SSF Tools: Generic Importer User Guide

Document Revision History

Revision Date	Written/Edited By	Comments
October 2016	Christian Cairney	Initial release with SSD v2
February 2017	Paul Wheeler	Added IdentityIQ version compatibility information

Table of Contents

Overview

The Generic Importer is a framework which allows for the rapid development of miscellaneous data imports into IdentityIQ. The framework abstracts away from the data source so business rules implemented using the data are source agnostic. The following high level features are available:

- Import data sources from a Delimited Text File, Excel Spreadsheet or a JDBC Database
- Schema manipulation and Rule based Transforms
- Set of Importer tools

The tool is designed to replace boiler plate code commonly found in other importers; some of the advantages of using the Generic Importer are:

- No need to write iterators, just use a Rule to process a row of data at a time
- Business rules can be used as-is from one data source to another.
- Source feeds no longer need to conform to schema names expected by your import task, rapid Schema manipulation allows Generic Imports to process the feeds without any changes to business rules or the source data.
- Enhanced transform helper removes the need of further boiler plate code when populating objects with row data.

The Generic Importer is compatible with IdentityIQ version 6.3 and later.

Installation

The Generic Importer consists of the following class files:

Filename	Description
GenricImportTaskExecutor.java	Task Executor used to instantiate the Controller class from an IdentityIQ Task.
ImporterUtil.java	Set of helper functions
Transmogrifier.java	Transform class to merge row data with SailPoint Object classes
Parser.java	Parses strings and builds an object hierarchy of the data.
Schema.java	Maintains the schema of a given iterator and allows for data transformations
GenericImportController.java	The main Generic Importer Controller class, responsible for processing the Iterators and applying schema changes etc.
GenericImport.java	Interface for the Generic Importer iterators
AbstractGenericImport.java	Abstract class based on the Generic Import interface/
ExcelSAXImport.java	Excel low memory footprint iterator
JDBCImport.java	JDBC Database iterator
TextFileImport.java	Delimited file iterator

The configuration files are:

Filename	Description
TaskDef-GenericImporter.xml	Task Definition for the import

The following template rule files can be used as they are prepared with the IdentityIQ Deployment Accelerator in mind:

Filename	Description
Template_Init.xml	Init, transform, row and finalise rules setup as a template with additional meta data to enable the IdentityIQ Developer Accelerator.
Template_Transform.xml	
Template_Row.xml	
Template_Finalise.xml	

These files are included in the SSD and automatically deployed with your project using the SSB. Follow the SSB instructions to create a build for your environment and deploy the files.

Using the Generic Importer

The Generic Importer iterates through the data feed supplied, allowing for transforms then business process applied to it. The basic flow is:

- The data feed is opened by the Generic Importer
- If any initialization code is required, this can be placed in the Init Rule
- Any Rapid Schema Transform changes needed are initialized
- Then the data feed is iterated over; for each iteration:
    - The Data Row, represented as a Map, is built from the iterator
    - Any Rapid Schema Transform configurations are executed and transform the Data Row Map.
    - If further transforms are required, The Data Row Map is presented to the Transform Rule and returns a transformed version of the Data Row (Map).
    - A Data Row Map is then presented to the Row Rule, where the business rules