database Driver Class Name 'oracle.jdbc.OracleDriver'
2019-07-22 08:03:35.787 [INFO ] [main] ApplicationProperties - <<getViolations(), returning=[]
2019-07-22 08:03:35.917 [INFO ] [main] ExtractToolConfiguration - >>filterFlowProperties
extractlistattributes
InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceDeniedPrograms.programNameTimeline,Insuranc
eAssistanceDisplayRuleSet.InsuranceAssistanceDetailsCategory.assistanceTypeList,InsuranceAssista
nceDisplayRuleSet.InsuranceAssistanceDetailsCategory.caseParticipantRoleID,InsuranceAssistanceDi
splayRuleSet.InsuranceAssistanceDetailsCategory.isPassedNonFinancialRules,InsuranceAssistanceDis
playRuleSet.InsuranceAssistanceIncomeCategory.countedMemberMAGITimeline
2019-07-22 08:03:35.926 [INFO ] [main] RulesMetaModel - Building an in-memory model of the rule
set definitions.
2019-07-22 08:05:59.483 [WARN ] [main] RulesMetaModel - ISScreening includes classpath 'curam/
citizenworkspace/rules/impl/CitizenWorkspaceScreeningInterface.xml'. Includes are not
supported. This can result in unqueriable inherited attributes.
2019-07-22 08:05:59.490 [WARN ] [main] RulesMetaModel - InternalScreeningRuleSet includes
classpath 'curam/citizenworkspace/rules/impl/CitizenWorkspaceScreeningInterface.xml'. Includes
are not supported. This can result in unqueriable inherited attributes.
2019-07-22 08:05:59.492 [WARN ] [main] RulesMetaModel - OnePageScreening includes classpath
'curam/citizenworkspace/rules/impl/CitizenWorkspaceScreeningInterface.xml'. Includes are not
supported. This can result in unqueriable inherited attributes.
2019-07-22 08:05:59.514 [WARN ] [main] RulesMetaModel - InternalScreeningRuleSet_V2 includes
classpath 'curam/citizenworkspace/rules/impl/CitizenWorkspaceScreeningInterface.xml'. Includes
are not supported. This can result in unqueriable inherited attributes.
2019-07-22 08:06:01.167 [INFO ] [main] MetaDestinationSchemaFactory - Queried Attribute Paths
and their XPath:-
2019-07-22 08:06:01.173 [INFO ] [main] MetaDestinationSchemaFactory -
InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceDeniedPrograms.programNameTimeline = /
Determination/DR/M/E/T/I/V/S/DecisionDetails/InsuranceAssistanceDeniedPrograms/
programNameTimeline/Item
2019-07-22 08:06:01.173 [INFO ] [main] MetaDestinationSchemaFactory -
InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceDetailsCategory.assistanceTypeList = /
Determination/DR/M/E/T/I/V/S/DecisionDetails/InsuranceAssistanceDetailsCategory/
assistanceTypeList/Item
2019-07-22 08:06:01.173 [INFO ] [main] MetaDestinationSchemaFactory -
InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceDetailsCategory.caseParticipantRoleID = /
Determination/DR/M/E/T/I/V/S/DecisionDetails/InsuranceAssistanceDetailsCategory/
caseParticipantRoleID
2019-07-22 08:06:01.173 [INFO ] [main] MetaDestinationSchemaFactory -
InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceDetailsCategory.isPassedNonFinancialRules
= /Determination/DR/M/E/T/I/V/S/DecisionDetails/InsuranceAssistanceDetailsCategory/
isPassedNonFinancialRules
2019-07-22 08:06:01.177 [INFO ] [main] MetaDestinationSchemaFactory -
InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceIncomeCategory.countedMemberMAGITimeline
= /Determination/DR/M/E/T/I/V/S/DecisionDetails/InsuranceAssistanceIncomeCategory/
countedMemberMAGITimeline/Item
2019-07-22 08:06:03.228 [INFO ] [main] ApplicationProperties - <<getViolations(), returning=[]
2019-07-22 08:06:05.764 [INFO ] [main] ExtractToolApplication - Started ExtractToolApplication
in 164.766 seconds (JVM running for 178.26)
2019-07-22 08:06:05.771 [INFO ] [main] ExtractToolApplication - ##### WELCOME #####
2019-07-22 08:06:05.771 [INFO ] [main] ExtractToolApplication - Ensure your Database is
running...
2019-07-22 08:06:05.774 [INFO ] [main] ExtractToolApplication - Please see product
documentation for details on how to call the functionality of this tool.

```

## Calling the functions of the tool through the MBean APIs by using JConsole

1. Where the operator sets the `JAVA_HOME` and `PATH` environment variables for the Java SE Development Kit (JDK), the operator can start the JConsole in these ways:

- For Windows, use: `jconsole.exe`.
- For \*nix, use: ``${JAVA_HOME}/bin/jconsole`.

For information about how to securely connect the IBM Social Program Management Data Extractor by using the JConsole, see the *Appendixes* related link.

2. When the operator starts the JConsole, the operator is prompted to create a new connection to either a local or remote process.

3. From a list of the processes, the operator selects the process whose ID corresponds to the process of the started application that is recorded in the `dataExtractor.log`: `[INFO ] 2019-07-22 11:19:40.704 [main] ExtractToolApplication - Starting ExtractToolApplication on LAPTOP-J35DTHTU with PID 24244 (C:\IBM\DataExtractor_Gold\DataExtractor\DataExtractor-<version_no>.jar started by userX in C:\IBM\DataExtractor_Gold\DataExtractor"`
4. The operator clicks **Connect**.
5. The operator clicks **Insecure Connection**. The **Overview** tab is displayed.
6. The operator clicks the **MBeans** tab. For information about how to call a specific function, see the *Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode* related link.

### Related information

[Spring Batch: The Executable Jar Format](#)

[Appendixes](#)

[Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode](#)

## Starting the IBM Social Program Management Data Extractor in non-server mode

When the tool is run in non-server mode, the operator must specify the tool function to run. The tool starts, runs that one function, and then exits. In non-server mode, the IBM Social Program Management Data Extractor MBeans are not registered, so the MBean API is unavailable. To schedule new batch job instances in non-server mode, use **JobLauncherApplication**. For all other non-server mode functions, use **GenericTaskLauncherApplication**.

### JobLauncherApplication

The following code is the general structure of the **JobLauncherApplication** command:

```
java
  -Dloader.path={path-to-directory-with-db-drivers}
  -Dspring.config.location={path-to-profiles-directory}
  -Dspring.profiles.active={spring-profile-name}
  -Dloader.main=com.ibm.spm.extracttool.JobLauncherApplication
  -Dloader.args="{JobName} {JobParameters}"
  -jar DataExtractor-<version_no>.jar
```

Where:

- `{JobName}` is either `XmlFlow`, `FilterFlowCsv`, or `FilterFlowDb`.
- `{JobParameters}` is a space-delimited set of parameters.

For example: `XmlFlow name=jobName101 startDate=2019-01-01 endDate=2019-06-01`. For more examples, see the *Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode* related link.

### GenericTaskLauncherApplication

The following code is the general structure of the **GenericTaskLauncherApplication** command:

```
java
  -Dloader.path={path-to-directory-with-db-drivers}
  -Dspring.config.location={path-to-profiles-directory}
  -Dspring.profiles.active={spring-profile-name}
  -Dloader.main=com.ibm.spm.extracttool.GenericTaskLauncherApplication
  -Dloader.args="{Function} {FunctionArgs}"
  -jar DataExtractor-<version_no>.jar
```



```

2019-07-22 08:27:27.068 [INFO ] [main] ExtractToolConfiguration - >>filterFlowProperties
extractlistattributes
InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceDeniedPrograms.programNameTimeline,Insuranc
eAssistanceDisplayRuleSet.InsuranceAssistanceDetailsCategory.assistanceTypeList,InsuranceAssista
nceDisplayRuleSet.InsuranceAssistanceDetailsCategory.caseParticipantRoleID,InsuranceAssistanceDi
splayRuleSet.InsuranceAssistanceDetailsCategory.isPassedNonFinancialRules,InsuranceAssistanceDis
playRuleSet.InsuranceAssistanceIncomeCategory.countedMemberMAGITimeline
2019-07-22 08:27:27.076 [INFO ] [main] RulesMetaModel - Building an in-memory model of the rule
set definitions.
2019-07-22 08:29:43.670 [WARN ] [main] RulesMetaModel - ISScreening includes classpath 'curam/
citizenworkspace/rules/impl/CitizenWorkspaceScreeningInterface.xml'. Includes are not
supported. This can result in unqueriable inherited attributes.
2019-07-22 08:29:43.675 [WARN ] [main] RulesMetaModel - InternalScreeningRuleSet includes
classpath 'curam/citizenworkspace/rules/impl/CitizenWorkspaceScreeningInterface.xml'. Includes
are not supported. This can result in unqueriable inherited attributes.
2019-07-22 08:29:43.677 [WARN ] [main] RulesMetaModel - OnePageScreening includes classpath
'curam/citizenworkspace/rules/impl/CitizenWorkspaceScreeningInterface.xml'. Includes are not
supported. This can result in unqueriable inherited attributes.
2019-07-22 08:29:43.687 [WARN ] [main] RulesMetaModel - InternalScreeningRuleSet_V2 includes
classpath 'curam/citizenworkspace/rules/impl/CitizenWorkspaceScreeningInterface.xml'. Includes
are not supported. This can result in unqueriable inherited attributes.
2019-07-22 08:29:44.822 [INFO ] [main] MetaDestinationSchemaFactory - Queried Attribute Paths
and their XPath:-
2019-07-22 08:29:44.824 [INFO ] [main] MetaDestinationSchemaFactory -
InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceDeniedPrograms.programNameTimeline = /
Determination/DR/M/E/T/I/V/S/DecisionDetails/InsuranceAssistanceDeniedPrograms/
programNameTimeline/Item
2019-07-22 08:29:44.825 [INFO ] [main] MetaDestinationSchemaFactory -
InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceDetailsCategory.assistanceTypeList = /
Determination/DR/M/E/T/I/V/S/DecisionDetails/InsuranceAssistanceDetailsCategory/
assistanceTypeList/Item
2019-07-22 08:29:44.825 [INFO ] [main] MetaDestinationSchemaFactory -
InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceDetailsCategory.caseParticipantRoleID = /
Determination/DR/M/E/T/I/V/S/DecisionDetails/InsuranceAssistanceDetailsCategory/
caseParticipantRoleID
2019-07-22 08:29:44.826 [INFO ] [main] MetaDestinationSchemaFactory -
InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceDetailsCategory.isPassedNonFinancialRules
= /Determination/DR/M/E/T/I/V/S/DecisionDetails/InsuranceAssistanceDetailsCategory/
isPassedNonFinancialRules
2019-07-22 08:29:44.827 [INFO ] [main] MetaDestinationSchemaFactory -
InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceIncomeCategory.countedMemberMAGITimeline
= /Determination/DR/M/E/T/I/V/S/DecisionDetails/InsuranceAssistanceIncomeCategory/
countedMemberMAGITimeline/Item
2019-07-22 08:29:46.779 [INFO ] [main] ApplicationProperties - <<getViolations(), returning=[]
2019-07-22 08:29:48.812 [INFO ] [main] JobLauncherApplication - Started JobLauncherApplication
in 153.808 seconds (JVM running for 166.668)
2019-07-22 08:30:01.519 [INFO ] [SimpleAsyncTaskExecutor-1] DeterminationStepLifeCycleListener
- >>logStepFinished:
-----
Job Name: 'XmlFlow'
Instance: 20
Execution: 20
Finished Step: 'xmlFlowStep0'
Step Summary: 'StepExecution: id=20, version=2, name=xmlFlowStep0, status=COMPLETED,
exitStatus=COMPLETED, readCount=2, filterCount=0, writeCount=2 readSkipCount=0,
writeSkipCount=0, processSkipCount=0, commitCount=1, rollbackCount=0'
-----
2019-07-22 08:30:01.522 [INFO ] [SimpleAsyncTaskExecutor-1] DeterminationStepLifeCycleListener
- >>logReaderExhaustedMessages:We have finished querying the backlog.The number of items
successfully read was 2.We were within the limit that was explicitly set for the reader 250.

C:\IBM\DataExtractor_Gold\DataExtractor>

```

## The console log output when the tool is started in non-server mode: searching for the valid Filter Flow attribute paths

A sample shows the console log output for searching for the valid Filter Flow attribute paths when the console log output is started in non-server mode.

When the operator enters the following command:

```

java -Dloader.path=./ora_tls_drivers/ -Dspring.config.location=./profiles/ -
Dspring.profiles.active=Oracle-12c -
Dloader.main=com.ibm.spm.extracttool.GenericTaskLauncherApplication -
Dloader.args="AttributePathSearch.searchAttributePaths attributePathPattern=.*(?
i)incometimeline." -jar DataExtractor-<version_no>.jar

```

the following output is displayed in the console:



```

2019-07-22 10:07:47.324 [INFO ] [main] MetaDestinationSchemaFactory -
InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceDetailsCategory.assistanceTypeList = /
Determination/DR/M/E/T/I/V/S/DecisionDetails/InsuranceAssistanceDetailsCategory/
assistanceTypeList/Item
2019-07-22 10:07:47.325 [INFO ] [main] MetaDestinationSchemaFactory -
InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceDetailsCategory.caseParticipantRoleID = /
Determination/DR/M/E/T/I/V/S/DecisionDetails/InsuranceAssistanceDetailsCategory/
caseParticipantRoleID
2019-07-22 10:07:47.326 [INFO ] [main] MetaDestinationSchemaFactory -
InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceDetailsCategory.isPassedNonFinancialRules
= /Determination/DR/M/E/T/I/V/S/DecisionDetails/InsuranceAssistanceDetailsCategory/
isPassedNonFinancialRules
2019-07-22 10:07:47.326 [INFO ] [main] MetaDestinationSchemaFactory -
InsuranceAssistanceDisplayRuleSet.InsuranceAssistanceIncomeCategory.countedMemberMAGITimeline
= /Determination/DR/M/E/T/I/V/S/DecisionDetails/InsuranceAssistanceIncomeCategory/
countedMemberMAGITimeline/Item
2019-07-22 10:07:49.246 [INFO ] [main] ApplicationProperties - <<getViolations(), returning=[]
2019-07-22 10:07:51.395 [INFO ] [main] GenericTaskLauncherApplication - Started
GenericTaskLauncherApplication in 151.239 seconds (JVM running for 164.815)
2019-07-22 10:07:53.926 [INFO ] [main] GenericTaskLauncherConfiguration -
Filtered Attribute Paths matching pattern '.*(?)incometimeline.':-
-----
FoodAssistanceDisplayRuleSet.FAMemberIncomeCategory.totalEarnedIncomeTimelineList,
FoodAssistanceDisplayRuleSet.FAMemberIncomeCategory.totalGrossIncomeTimelineList,
FoodAssistanceDisplayRuleSet.FAMemberIncomeCategory.totalGrossUnearnedIncomeTimelineList,
FoodAssistanceDisplayRuleSet.FAMemberIncomeCategory.totalNonCountableEarnedIncomeTimelineList,
FoodAssistanceDisplayRuleSet.FAMemberIncomeCategory.totalNonCountableIncomeTimelineList,
FoodAssistanceDisplayRuleSet.FAMemberIncomeCategory.totalNonCountableUnearnedIncomeTimelineList,
FoodAssistanceDisplayRuleSet.FAMemberIncomeCategory.totalSelfEmploymentIncomeTimelineList,
LIFCDisplayRuleSet.LIFCSummaryCategory.extendedLIFCSummaryCategory.riseInChildOrSpouseNetEarnedI
ncomeTimelineMessage,
LIFCDisplayRuleSet.LIFCSummaryCategory.transitionalLIFCSummaryCategory.riseInCaretakersNetEarned
IncomeTimelineMessage
-----
Found 9 Attribute Paths
-----
2019-07-22 10:07:53.931 [INFO ] [main] GenericTaskLauncherApplication - ##### END
GenericTaskLauncherApplication #####

```

## Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode

The functions of the IBM Social Program Management Data Extractor include scheduling jobs, searching for the Filter Flow attribute paths, and batch jobs.

### Scheduling jobs

Operators can schedule the XML Flow job, the Filter Flow CSV job, and the Filter Flow database job in server mode and non-server mode.

#### Scheduling the XML Flow job

Operators can schedule the XML Flow job in server mode and non-server mode.

#### Scheduling the XML Flow job in server mode

1. The operator follows steps 1-6 for [starting JConsole and connecting to the MBean server](#). For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the [Starting the IBM Social Program Management Data Extractor in server mode](#) related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.controller**.
3. The operator expands **JobLaunchingController**. Six buttons are displayed. The following two buttons are relevant for the XML Flow job:
  - **launchXmlFlowJob** (name:String, startDate:String, endDate:String, gridSize:int):long

- **launchXmlFlowJob** (name:String, startDate:String, endDate:String, pageSize:int, maxNumItems:int, gridSize:int):long

The **launchXmlFlowJob** operations support the parameters that are listed in the *Parameters for the IBM Social Program Management Data Extractor's functions* related link.

The **launchXmlFlowJob** operation is overloaded with two versions that are shown in the preceding list.

The **launchXmlFlowJob** (name:String, startDate:String, endDate:String, gridSize:int):long omits the **pageSize** and the **maxNumItems** parameters. Instead, the effective values for these settings are read from the `application.properties`. To run the job with effective page size, fetch size, and chunk size that differ from each other, operators use this version.

The **launchXmlFlowJob** (name:String, startDate:String, endDate:String, pageSize:int, maxNumItems:int, gridSize:int):long includes the **pageSize** and the **maxNumItems** parameters. The value that is specified for the **pageSize** parameter becomes the effective value for page size, fetch size, and chunk size.

4. The operator specifies the value for a job parameter by completing the applicable text box for one of the two versions of the method.
5. The operator clicks the corresponding **launchXmlFlowJob** button. The Job Execution ID of the scheduled batch job is displayed.

### Scheduling the XML Flow job in non-server mode

Enter the following command to schedule the XML Flow job in non-server mode:

```
java -Dloader.main=com.ibm.spm.extracttool.JobLauncherApplication -Dloader.args="XmlFlow
{JobParameters}" -Dspring.config.location={path-to-profiles-directory} -
Dspring.profiles.active={spring-profile-name} -Dloader.path={path-to-database-drivers-
directory} -jar DataExtractor-<version_no>.jar
```

### Related information

[Starting the IBM Social Program Management Data Extractor in server mode](#)

[Parameters for the IBM Social Program Management Data Extractor's functions](#)

### Scheduling the Filter Flow CSV job

Operators can schedule the Filter Flow CSV job in server mode and non-server mode.

### Scheduling the Filter Flow CSV job in server mode

1. The operator follows steps 1-6 for starting JConsole and connecting to the MBean server. For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the *Starting the IBM Social Program Management Data Extractor in server mode* related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.controller**.
3. The operator expands **JobLaunchingController**. Six buttons are displayed. The following two buttons are relevant for the Filter Flow CSV job:
  - **launchFilterFlowCsvJob** (name:String, startDate:String, endDate:String):long
  - **launchFilterFlowCsvJob** (name:String, startDate:String, endDate:String, pageSize:int, maxNumItems:int):long

The **launchFilterFlowCsv** operations support the parameters that are listed in the *Parameters for the IBM Social Program Management Data Extractor's functions* related link.

The **launchFilterFlowCsvJob** operation is overloaded with two versions that are shown in the preceding list.

The **launchFilterFlowCsvJob** (name:String, startDate:String, endDate:String):long omits the **pageSize** and the **maxNumItems** parameters. Instead, the effective values for these settings are read

from the `application.properties`. To run the job with effective page size, fetch size, and chunk size that differ from each other, operators use this version.

The **launchFilterFlowCsvJob** (name:String, startDate:String, endDate:String, pageSize:int, maxNumItems:int):long includes the **pageSize** and the **maxNumItems** parameters. The value that is specified for the **pageSize** parameter becomes the effective value for page size, fetch size, and chunk size.

4. The operator specifies the value for a job parameter by completing the applicable text box for one of the two versions of the method.
5. The operator clicks the corresponding **launchFilterFlowCsvJob** button. The Job Execution ID of the scheduled batch job is displayed.

### Scheduling the Filter Flow CSV job in non-server mode

Enter the following command to schedule the Filter Flow CSV job in non-server mode:

```
java -Dloader.main=com.ibm.spm.extracttool.JobLauncherApplication -Dloader.args="FilterFlowCsv {JobParameters}" -Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-name} -Dloader.path={path-to-database-drivers-directory} -jar DataExtractor-<version_no>.jar
```

### Related information

[Starting the IBM Social Program Management Data Extractor in server mode](#)

[Parameters for the IBM Social Program Management Data Extractor's functions](#)

### Scheduling the Filter Flow database job

Operators can schedule the Filter Flow database job in server mode and non-server mode.

### Schedule the Filter Flow database job in server mode

1. The operator follows steps 1-6 for starting JConsole and connecting to the MBean server. For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the [Starting the IBM Social Program Management Data Extractor in server mode](#) related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.controller**.
3. The operator expands **JobLaunchingController**. Six buttons are displayed. The following two buttons are relevant for the Filter Flow database job:
  - **launchFilterFlowDbJob** (name:String, startDate:String, endDate:String, gridSize:int):long
  - **launchFilterFlowDbJob** (name:String, startDate:String, endDate:String, pageSize:int, maxNumItems:int, gridSize:int):long

The **launchFilterFlowDbJob** operation is overloaded with two versions that are shown in the preceding list.

The **launchFilterFlowDbJob** (name:String, startDate:String, endDate:String, gridSize:int):long omits the **pageSize** and the **maxNumItems** parameters. Instead, the effective values for these settings are read from the `application.properties`. To run the job with effective page size, fetch size and chunk size that differ from each other, operators use this version.

The **launchFilterFlowDbJob** (name:String, startDate:String, endDate:String, pageSize:int, maxNumItems:int, gridSize:int):long includes the **pageSize** and the **maxNumItems** parameters. The value that is specified for the **pageSize** parameter becomes the effective value for page size, fetch size, and chunk size.

4. The operator specifies the value for a job parameter by completing the applicable text box for one of the two versions of the method.
5. The operator clicks the corresponding **launchFilterFlowDbJob** button. The Job Execution ID of the scheduled batch job is displayed.

## Schedule the Filter Flow database job in non-server mode

Enter the following command to schedule the Filter Flow database job in non-server mode:

```
java -Dloader.main=com.ibm.spm.extracttool.JobLauncherApplication -Dloader.args="FilterFlowDb
{JobParameters}" -Dspring.config.location={path-to-profiles-directory} -
Dspring.profiles.active={spring-profile-name} -Dloader.path={path-to-database-drivers-
directory} -jar DataExtractor-<version_no>.jar
```

### Related information

[Starting the IBM Social Program Management Data Extractor in server mode](#)

## Generating the Filter Flow database tables DDL

Operators can generate the Creation DDL for the attribute value tables in server and non-server mode. Operators can also generate the Drop DDL for the attribute value tables in server mode and non-server mode.

### Generating the Creation DDL for the attribute value tables in server mode

1. The operator follows steps 1-6 for [starting JConsole and connecting to the MBean server](#). For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the [Starting the IBM Social Program Management Data Extractor in server mode](#) related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.controller**.
3. The operator expands **DDLGeneratorController**. The following two buttons for this MBean are displayed:
  - **generateCreationDDL ()**
  - **generateDropDDL ()**
4. The operator clicks **generateCreationDDL ()** to generate the creation DDL.

### Generating the Creation DDL for the attribute value tables in non-server mode

Enter the following command to generate the Creation DDL for the attribute value tables in non-server mode:

```
java -Dloader.args="DDLGenerator.generateCreateDDL" -Dloader.path={path-to-directory-with-db-
drivers} -Dloader.main=com.ibm.spm.extracttool.GenericTaskLauncherApplication -
Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-
name} -jar DataExtractor-<version_no>.jar
```

### Generating the Drop DDL for the attribute value tables in server mode

1. The operator follows steps 1-6 for [starting JConsole and connecting to the MBean server](#). For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the [Starting the IBM Social Program Management Data Extractor in server mode](#) related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.controller**.
3. The operator expands **DDLGeneratorController**. The following two buttons for this MBean are displayed:
  - **generateCreationDDL ()**
  - **generateDropDDL ()**
4. The operator clicks **generateDropDDL ()** to generate the drop DDL.

### Generating the Drop DDL for the attribute value tables in non-server mode

Enter the following command to generate the Drop DDL for the attribute value tables in non-server mode:

```
java -Dloader.args="DDLGenerator.generateDropDDL" -Dloader.path={path-to-directory-with-db-
drivers} -Dloader.main=com.ibm.spm.extracttool.GenericTaskLauncherApplication -
```

```
Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-name} -jar DataExtractor-<version_no>.jar
```

## Related information

[Starting the IBM Social Program Management Data Extractor in server mode](#)

## Searching for the Filter Flow attribute paths

Operators can search for the valid Filter Flow attribute paths in server mode and non-server mode.

### Searching for the valid Filter Flow attribute paths in server mode

1. The operator follows steps 1-6 for starting JConsole and connecting to the MBean server. For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the [Starting the IBM Social Program Management Data Extractor in server mode](#) related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.controller**.
3. The operator expands **AttributePathSearchController**. A **searchAttributePaths** (String) button for this MBean is displayed.
4. If the operator does not want to specify an **attributePathPattern** parameter, the operator removes String from the **attributePathPattern** field that corresponds to the **searchAttributePaths** button.
5. The operator clicks **searchAttributePaths** to search for valid attribute paths.

### Searching for the valid Filter Flow attribute paths in non-server mode

Enter the following command to search for the valid Filter Flow attribute paths in non-server mode:

```
java -Dloader.args="AttributePathSearch.searchAttributePaths  
attributePathPattern={regexPattern}" -Dloader.path={path-to-directory-with-db-drivers} -  
Dloader.main=com.ibm.spm.extracttool.GenericTaskLauncherApplication -  
Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-  
name} -jar DataExtractor-<version_no>.jar
```

## Related information

[Starting the IBM Social Program Management Data Extractor in server mode](#)

## Generating XSDs for XML Flow ingested determination XML

Operators can list all valid CER static XSDs, list all valid CER products, read CER product details, print a CER product period XSD, list all valid CER ruleset XSDs, print a CER ruleset XSD, and print a CER static XSD. Operators can perform all these functions in server mode and non-server mode.

### To list all valid CER static XSDs in server mode

1. The operator follows steps 1-6 for [starting JConsole and connecting to the MBean server](#). For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the [Starting the IBM Social Program Management Data Extractor in server mode](#) related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.controller**.
3. The operator expands **XSDGenerationController** and clicks **Operations**. From the operations node, all operation buttons are displayed in the **Operation invocation** pane.
4. The operator clicks **listCERStatics ()**.

### To list all valid CER static XSDs in non-server mode

Enter the following command to list all valid CER static XSDs in non-server mode:

```
java -Dloader.args="XSDGenerator.listCERStatics" -Dloader.path={path-to-directory-with-db-  
drivers} -Dloader.main=com.ibm.spm.extracttool.GenericTaskLauncherApplication -  
Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-  
name} -jar DataExtractor-<version_no>.jar
```

### To list all valid CER products in server mode

1. The operator follows steps 1-6 for [starting JConsole and connecting to the MBean server](#). For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the [Starting the IBM Social Program Management Data Extractor in server mode](#) related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.controller**.
3. The operator expands **XSDGenerationController** and clicks **Operations**. From the operations node, all operation buttons are displayed in the **Operation invocation** pane.
4. The operator clicks **listCERProducts ()**.

### To list all valid CER products in non-server mode

Enter the following command to list all valid CER products in non-server mode:

```
java -Dloader.args="XSDGenerator.listCERProducts" -Dloader.path={path-to-directory-with-db-drivers} -Dloader.main=com.ibm.spm.extracttool.GenericTaskLauncherApplication -Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-name} -jar DataExtractor-<version_no>.jar
```

### To read CER product details, including all periods and each period's dependent XSDs, in server mode

1. The operator follows steps 1-6 for [starting JConsole and connecting to the MBean server](#). For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the [Starting the IBM Social Program Management Data Extractor in server mode](#) related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.controller**.
3. The operator expands **XSDGenerationController** and clicks **Operations**. From the operations node, all operation buttons are displayed in the **Operation invocation** pane.
4. The operator specifies the value for a job parameter by completing the applicable text box.
5. The operator clicks **listCERPeriodsByProduct ()**.

### To read CER product details, including all periods and each period's dependent XSDs, in non-server mode

Enter the following command to read CER product details, including all periods and each period's dependent XSDs, in non-server mode:

```
java -Dloader.args="XSDGenerator.listCERPeriodsByProduct {productId}" -Dloader.path={path-to-directory-with-db-drivers} -Dloader.main=com.ibm.spm.extracttool.GenericTaskLauncherApplication -Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-name} -jar DataExtractor-<version_no>.jar
```

### To print a CER product period XSD in server mode

1. The operator follows steps 1-6 for [starting JConsole and connecting to the MBean server](#). For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the [Starting the IBM Social Program Management Data Extractor in server mode](#) related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.controller**.
3. The operator expands **XSDGenerationController** and clicks **Operations**. From the operations node, all operation buttons are displayed in the **Operation invocation** pane.
4. The operator specifies the value for a job parameter by completing the applicable text box.
5. The operator clicks **getCERPeriod (productid, date)**.

### To print a CER product period XSD in non-server mode

Enter the following command to print a CER product period XSD in non-server mode:

```
java -Dloader.args="XSDGenerator.getCERProductPeriodByProductIDAndDate {productId} {effective_date}" -Dloader.path={path-to-directory-with-db-drivers} -
```

```
Dloader.main=com.ibm.spm.extracttool.GenericTaskLauncherApplication -
Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-
name} -jar DataExtractor-<version_no>.jar
```

### To list all valid CER ruleset XSDs in server mode

1. The operator follows steps 1-6 for [starting JConsole and connecting to the MBean server](#). For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the *Starting the IBM Social Program Management Data Extractor in server mode* related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.controller**.
3. The operator expands **XSDGenerationController** and clicks **Operations**. From the operations node, all operation buttons are displayed in the **Operation invocation** pane.
4. The operator clicks **listCERRuleSets ()**.

### To list all valid CER ruleset XSDs in non-server mode

Enter the following command to list all valid CER ruleset XSDs in non-server mode:

```
java -Dloader.args="XSDGenerator.listCERRuleSets" -Dloader.path={path-to-directory-with-db-
drivers} -Dloader.main=com.ibm.spm.extracttool.GenericTaskLauncherApplication -
Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-
name} -jar DataExtractor-<version_no>.jar
```

### To print a CER ruleset XSD in server mode

1. The operator follows steps 1-6 for [starting JConsole and connecting to the MBean server](#). For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the *Starting the IBM Social Program Management Data Extractor in server mode* related link.
2. In the folder view of the MBean pane, the operator expands the **com.ibm.spm.extracttool.controller** section.
3. The operator expands **XSDGenerationController** and clicks **Operations**. From the operations node, all operation buttons are displayed in the **Operation invocation** pane.
4. The operator specifies the value for a job parameter by completing the applicable text box.
5. The operator clicks **getCERRuleSetByName (ruleSetName)**.

### To print a CER ruleset XSD in non-server mode

Enter the following command to print a CER ruleset XSD in non-server mode:

```
java -Dloader.args="XSDGenerator.getCERRuleSetByName {ruleSetName}" -Dloader.path={path-to-
directory-with-db-drivers} -Dloader.main=com.ibm.spm.extracttool.GenericTaskLauncherApplication
-Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-
name} -jar DataExtractor-<version_no>.jar
```

### To print a CER static XSD in server mode

1. The operator follows steps 1-6 for [starting JConsole and connecting to the MBean server](#). For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the *Starting the IBM Social Program Management Data Extractor in server mode* related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.controller**.
3. The operator expands **XSDGenerationController** and clicks **Operations**. From the operations node, all operation buttons are displayed in the **Operation invocation** pane.
4. The operator specifies the value for a job parameter by completing the applicable text box.
5. The operator clicks the button **getCERStaticBySchemaName (schemaName)**.

## To print a CER static XSD in non-server mode

Enter the following command to print a CER static XSD in non-server mode:

```
java -Dloader.args="XSDGenerator.getCERStaticBySchemaName {schemaName}" -Dloader.path={path-to-directory-with-db-drivers} -Dloader.main=com.ibm.spm.extracttool.GenericTaskLauncherApplication -Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-name} -jar DataExtractor-<version_no>.jar
```

### Related information

[Starting the IBM Social Program Management Data Extractor in server mode](#)

## Batch jobs

Operators can perform 10 batch jobs in server and non-server mode. The batch jobs include abandoning a job execution, restarting a job, and listing job instances.

### Abandoning a job execution

An operator can abandon a job execution in server mode or in non-server mode.

#### Abandon a job execution in server mode

1. The operator follows steps 1-6 for [starting JConsole and connecting to the MBean server](#). For more information about calling the functions of the tool through the MBean APIs by using J Console, see the [Starting the IBM Social Program Management Data Extractor in server mode](#) related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.batch.launch**.
3. The operator expands **com.ibm.spm.extracttool.batch.launch > JobOperatorProxy > Operations**. The list of buttons that is displayed includes **(i) abandonExecution(executionId:long):Long**.
4. The operator specifies the required values in the respective text boxes.
5. The operator clicks the **abandonExecution** button.

#### Abandon a job execution in non-server mode

Enter the following command to abandon a job execution in non-server mode:

```
java -Dloader.args="JobOperatorProxy.abandon {executionId}" -Dloader.path={path-to-directory-with-db-drivers} -Dloader.main=com.ibm.spm.extracttool.GenericTaskLauncherApplication -Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-name} -jar DataExtractor-<version_no>.jar
```

### Related information

[Starting the IBM Social Program Management Data Extractor in server mode](#)

### Stopping a job execution

An operator can stop a job execution in server mode or in non-server mode.

#### Stop a job in server mode

1. The operator follows steps 1-6 for [starting JConsole and connecting to the MBean server](#). For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the [Starting the IBM Social Program Management Data Extractor in server mode](#) related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.batch.launch**.
3. The operator expands **com.ibm.spm.extracttool.batch.launch > JobOperatorProxy > Operations**. The list of buttons that is displayed includes **(i) stop(executionId:long):boolean**.
4. The operator specifies the required values in the respective text boxes.
5. The operator clicks the **stop** button.

## Stop a job in non-server mode

Enter the following command to stop a job execution in non-server mode:

```
java -Dloader.args="JobOperatorProxy.stop {executionId}" -Dloader.path={path-to-directory-with-db-drivers} -Dloader.main=com.ibm.spm.extracttool.GenericTaskLauncherApplication -Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-name} -jar DataExtractor-<version_no>.jar
```

## Related information

[Starting the IBM Social Program Management Data Extractor in server mode](#)

## Getting job parameters

An operator can get job parameters in server mode or in non-server mode.

### Get job parameters in server mode

1. The operator follows steps 1-6 for [starting JConsole](#) and [connecting to the MBean server](#). For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the [Starting the IBM Social Program Management Data Extractor in server mode](#) related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.batch.launch**.
3. The operator expands **com.ibm.spm.extracttool.batch.launch > JobOperatorProxy > Operations**. The list of buttons that is displayed includes **(i) getParameters(executionId:long):String**.
4. The operator specifies the required values in the respective text boxes.
5. The operator clicks the **getParameters** button.

### Get job parameters in non-server mode

Enter the following command to get job parameters in non-server mode:

```
java -Dloader.args="JobOperatorProxy.getJobParameters {executionId}" -Dloader.path={path-to-directory-with-db-drivers} -Dloader.main=com.ibm.spm.extracttool.GenericTaskLauncherApplication -Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-name} -jar DataExtractor-<version_no>.jar
```

## Related information

[Starting the IBM Social Program Management Data Extractor in server mode](#)

## Restarting a job

An operator can restart a job in server mode or in non-server mode.

### Restart a job in server mode

1. The operator follows steps 1-6 for [starting JConsole](#) and [connecting to the MBean server](#). For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the [Starting the IBM Social Program Management Data Extractor in server mode](#) related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.batch.launch**.
3. The operator expands **com.ibm.spm.extracttool.batch.launch > JobOperatorProxy > Operations**. The list of buttons that is displayed includes **(i) restart(executionId:long):Long**.
4. The operator specifies the required values in the respective text boxes.
5. The operator clicks the **restart** button.

### Restart a job in non-server mode

Enter the following command to restart a job in non-server mode:

```
java -Dloader.args="JobOperatorProxy.restart {executionId}" -Dloader.path={path-to-directory-with-db-drivers} -Dloader.main=com.ibm.spm.extracttool.GenericTaskLauncherApplication -Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-name} -jar DataExtractor-<version_no>.jar
```

## Related information

[Starting the IBM Social Program Management Data Extractor in server mode](#)

## Summarizing step executions

An operator can summarize step executions in server mode or in non-server mode.

### Summarize step executions in server mode

1. The operator follows steps 1-6 for [starting JConsole and connecting to the MBean server](#). For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the [Starting the IBM Social Program Management Data Extractor in server mode](#) related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.batch.launch**.
3. The operator expands **com.ibm.spm.extracttool.batch.launch > JobOperatorProxy > Operations**. The list of buttons that is displayed includes **(i) getStepExecutionSummaries(executionId:long):Map**.
4. The operator specifies the required values in the respective text boxes.
5. The operator clicks the **getStepExecutionSummaries** button.

### Summarize step executions in non-server mode

Enter the following command to summarize step executions in non-server mode:

```
java -Dloader.args="JobOperatorProxy.getStepExecutionSummaries {executionId}" -Dloader.path={path-to-directory-with-db-drivers} -Dloader.main=com.ibm.spm.extracttool.GenericTaskLauncherApplication -Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-name} -jar DataExtractor-<version_no>.jar
```

## Related information

[Starting the IBM Social Program Management Data Extractor in server mode](#)

## Summarizing job executions

An operator can summarize job executions in server mode or in non-server mode.

### Summarize job executions in server mode

1. The operator follows steps 1-6 for [starting JConsole and connecting to the MBean server](#). For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the [Starting the IBM Social Program Management Data Extractor in server mode](#) related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.batch.launch**.
3. The operator expands **com.ibm.spm.extracttool.batch.launch > JobOperatorProxy > Operations**. The list of buttons that is displayed includes **(i) getSummary(executionId:long):String**.
4. The operator specifies the required values in the respective text boxes.
5. The operator clicks the **getSummary** button.

### Summarize job executions in non-server mode

Enter the following command to summarize job executions in non-server mode:

```
java -Dloader.args="JobOperatorProxy.getSummary {executionId}" -Dloader.path={path-to-directory-with-db-drivers} -Dloader.main=com.ibm.spm.extracttool.GenericTaskLauncherApplication -Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-name} -jar DataExtractor-<version_no>.jar
```

## Related information

[Starting the IBM Social Program Management Data Extractor in server mode](#)

## Listing job instances

An operator can list job instances in server mode or in non-server mode.

### List job instances in server mode

1. The operator follows steps 1-6 for [starting JConsole and connecting to the MBean server](#). For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the [Starting the IBM Social Program Management Data Extractor in server mode](#) related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.batch.launch**.
3. The operator expands **com.ibm.spm.extracttool.batch.launch > JobOperatorProxy > Operations**. The list of buttons that is displayed includes **(i)getJobInstances(jobName:String,start:int,count:int):List**.
4. The operator specifies the required values in the respective text boxes.
5. The operator clicks the **getJobInstances** button.

### List job instances in non-server mode

Enter the following command to list job instances in non-server mode:

```
java -Dloader.args="JobOperatorProxy.getJobInstances {jobTypeName} {start} {count}" -Dloader.path={path-to-directory-with-db-drivers} -Dloader.main=com.ibm.spm.extracttool.GenericTaskLauncherApplication -Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-name} -jar DataExtractor-<version_no>.jar
```

### Related information

[Starting the IBM Social Program Management Data Extractor in server mode](#)

## Listing the executions of an instance

An operator can list the executions of an instance in server mode or in non-server mode.

### Listing the executions of an instance in server mode

1. The operator follows steps 1-6 for [starting JConsole and connecting to the MBean server](#). For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the [Starting the IBM Social Program Management Data Extractor in server mode](#) related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.batch.launch**.
3. The operator expands **com.ibm.spm.extracttool.batch.launch > JobOperatorProxy > Operations**. The list of buttons that is displayed includes **(i) getExecutions(instanceId:long):List**.
4. The operator specifies the required values in the respective text boxes.
5. The operator clicks the **getExecutions** button.

### Listing the executions of an instance in non-server mode

Enter the following command to list the executions of an instance in non-server mode:

```
java -Dloader.args="JobOperatorProxy.getExecutions {instanceId} " -Dloader.path={path-to-directory-with-db-drivers} -Dloader.main=com.ibm.spm.extracttool.GenericTaskLauncherApplication -Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-name} -jar DataExtractor-<version_no>.jar
```

### Related information

[Starting the IBM Social Program Management Data Extractor in server mode](#)

## Listing the available jobs

An operator can list the available jobs in server mode or in non-server mode.

### List the available jobs in server mode

1. The operator follows steps 1-6 for [starting JConsole and connecting to the MBean server](#). For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the [Starting the IBM Social Program Management Data Extractor in server mode](#) related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.batch.launch**.
3. The operator expands **com.ibm.spm.extracttool.batch.launch > JobOperatorProxy > Operations**. The list of buttons that is displayed includes **(i) getJobNames():Set**.
4. The operator clicks the **getJobNames** button.

### List the available jobs in non-server mode

Enter the following command to list the available jobs in non-server mode:

```
java -Dloader.args="JobOperatorProxy.getJobNames" -Dloader.path={path-to-directory-with-db-drivers} -Dloader.main=com.ibm.spm.extracttool.GenericTaskLauncherApplication -Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-name} -jar DataExtractor-<version_no>.jar
```

### Related information

[Starting the IBM Social Program Management Data Extractor in server mode](#)

## Listing the running executions for a job

An operator can list the running executions for a job in server mode or in non-server mode.

### List the running executions for a job in server mode

1. The operator follows steps 1-6 for [starting JConsole and connecting to the MBean server](#). For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the [Starting the IBM Social Program Management Data Extractor in server mode](#) related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.batch.launch**.
3. The operator expands **com.ibm.spm.extracttool.batch.launch > JobOperatorProxy > Operations**. The list of buttons that is displayed includes **(i) getRunningExecutions(jobName:String):Set**.
4. The operator specifies the required values in the respective text boxes.
5. The operator clicks the **getRunningExecutions** button.

### List the running executions for a job in non-server mode

Enter the following command to run executions for a job in non-server mode:

```
java -Dloader.args=" JobOperatorProxy.getRunningExecutions {jobTypeName}" -Dloader.path={path-to-directory-with-db-drivers} -Dloader.main=com.ibm.spm.extracttool.GenericTaskLauncherApplication -Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-name} -jar DataExtractor-<version_no>.jar
```

### Related information

[Starting the IBM Social Program Management Data Extractor in server mode](#)

## Retry Determination Extraction operations

The operator can run the retry Determination XML Flow operation, the retry Determination Filter Flow CSV operation, and the retry Determination Filter Flow database operation. Operators can perform the three functions in server mode and non-server mode.

**Note:** The retry functionality was originally intended to be used to retry individual determinations that failed as part of a standard extraction. However, the retry functionality can also be used to extract an

individual determination's data by using the determination's ID. The retry functionality can be used regardless of whether that determination was part of a previous extract.

### Retry Determination XML Flow

The operator can run the retry Determination XML Flow in server mode and non-server mode.

#### Retry Determination XML Flow in server mode

1. The operator follows steps 1-6 for starting JConsole and connecting to the MBean server. For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the [Starting the IBM Social Program Management Data Extractor in server mode](#) related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.controller**.
3. The operator expands **JobLaunchingController**. Six buttons are displayed. The button **retryXmlFlowJob** (String, String) (jobInstanceName:String, retryDeterminationIds:String):long is relevant for the retry Determination XML Flow. The `retryXmlFlowJob` operation supports the parameters that are listed in the [Parameters for the IBM Social Program Management Data Extractor's functions](#) related link.
4. The operator specifies the value for a job parameter by completing the applicable text box for one of the two versions of the method.
5. The operator clicks the corresponding **retryXmlFlowJob** button. The Job Execution ID of the scheduled batch job is displayed.

#### Retry Determination XML Flow in non-server mode

Enter the following command to retry Determination XML Flow in non-server mode:

```
java -Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-name} -Dloader.path={path-to-directory-with-db-drivers} -Dloader.main=com.ibm.spm.extracttool.JobLauncherApplication -Dloader.args="XmlFlow name={jobInstanceName} retryDeterminationIds={determination_ids} {OtherJobParameters}" -jar DataExtractor-<version_no>.jar
```

#### Related information

[Starting the IBM Social Program Management Data Extractor in server mode](#)

[Parameters for the IBM Social Program Management Data Extractor's functions](#)

### Retry Determination Filter Flow CSV

The operator can run the retry Determination Filter Flow CSV in server mode and in non-server mode.

#### Retry Determination Filter Flow CSV in server mode

1. The operator follows steps 1-6 for starting JConsole and connecting to the MBean server. For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the [Starting the IBM Social Program Management Data Extractor in server mode](#) related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.controller**.
3. The operator expands **JobLaunchingController**. Six buttons are displayed. The relevant button for retry Determination Filter Flow CSV is **retryFilterFlowCsvJob** (String, String) (jobInstanceName:String, retryDeterminationIds:String):long. The `retryFilterFlowCsvJob` operation supports the parameters that are listed in the [Parameters for the IBM Social Program Management Data Extractor's functions](#) related link.
4. The operator specifies the value for a job parameter by completing the applicable text box for one of the two versions of the method.
5. The operator clicks the corresponding **retryFilterFlowCsvJob** button. The Job Execution ID of the scheduled batch job is displayed.

## Retry Determination Filter Flow CSV in non-server mode

Enter the following command to retry Determination Filter Flow CSV in non-server mode:

```
java -Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-name} -Dloader.path={path-to-directory-with-db-drivers} -Dloader.main=com.ibm.spm.extracttool.JobLauncherApplication -Dloader.args="FilterFlowCsv name={jobInstanceName} retryDeterminationIds={ determination_ids } {OtherJobParameters}" -jar DataExtractor-<version_no>.jar
```

### Related information

[Starting the IBM Social Program Management Data Extractor in server mode](#)

[Parameters for the IBM Social Program Management Data Extractor's functions](#)

### Retry Determination Filter Flow database

The operator can run the retry Determination Filter Flow database in server mode and in non-server mode.

### Retry Determination Filter Flow Database in server mode

1. The operator follows steps 1-6 for [starting JConsole and connecting to the MBean server](#). For more information about calling the functions of the tool through the MBean APIs by using JConsole, see the [Starting the IBM Social Program Management Data Extractor in server mode](#) related link.
2. In the folder view of the MBean pane, the operator expands **com.ibm.spm.extracttool.controller**.
3. The operator expands **JobLaunchingController**. Six buttons are displayed. The relevant button for retry Determination Filter Flow database is **retryFilterFlowDbJob** (String, String) (jobInstanceName:String, retryDeterminationIds:String):long. The **retryFilterFlowDbJob** operation supports the parameters that are listed in the [Parameters for the IBM Social Program Management Data Extractor's functions](#) related link.
4. The operator specifies the value for a job parameter by completing the applicable text box for one of the two versions of the method.
5. The operator clicks the corresponding **retryFilterFlowDbJob** button. The Job Execution ID of the scheduled batch job is displayed.

### Retry Determination Filter Flow database in non-server mode

Enter the following command to retry Determination Filter Flow database in non-server mode:

```
java -Dspring.config.location={path-to-profiles-directory} -Dspring.profiles.active={spring-profile-name} -Dloader.path={path-to-directory-with-db-drivers} -Dloader.main=com.ibm.spm.extracttool.JobLauncherApplication -Dloader.args="FilterFlowDb name={jobInstanceName} retryDeterminationIds={ determination_ids } {OtherJobParameters}" -jar DataExtractor-<version_no>.jar
```

### Related information

[Starting the IBM Social Program Management Data Extractor in server mode](#)

[Parameters for the IBM Social Program Management Data Extractor's functions](#)

## Parameters for the IBM Social Program Management Data Extractor's functions

The parameters for the IBM Social Program Management Data Extractor include scheduling jobs, retrying Determination extraction operations, and searching for the valid Filter Flow attribute paths.

### Parameters for scheduling jobs

Operators can use the parameters of the IBM Social Program Management Data Extractor to schedule the XML Flow job, the Filter Flow CSV job, and the Filter Flow database job.

## Parameters for scheduling the XML Flow job

The parameters for scheduling the XML Flow job are **name**, **startDate**, **endDate**, **maxNumItems**, **pageSize**, and **gridSize**. The XML Flow job returns a long integer that represents a Job Execution ID.

### name

The following lists the characteristics of the **name** parameter:

- **name** is a means of identifying a job instance.
- **name** is a string.
- **name** is a required parameter.

### startDate

The following lists the characteristics of the **startDate** parameter:

- **startDate** is a formatted date string. The format is yyyy-MM-dd.
- **startDate** is used to match determinations that were made on or after the date and time.
- Currently, the API only supports specifying a day in the format yyyy-MM-dd.
- The time for a specified day is taken as midnight of that day. For example: 2018-10-01 is classed as midnight of 10 October 2018.
- **startDate** is an optional parameter.
- If unspecified, the job calculates **startDate** as the **endDate** minus `spm.extract.maxdateinterval` number of days.

### endDate

The following lists the characteristics of the **endDate** parameter:

- **endDate** is a formatted date string. The format is yyyy-MM-dd.
- **endDate** matches determinations that were made before the date and time.
- Currently, the API only supports specifying a day in the format yyyy-MM-dd.
- The time for a specified day is taken as midnight of that day. For example: 2018-10-01 is classed as midnight of 10 October 2018.
- **endDate** is an optional parameter.
- If unspecified, the job calculates **endDate** in the following ways:
  - The current date time, if **startDate** is not specified.
  - The **startDate** plus `spm.extract.maxdateinterval` number of days, if **startDate** is specified.

### maxNumItems

The following lists the characteristics of the **maxNumItems** parameter:

- **maxNumItems** is the maximum number of determinations that are read by an XML Flow or Filter Flow job.
- **maxNumItems** is a positive integer greater than zero.
- **maxNumItems** overrides the `spm.extract.maxitemcount` property in `application.properties`.
- If the operator uses the MBean start operation that does not have the **maxNumItems** parameter, the scheduled job uses the value from the `spm.extract.maxitemcount` configuration property.

### pageSize

The following lists the characteristics of the **pageSize** parameter:

- **pageSize** is a positive integer, *M*.

- The **pageSize** parameter overrides the following `application.properties` properties:
  - `spm.extract.pagesize`
  - `spm.extract.fetchsize`
  - `spm.extract.chunksize`
- To vary the configuration properties so that the properties have different values, do not use the version of the method that uses the preceding properties as parameters. Instead, use the corresponding properties in `application.properties` and use the MBean start operation or operations that do not use the **pageSize** parameter.
- The **pageSize** parameter is used by all batch jobs.
- The **pageSize** parameter is used by the paged, reader SQL query that is run by the all batch jobs types to read the `CREOLECASEDETERMINATION.CREOLECASEDETERMINATIONIDs` from the `CREOLECASEDETERMINATION` rows that match the query criteria, *M* IDs at a time.
- A larger *M* means that fewer reader queries are run against the database.

### **gridSize**

The following lists the characteristics of the **gridSize** parameter:

- **gridSize** is a positive integer, *P*.
- The **gridSize** parameter overrides the `application.properties` property `spm.extract.gridsize`.
- The **gridSize** parameter is used by the Filter Flow database and the XML Flow. The parameter is not used by the Filter Flow CSV.
- When specified as a value greater than 1, the tool creates partitioned step executions. The following changes then apply:
  - A single execution of a job starts multiple step executions in parallel. For example, with **gridSize** *P*, *P* step executions are started with their own dedicated `ItemReader` instance and run the read, process, and write execution cycle. For more information, see Table 1 in the *Batch jobs: job types and their purposes* related link.
  - A dedicated partitioner creates a separate step execution context that effectively divides the range of `CREOLECASEDETERMINATION.CREOLECASEDETERMINATIONIDs` that are to be processed by the job into *P* partitions.
  - Each step execution is then used to read and process the IDs in the partition for which it is responsible.
- In the release of the tool, a thread-based grid execution fabric is used. So, each step execution has a corresponding thread in the same JVM as the IBM Social Program Management Data Extractor itself.
- A larger grid size *P* means that instead of a job execution that reads-processes-writes items serially, the job execution is performed in parallel over *#P* threads. So, the job execution typically takes less time than with a single thread.

### **Related information**

[Batch jobs: job types and their purposes](#)

### **Parameters for scheduling the Filter Flow CSV job**

The parameters for scheduling the Filter Flow CSV job are **name**, **startDate**, **endDate**, **maxNumItems**, and **pageSize**. The Filter Flow CSV job returns a long integer that represents a Job Execution ID.

#### **name**

The following lists the characteristics of the **name** parameter:

- **name** is a means of identifying a job instance.
- **name** is a string.
- **name** is a required parameter.

## **startDate**

The following lists the characteristics of the **startDate** parameter:

- **startDate** is a formatted date string. The format is yyyy-MM-dd.
- **startDate** is used used to match determinations that were made on or after the date and time.
- Currently, the API only supports specifying a day in the format yyyy-MM-dd.
- The time for a specified day is taken as midnight of that day. For example: 2018-10-01 is classed as midnight of 10 October 2018.
- **startDate** is an optional parameter.
- If unspecified, the job calculates **startDate** as the **endDate** minus `spm.extract.maxdateinterval` number of days.

## **endDate**

The following lists the characteristics of the **endDate** parameter:

- **endDate** is a formatted date string. The format is yyyy-MM-dd.
- **endDate** matches determinations that were made before the date and time.
- Currently, the API only supports specifying a day in the format yyyy-MM-dd.
- The time for a specified day is taken as midnight of that day. For example: 2018-10-01 is classed as midnight of 10 October 2018.
- **endDate** is an optional parameter.
- If unspecified, the job calculates **endDate** in the following ways:
  - The current date time, if **startDate** is not specified.
  - The **startDate** plus `spm.extract.maxdateinterval` number of days, if **startDate** is specified.

## **maxNumItems**

The following lists the characteristics of the **maxNumItems** parameter:

- **maxNumItems** is the maximum number of determinations that are read by an XML Flow or Filter Flow job.
- **maxNumItems** is a positive integer greater than zero.
- **maxNumItems** overrides the `spm.extract.maxitemcount` property in `application.properties`.
- If the operator uses the MBean start operation that does not have the **maxNumItems** parameter, the scheduled job uses the value from the `spm.extract.maxitemcount` configuration property.

## **pageSize**

The following lists the characteristics of the **pageSize** parameter:

- **pageSize** is a positive integer, *M*.
- The **pageSize** parameter overrides the following `application.properties` properties:
  - `spm.extract.pagesize`
  - `spm.extract.fetchsize`
  - `spm.extract.chunksize`
- To vary the configuration properties so that the properties have different values, do not use the version of the method that uses the preceding properties as parameters. Instead, use the corresponding properties in `application.properties` and use the MBean start operation or operations that do not use the **pageSize** parameter.
- The **pageSize** parameter is used by all batch jobs.

- The **pageSize** parameter is used by the paged, reader SQL query that is run by the all batch jobs types to read the CREOLECASEDETERMINATION.CREOLECASEDETERMINATIONIDs from the CREOLECASEDETERMINATION rows that match the query criteria, *M* IDs at a time.
- A larger *M* means that fewer reader queries are run against the database.

### Parameters for scheduling a Filter Flow database job

The parameters for scheduling the Filter Flow database job are **name**, **startDate**, **endDate**, **maxNumItems**, **pageSize**, and **gridSize**. The Filter Flow database job returns a long integer that represents a Job Execution ID.

#### name

The following lists the characteristics of the **name** parameter:

- **name** is a means of identifying a job instance.
- **name** is a string.
- **name** is a required parameter.

#### startDate

The following lists the characteristics of the **startDate** parameter:

- **startDate** is a formatted date string. The format is yyyy-MM-dd.
- **startDate** is used used to match determinations that were made on or after the date and time.
- Currently, the API only supports specifying a day in the format yyyy-MM-dd.
- The time for a specified day is taken as midnight of that day. For example: 2018-10-01 is classed as midnight of 10 October 2018.
- **startDate** is an optional parameter.
- If unspecified, the job calculates **startDate** as the **endDate** minus `spm.extract.maxdateinterval` number of days.

#### endDate

The following lists the characteristics of the **endDate** parameter:

- **endDate** is a formatted date string. The format is yyyy-MM-dd.
- **endDate** matches determinations that were made before the date and time.
- Currently, the API only supports specifying a day in the format yyyy-MM-dd.
- The time for a specified day is taken as midnight of that day. For example: 2018-10-01 is classed as midnight of 10 October 2018.
- **endDate** is an optional parameter.
- If unspecified, the job calculates **endDate** in the following ways:
  - The current date time, if **startDate** is not specified.
  - The **startDate** plus `spm.extract.maxdateinterval` number of days, if **startDate** is specified.

#### maxNumItems

The following lists the characteristics of the **maxNumItems** parameter:

- **maxNumItems** is the maximum number of determinations that are read by an XML Flow or Filter Flow job.
- **maxNumItems** is a positive integer greater than zero.
- **maxNumItems** overrides the `spm.extract.maxitemcount` property in `application.properties`.
- If the operator uses the MBean start operation that does not have the **maxNumItems** parameter, the scheduled job uses the value from the `spm.extract.maxitemcount` configuration property.

## pageSize

The following lists the characteristics of the **pageSize** parameter:

- **pageSize** is a positive integer, *M*.
- The **pageSize** parameter overrides the following `application.properties` properties:
  - `spm.extract.pagesize`
  - `spm.extract.fetchsize`
  - `spm.extract.chunksize`
- To vary the configuration properties so that the properties have different values, do not use the version of the method that uses the preceding properties as parameters. Instead, use the corresponding properties in `application.properties` and use the MBean start operation or operations that do not use the **pageSize** parameter.
- The **pageSize** parameter is used by all batch jobs.
- The **pageSize** parameter is used by the paged, reader SQL query that is run by the all batch jobs types to read the `CREOLECASEDETERMINATION.CREOLECASEDETERMINATIONIDs` from the `CREOLECASEDETERMINATION` rows that match the query criteria, *M* IDs at a time.
- A larger *M* means that fewer reader queries are run against the database.

## gridSize

The following lists the characteristics of the **gridSize** parameter:

- **gridSize** is a positive integer, *P*.
- The **gridSize** parameter overrides the `application.properties` property `spm.extract.gridsize`.
- The **gridSize** parameter is used by the Filter Flow database and the XML Flow. The parameter is not used by the Filter Flow CSV.
- When specified as a value greater than 1, the tool creates partitioned step executions. The following changes then apply:
  - A single execution of a job starts multiple step executions in parallel. For example, with **gridSize** *P*, *P* step executions are started with their own dedicated `ItemReader` instance and run the read, process, and write execution cycle. For more information, see Table 1 in the *Batch jobs: job types and their purposes* related link.
  - A dedicated partitioner creates a separate step execution context that effectively divides the range of `CREOLECASEDETERMINATION.CREOLECASEDETERMINATIONIDs` that are to be processed by the job into *P* partitions.
  - Each step execution is then used to read and process the IDs in the partition for which it is responsible.
- In the release of the tool, a thread-based grid execution fabric is used. So, each step execution has a corresponding thread in the same JVM as the IBM Social Program Management Data Extractor itself.
- A larger grid size *P* means that instead of a job execution that reads-processes-writes items serially, the job execution is performed in parallel over *#P* threads. So, the job execution typically takes less time than with a single thread.

### Related information

[Batch jobs: job types and their purposes](#)

## Parameters for retrying Determination extraction operations

Operators can use the parameters of the IBM Social Program Management Data Extractor to run the retry Determination XML Flow job, the retry Determination Filter Flow CSV job, and the retry Determination Filter Flow database job.

### Parameters for the retry Determination XML Flow job

The parameters for the retry Determination XML Flow job are **name** and **retryDeterminationIds**. The retry Determination XML Flow job returns a long integer that represents a Job Execution ID.

#### **name**

The following lists the characteristics of the **name** parameter:

- **name** is a means of identifying a job instance.
- **name** is a string.
- **name** is a required parameter.

#### **retryDeterminationIds**

The following lists the characteristics of the **retryDeterminationIds** parameter:

- **retryDeterminationIds** is a string.
- **retryDeterminationIds** is a comma-delimited list of determination IDs that corresponds to values for CREOLECASEDETERMINATION.CREOLECASEDETERMINATIONID in the Social Program Management database.
- **retryDeterminationIds** is a required parameter.

### Parameters for the retry Determination Filter Flow CSV job

The parameters for the retry Determination Filter Flow CSV job are **name** and **retryDeterminationIds**. The retry Determination Filter Flow CSV job returns a long integer that represents a Job Execution ID.

#### **name**

The following lists the characteristics of the **name** parameter:

- **name** is a means of identifying a job instance.
- **name** is a string.
- **name** is a required parameter.

#### **retryDeterminationIds**

The following lists the characteristics of the **retryDeterminationIds** parameter:

- **retryDeterminationIds** is a string.
- **retryDeterminationIds** is a comma-delimited list of determination IDs that corresponds to values for CREOLECASEDETERMINATION.CREOLECASEDETERMINATIONID in the Social Program Management database.
- **retryDeterminationIds** is a required parameter.

### Parameters for the retry Determination Filter Flow database job

The parameters for the retry Determination Filter Flow database job are **name** and **retryDeterminationIds**. The retry Determination Filter Flow database job returns a long integer that represents a Job Execution ID

#### **name**

The following lists the characteristics of the **name** parameter:

- **name** is a means of identifying a job instance.

- **name** is a string.
- **name** is a required parameter.

### **retryDeterminationIds**

The following lists the characteristics of the **retryDeterminationIds** parameter:

- **retryDeterminationIds** is a string.
- **retryDeterminationIds** is a comma-delimited list of determination IDs that corresponds to values for CREOLECASEDETERMINATION.CREOLECASEDETERMINATIONID in the Social Program Management database.
- **retryDeterminationIds** is a required parameter.

## **Parameters for generating the Filter Flow database tables DDL**

Operators can use return values to generate the Creation DDL for the attribute value table and to generate the Drop DDL for the attribute value tables.

### **Generating the Creation DDL for the attribute value tables**

To create the Filter Flow attribute value tables for the Data Warehouse, the return value is DDL.

### **Generating the Drop DDL for the attribute value tables**

To create the Filter Flow attribute value tables for the Data Warehouse, the return value is DDL.

## **Parameters for searching for the valid Filter Flow attribute paths**

Operators can use the **attributePathPattern** parameter to search for the valid Filter Flow attribute paths.

The parameter **attributePathPattern** is an optional parameter that takes a valid regular expression pattern. The return value is a list of all valid filter flow attribute paths, optionally filtered by the parameter.

## **Parameters for generating XSDs for XML Flow- ingested determination XML**

Operators can use parameters to list all valid CER static XSDs, list all valid CER products, read CER product details, print a CER product period XSD, list all valid CER ruleset XSD, print a CER ruleset XSD, and print a CER static XSD. Operators can perform all these functions in server mode and non-server mode.

### **List all valid CER static XSDs**

#### **Return value**

The function returns a listing of all available CER static XSDs as a string in JSON format.

### **List all valid CER products**

#### **Return value**

The function returns a listing of all available CER products as a string in JSON format.

### **Read CER product details, including all periods and each period's dependent XSDs**

#### **productid**

The unique identifier for the product.

#### **Return value**

The function returns a listing of all available CER period XSDs for a CER product as a string in JSON format.

### **Print a CER product period XSD**

#### **productid**

The unique identifier for the product.

**effective\_date**

The parameter **effective\_date** is a formatted date string with the date format is yyyy-MM-dd, where the following characteristics apply:

- Represents the date for which to produce the XSD.
- The function is contained in only one product period.

**Return values**

The function returns a string in XSD format:

- Save to a file and name it using the name that is given for it in `listCERPeriodsByProduct`.
- Defines a portion of the structure of the DETERMINATION XML that differs per product per period.

**List all valid CER ruleset XSDs****Return value**

The function returns a listing of all available CER ruleset XSDs as a string in JSON format.

**Print a CER ruleset XSD****ruleSetName**

The unique identifier for the ruleset.

**Return values**

The function returns a string in XSD format:

- Save to a file and name it using the name that is given to it in `listCERPeriodsByProduct / listCERRuleSets`.
- Defines the portion of the structure of the DETERMINATION XML that is related to the rule set.

**Print a CER static XSD****schemaName**

The unique name for the static schema.

**Return values**

The function returns a string in XSD format:

- Save to a file and name it using the name that is given to it in `listCERPeriodsByProduct / listCERStatics`.
- Defines the static portion of the structure of the DETERMINATION XML.

**Parameters for batch jobs**

Operators can use parameters to perform 10 batch jobs. The batch jobs include stopping a job, getting job parameters, and listing the executions of an instance.

**Stop a job****executionId**

The unique identifier for the product.

**Return value**

The return value is true if the message was successfully sent, although it does not ensure that the job is stopped.

**Abandon a job execution****executionId**

The unique identifier for the execution.

**Return value**

The return value is the ID of the execution that was started.

## **Get job parameters**

### **executionId**

The unique identifier for the execution.

### **Return value**

The return value is the job parameters that were used to start the associated instance.

## **Restart a job**

### **executionId**

The unique identifier for the execution.

### **Return value**

The return value is the ID of the execution that was started.

## **Summarize step executions**

### **executionId**

The unique identifier for the execution.

### **Return value**

The return value is a map of step execution ID to string that summarizes the state of the execution.

## **Summarize job execution**

### **executionId**

The unique identifier for the execution.

### **Return value**

The return value is a string that summarizes the state of the job execution.

## **List job instances**

The list job instances batch job has three parameters:

### **jobname**

The job name that is given to all the instances.

### **start**

The start index of the instances.

### **count**

The maximum number of values.

### **Return value**

The return value is the ID values of the instances.

## **List executions of an instance**

### **instanceId**

The unique identifier for the instance.

### **Return value**

The return value is the ID values of all the executions that are associated with this instance.

## **List available jobs**

### **Return value**

The return value is a set of job names.

## **List running executions for a job**

### **jobname**

The job name that is given to all the instances.

**Return value**

The return value is the ID values of the running execution instances.

## **Stopping the IBM Social Program Management Data Extractor**

---

To stop the IBM Social Program Management Data Extractor when the tool is running in server mode, the operator issues **Ctrl+C** in the command window where the operator started the tool.

---

## Chapter 9. Appendixes

Six appendixes provide background information for using the IBM Social Program Management Data Extractor.

---

### Appendix A: Determination BLOB XML structure

---

When the operator opens a determination in the Social Program Management application, a set of decisions for different periods is displayed. When the operator selects a period, a list of categories is displayed. Under each category, a set of values is displayed.

#### The Determination Data XML file

To view a full schema definition, see the *Generating schema definitions for Determination BLOB XML* related link. The Determination Data XML file, which is based on a display rules model, uses a similar hierarchy structure. The Determination Data XML file delivers the XML into the category page only for the category for the period under consideration. The XML file starts with high-level information about the determination. Then, the XML file branches into an overall eligibility timeline where contiguous decisions with the same eligibility are merged. The XML file then displays a list of categories in the UI mapped to timelines of display rules results, where the display rules results are the category page data. For populated entries in the map, the start dates are the start dates of decision splits - see the following note. The end dates are the day before the next start date. For an open-ended case, there is no end date for the final decision.

**Note:** Typically, there are many more decision splits than there are CaseDecision records for the same determination. CaseDecision records are typically created for discrete periods of eligibility or ineligibility. For example, there might be three decision periods in a row that are eligible but with different display rules results. In that case, the decision periods map to the one CaseDecision record. The associated caseDecisionId is recorded on the DECISIONS table and on the Decision nodes of the IngestedData on DETERMINATIONXML.XMLVALUE.

#### The display rules results

The display rules results are formatted in escaped XML. The structure of the display rules results is predictable based on the following design of the corresponding CREOLERuleSet definition:

1. The rule class on top corresponds to the display category  
<StreamlineMedicaidIncomeCategory>.
2. Under the display category are attributes of the category <isInPostPartumPeriodTimeline domain="SVR\_BOOLEAN">>false</isInPostPartumPeriodTimeline>.
3. Under any attributes whose type is a list of simple types is a list of item nodes that contain values as text: <eligibleMembersName> <Item domain="SVR\_UNBOUNDED\_STRING">John Stephens</Item> <Item domain="SVR\_UNBOUNDED\_STRING">Mary Mitchell</Item> <eligibleMembersName>.
4. Under any attributes whose type is another rule class are attributes of that rule class:  
<taxFilerMAGIDisplay> <displayAmount domain="CURAM\_AMOUNT">0</displayAmount>  
<displayType domain="SVR\_UNBOUNDED\_STRING">MAGI</displayType>  
<taxFilerMAGIDisplay>.
5. Under any attribute whose type is a list of rule classes is a list of item nodes that contain all attributes of the sub-rule class. The sub-rule class itself does not get an element.

```
<taxFilerMAGIDisplayList> <Item> <displayAmount domain="CURAM_AMOUNT">0</displayAmount>  
<displayType domain="SVR_UNBOUNDED_STRING">MAGI</displayType> </Item>  
<taxFilerMAGIDisplayList>
```

6. The previous sub-rule class patterns can be nested.

## Related information

[Generating schema definitions for Determination BLOB XML](#)

## Appendix B: Logging

The IBM Social Program Management Data Extractor uses Apache Log4j 2 as its logging service.

The uber jar contains a default `log4j2.properties`. The `log4j2.properties` is used to specify the logging level for the packages that are inside the jar and the name of the log file.

A copy of IBM's `log4j2.properties` is included in the `{INSTALL_LOCATION}/samples` directory.

For a support scenario, the customer must copy the `log4j2.properties` file to the same directory where the customer stores the database drivers that are used by the tool. The next time that the customer runs the IBM Social Program Management Data Extractor, the tool selects this file rather than the file that is packaged in the uber jar.

For more verbose logging to help troubleshoot any issues, the operator can customize the `log4j2.properties` file. The operator can also change the name of the log file to which the application logs. For more information about Apache Log4j 2, see the [Apache Log4j 2](#) related link.

## Related information

[Apache Log4j 2](#)

## Appendix C: An extract of the rule class that declares the attributes that are requested

The sample in this topic shows a snippet of a `StreamlineMedicaidIncomeCategory` rule class.

```
<Class extends="DefaultCase" extendsRuleSet="DefaultProductDecisionDetailsRuleSet" name =
"StreamlineMedicaidIncomeCategory"...>
  ...
  <Attribute name = "eligibleMembersName">
    <Annotations>
      <Display/>
    </Annotations>
    <type>
      <javaclass name = "curam.creole.value.Timeline">
        <javaclass name = "List">
          <javaclass name = "String"/>
        </javaclass>
      </javaclass>
    </type>
  </Attribute>
  <Attribute name="membersEligibleOrNotMessage">
    <Annotations>
      <Display/>
    </Annotations>
    <type>
      <javaclass name = "curam.creole.value.Timeline">
        <javaclass name = "curam.creole.value.Message"/>
      </javaclass>
    </type>
  </Attribute>
  ...
  <Attribute name=" householdNonFinancialNonEligibleTimeline">
    <Annotations>
      <Display/>
    </Annotations>
    <type>
      <javaclass name = "curam.creole.value.Timeline">
        <javaclass name = "Boolean"/>
      </javaclass>
    </type>
  </Attribute>
  ...
</Class>
```

```

<Attribute name = "eligibleMembersIncomePercentage">
  <Annotations>
    <Display/>
  </Annotations>
  <type>
    <javaclass name = "curam.creole.value.Timeline">
      <javaclass name = "List">
        <javaclass name = "curam.creole.value.Message"/>
      </javaclass>
    </javaclass>
  </type>
  ""
</Attribute>
<Attribute name = "sMMemIncomeCategorySubscreens">
  <Annotations>
    <DisplaySubscreen/>
  </Annotations>
  <type>
    <javaclass name = "curam.creole.value.Timeline">
      <javaclass name = "List" >
<ruleclass name = "StreamlineMedicaidIncomeSubScreen" ruleset =
"StreamlineMedicaidDisplayRuleSet"/>
      </javaclass>
    </javaclass>
  </type>
  ""
</Attribute>

```

For more information about attribute value CSVs, see the *The output of the Filter Flow CSV job* related link.

### Related information

[The output of the Filter Flow CSV job](#)

## Appendix D: Securing a connection to the Social Program Management and Data Warehouse databases over TLS 1.2

The procedure for securing a connection to the Social Program Management and Data Warehouse databases over TLS 1.2 is determined by the database that you use.

### Generic information about how to secure a connection

The following criteria applies to securing a connection to the Social Program Management (SPM) and Data Warehouse (DW) databases over TLS 1.2:

- The instructions were verified against the following database versions:
  - Db2 v10.5.0.9
  - Oracle v12.2.0.1.0
- For simplicity, the instructions assume that SPM and DW tables are on the same database. This is not a recommended configuration. For each database, the steps for securing the connection against the different databases must be repeated.
- Customers can import the exported database certificates to the cacerts truststore for the Java SE Development Kit (JDK) that is used to run the tool. As an alternative, the customer can create a separate truststore for the IBM Social Program Management Data Extractor and point the tool to the truststore by using the system property `javax.net.ssl.trustStore`, as shown in the following example:

```

-Djavax.net.ssl.trustStore=<trustStoreFile> -
Djavax.net.trustStorePassword=<trustStorePassword>

```

For more information, see *Table 6: Customizable Items* in the *Java Secure Socket Extension (JSSE) Reference Guide* related link.

- To debug issues with the SSL/TLS, the operator specifies the `javax.net.debug` system property when the operator starts the IBM Social Program Management Data Extractor, that is, -

`Djavax.net.debug=all`. For more information about how to debug issues with SSL/TLS, see the [Debugging SSL/TLS Connections](#) related link.

### Securing a connection for Db2 (Db2 v10.5.0.9)

1. For the database, the operator performs a set of steps. For more information about the steps to perform, see the [Configuring Secure Sockets Layer \(SSL\) support in a Db2 instance](#) related link.
2. For the client, that is the IBM Social Program Management Data Extractor, the operator copies the exported public certificate of the database to the client. The operator uses `keytool` to import the public certificate to the truststore that the operator configured for use by the IBM Social Program Management Data Extractor. Customers can import the exported database certificates to the `cacerts` truststore for the Java SE Development Kit (JDK) that is used to run the tool. As an alternative, the customer can create a separate truststore for the IBM Social Program Management Data Extractor and point the tool to the truststore by using the system properties `javax.net.ssl.trustStore`, as shown in the following example:

```
-Djavax.net.ssl.trustStore=<trustStoreFile> -  
Djavax.net.trustStorePassword=<trustStorePassword>
```

For more information, see [Table 6: Customizable Items](#) in the [Java Secure Socket Extension \(JSSE\) Reference Guide](#) related link. Operators must refer to the `keytool` documentation that applies to the JDK that the operator is using.

3. The operator updates the data source URLs for the SPM and the Data Warehouse data source to point to the secured port for the operator's database server. For example, where the database server name is `myhost.com`, the database name is `DATABASE`, and the secure port is `50001`, then the following sample shows the URL:

```
spm.extract.spm.datasource.url=jdbc:db2://myhost.com:50001/  
DATABASE:sslConnection=true  
spm.extract.dw.datasource.url=jdbc:db2://myhost.com:50001/  
DATABASE:sslConnection=true
```

### Oracle (v12.2.0.1.0)

1. For the database, the operator performs a set of steps. For more information about the steps to perform, see the [B Appendix: Secure JDBC with Oracle 12c Database](#) related link.
2. For the client, that is the IBM Social Program Management Data Extractor, the operator copies the public certificate of the exported database to the client. The operator uses the `keytool` to import the public certificate to the truststore that the operator configured for use by the IBM Social Program Management Data Extractor. Customers can import the exported database certificates to the `cacerts` truststore for the Java SE Development Kit (JDK) that is used to run the tool. As an alternative, the customer can create a separate truststore for the IBM Social Program Management Data Extractor and point the tool to the truststore by using the system properties `javax.net.ssl.trustStore`, as shown in the following example:

```
-Djavax.net.ssl.trustStore=<trustStoreFile> -  
Djavax.net.trustStorePassword=<trustStorePassword>
```

For more information, see [Table 6: Customizable Items](#) in the [Java Secure Socket Extension \(JSSE\) Reference Guide](#) related link. Operators must refer to the `keytool` documentation that applies to the JDK that the operator is using.

3. The operator updates the `Spring Profile application.properties` to use the correct driver class and the Java Database Connectivity (JDBC) URLs for the secured data sources. For example, where the

database server name is myhost.com, the database service name is ORCL, and the secure port is 2484, then the following sample shows the URL:

```
spm.extract.spm.datasource.driver-class-name=oracle.jdbc.OracleDriver
spm.extract.spm.datasource.url=jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCPS)
(HOST=myhost.com)(PORT=2484))(CONNECT_DATA=(SERVICE_NAME=ORCL)))
spm.extract.dw.datasource.url=jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCPS)
(HOST=myhost.com)(PORT=2484))(CONNECT_DATA=(SERVICE_NAME=ORCL)))
spm.extract.dw.datasource.driver-class-name=oracle.jdbc.OracleDriver
```

**Note:** For Oracle 12.2, the customer must use the ojdbc8.jar database driver JAR.

### Related information

[Java Secure Socket Extension \(JSSE\) Reference Guide](#)

[Debugging SSL/TLS Connections](#)

[Configuring Secure Sockets Layer \(SSL\) support in a Db2 instance](#)

[B Appendix: Secure JDBC with Oracle 12c Database](#)

## Appendix E: Configuring the IBM Social Program Management Data Extractor JMX/MBean API for remote access

---

Customers can perform a series of steps to expose the MBean API. However, to expose the MBean API customers must secure the connection over TLS v1.2 and configure client certificate-based authentication as a requirement.

### About this task

IBM does not recommend that customers expose remote access to the MBean functionality. However, if necessary, customers can expose the MBean API if customers secure the connection over TLS v1.2 and configure client certificate-based authentication as a requirement. For more information about securing the JMX agent, see the *Monitoring and Management Using JMX Technology* related link.

### Procedure

1. Create a valid certificate for the IBM Social Program Management Data Extractor. For information about how to create a valid certificate, see the *Using SSL* related link.
2. Use **keytool** to export the certificate file.
3. Copy the exported certificate file to the remote client computer.
4. On the client computer, use **keytool** to import the certificate to a truststore that is used when the JConsole is started.
5. Create a valid certificate for the JConsole client that is used to connect to the IBM Social Program Management Data Extractor. For information about how to create a valid certificate, see the *Using SSL* related link. The certificate is used to authenticate the client to the IBM Social Program Management Data Extractor.
6. Use **keytool** to export the certificate.
7. Copy the exported client certificate to the IBM Social Program Management Data Extractor computer.
8. Use **keytool** to import the client certificate to a truststore on the IBM Social Program Management Data Extractor computer.
9. Designate a port on the IBM Social Program Management Data Extractor host computer as the JMX port for the IBM Social Program Management Data Extractor Java process.
10. Configure the firewall of your operating system to permit remote access to the JMX port from the permitted IP addresses.
11. On the IBM Social Program Management Data Extractor computer, start the IBM Social Program Management Data Extractor in server mode with the following list of options:

```

java
-Djavax.net.ssl.keyStore=<data-extractor-keystore>
-Djavax.net.ssl.keyStorePassword=<data-extractor-keystore-password>
-Djavax.net.ssl.trustStore=<data-extractor-truststore>
-Djavax.net.ssl.trustStorePassword=<data-extractor-truststore-password>
-Djava.rmi.server.hostname=<data-extractor-machine-ip-address>
-Dcom.sun.management.jmxremote.port=<data-extractor-jmx-port>
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl.need.client.auth=true
-Dcom.sun.management.jmxremote.ssl=true
-Dloader.path=<data-extractor-lib-directory>
-Dspring.config.location=<data-extractor-spring-profiles-directory>
-Dspring.profiles.active=<data-extractor-spring-profile-name>
-jar DataExtractor-<version_no>.jar

```

12. When the server is started, the server can accept connections. From the client computer, use the following code to start the JConsole:

```

jconsole
-J-Djavax.net.ssl.trustStore=<jconsole-trust-store>
-J-Djavax.net.ssl.trustStorePassword=<jconsole-trust-store-password>
-J-Djavax.net.ssl.keyStore=<jconsole-key-store>
-J-Djavax.net.ssl.keyStorePassword=<jconsole-key-store-password>

```

13. When the user interface (UI) of the JConsole starts, click the **Remote Process** button.
14. Specify the connection details in the text box that is below the radio button. Use the format: <data-extractor-machine-ip-address>:<data-extractor-jmx-port>.
15. Click the **Connect** button.
16. For more information about how to call the tool's functions, see the *Running the IBM Social Program Management Data Extractor functions in server mode and non-server mode* related link.

To debug issues with the SSL/TLS, specify the `javax.net.debug` system property, which is `-Djavax.net.debug=all`. For more information, see the *Debugging SSL/TLS Connections* related link. Use the following code for the IBM Social Program Management Data Extractor:

```

java
-Djavax.net.debug=all
-Djavax.net.ssl.keyStore=<data-extractor-keystore>
-Djavax.net.ssl.keyStorePassword=<data-extractor-keystore-password>
-Djavax.net.ssl.trustStore=<data-extractor-truststore>
-Djavax.net.ssl.trustStorePassword=<data-extractor-truststore-password>
-Djava.rmi.server.hostname=<data-extractor-machine-ip-address>
-Dcom.sun.management.jmxremote.port=<data-extractor-jmx-port>
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl.need.client.auth=true
-Dcom.sun.management.jmxremote.ssl=true
-Dloader.path=<data-extractor-lib-directory>
-Dspring.config.location=<data-extractor-spring-profiles-directory>
-Dspring.profiles.active=<data-extractor-spring-profile-name>
-jar DataExtractor-<version_no>.jar

```

Use following code for JConsole:

```

jconsole
-J-Djavax.net.debug=all
-J-Djavax.net.ssl.trustStore=<jconsole-trust-store>
-J-Djavax.net.ssl.trustStorePassword=<jconsole-trust-store-password>
-J-Djavax.net.ssl.keyStore=<jconsole-key-store>
-J-Djavax.net.ssl.keyStorePassword=<jconsole-key-store-password>

```

### What to do next

A customer might want to perform the extra step of enabling password authentication for the JMX agent. The step is not verified by IBM, but for more information about how to enable password authentication see the *Monitoring and Management Using JMX Technology* related link.

### Related information

[Monitoring and Management Using JMX Technology](#)

[Using SSL](#)

[Debugging SSL/TLS Connections](#)

[Running the IBM Social Program Management Data Extractor's functions in server mode and non-server mode](#)

## Appendix F: How to perform profiling by using J9 VM tracing

---

Customers who select to use a J9 VM (OpenJ9 or IBM J9) can use J9's tracing facility, XTrace, to profile the performance of key methods in the extraction batch jobs.

The IBM J9 VM component of the SDK was contributed to the Eclipse Foundation as the Eclipse OpenJ9 project in September 2017. For more information, see the *IBM SDK, Java Technology Edition, Version 8* related link.

For more information about the IBM JDK documentation for XTrace, see the *How to use IBM Java -Xtrace* related link. For more information about the OpenJ9 documentation for XTrace, see the *-Xtrace* related link.

The `.trc` file can be formatted by using the trace formatter. For more information about the trace formatter, see the *Trace formatter (traceformat)* related link. Optionally, customers might use the IBM Health Center client to analyze the `.trc` files. For more information about the IBM Health Center client, see the *Installing the Health Center client* related link. The Health Center Client can be installed as an add-on for the IBM Support Assistant or as an Eclipse plug-in. For more information, see the *IBM Support Assistant*, login is required, and the *IBM WebSphere tools* related links.

To profile the execution time for specific methods, specify the method whose execution time you want to measure as an option to the J9 Java virtual machine (JVM), for example:

```
Xtrace:none,output={methods%p_#.trc,100m,5},maximal=mt,methods={org/springframework/batch/item/database/JdbcPagingItemReader.doReadPage}
```

The following list outlines each metric, its associated method, and the Xtrace string to use to measure the method's execution time:

### Timing for the initial read of IDs

#### Method

```
org.springframework.batch.item.database.JdbcPagingItemReader.doReadPage()
```

#### Corresponding Xtrace string

```
org/springframework/batch/item/database/JdbcPagingItemReader.doReadPage
```

### Timing for the retrieval of IDs from BLOB CREOLECaseDeterminationData

#### Method

```
com.ibm.spm.extracttool.batch.processors.CREOLEDeterminationDataProcessor.process(Long)
```

#### Corresponding Xtrace string

```
com/ibm/spm/extracttool/batch/processors/CREOLEDeterminationDataProcessor.process
```

## Timing for determination processing

### Filter Flow jobs

#### Method

```
com.ibm.spm.extracttool.batch.processors.FilterJobProcessor.process(CREOLEDeterminatioPojo)
```

#### Corresponding trace string

```
com/ibm/spm/extracttool/batch/processors/FilterJobProcessor.process
```

### XML Flow jobs

#### Method

```
com.ibm.spm.extracttool.batch.processors.XMLFlowProcessor.process(CREOLEDeterminatioPojo)
```

#### Corresponding Xtrace string

```
com/ibm/spm/extracttool/batch/processors/XMLFlowProcessor.process
```

## Write-out time to database

### Filter Flow jobs

#### Method

```
com.ibm.spm.extracttool.batch.writers.FilterJobWriter.write(List<? extends TabularDataAggregation>)
```

#### Corresponding Xtrace string

```
com/ibm/spm/extracttool/batch/writers/FilterJobWriter.write
```

### XML Flow jobs

#### Method

```
com.ibm.spm.extracttool.batch.writers.XMLDeterminationWriter.write(List<? extends XMLDeterminationPojo>)
```

#### Corresponding Xtrace string

```
com/ibm/spm/extracttool/batch/writers/XMLDeterminationWriter.write
```

## Job execution time

### Method

```
org.springframework.batch.core.job.AbstractJob.execute(JobExecution)
```

### Corresponding Xtrace string

```
org/springframework/batch/core/job/AbstractJob.execute
```

## Sample command with method profiling

Enter the following command, on one line, to start the tool in non-server mode, schedule an XML Flow job, and exit when the job completes. The `.trc` file represents a profile of the timings for the preceding methods.

```
java "-Xtrace:none,output={methods%p_#.trc,100m,5},maximal=mt,methods={org/springframework/batch/item/database/JdbcPagingItemReader.doReadPage,
```

```
com/ibm/spm/extracttool/batch/processors/CREOLEdeterminationDataProcessor.process,  
com/ibm/spm/extracttool/batch/processors/XMLFlowProcessor.process,  
com/ibm/spm/extracttool/batch/writers/FilterJobWriter.write,  
com/ibm/spm/extracttool/batch/writers/XMLDeterminationWriter.write,  
com/ibm/spm/extracttool/batch/processors/FilterJobProcessor.process,  
org/springframework/batch/core/job/AbstractJob.execute}" -Dspring.config.location=./profiles/  
-Dspring.profiles.active=Db2 -Dloader.path=./db2_drivers/  
-Dloader.main=com.ibm.spm.extracttool.JobLauncherApplication  
-Dloader.args="XmlFlow name=XmlFlowJob_20190702_003 startDate=1970-01-01 endDate=2019-07-01"  
-jar DataExtractor-<version_no>.jar
```

### **Related information**

[IBM SDK, Java Technology Edition, Version 8](#)

[How to use IBM Java -Xtrace](#)

[-Xtrace](#)

[Trace formatter \(traceformat\)](#)

[Installing the Health Center client](#)

[IBM Support Assistant](#)

[IBM WebSphere tools](#)

## Notices

---

This information was developed for products and services offered in the United States.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing IBM Corporation North Castle Drive, MD-NC119 Armonk, NY 10504-1785 US*

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*Intellectual Property Licensing Legal and Intellectual Property Law IBM Japan Ltd. 19-21, Nihonbashi-Hakozakicho, Chuo-ku Tokyo 103-8510, Japan*

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

*IBM Director of Licensing IBM Corporation North Castle Drive, MD-NC119 Armonk, NY 10504-1785 US*

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

#### COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

## Privacy Policy considerations

---

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies or other similar technologies that collect each user's name, user name, password, and/or other personally identifiable information for purposes of session management, authentication, enhanced user usability, single sign-on configuration and/or other usage tracking and/or functional purposes. These cookies or other similar technologies cannot be disabled.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at <http://www.ibm.com/privacy> and IBM's Online Privacy Statement at <http://www.ibm.com/privacy/details> the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at <http://www.ibm.com/software/info/product-privacy>.

## Trademarks

---

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at <http://www.ibm.com/legal/copytrade.shtml>.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

The registered trademark Linux is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other names may be trademarks of their respective owners. Other company, product, and service names may be trademarks or service marks of others.



Part Number:

(1P) P/N: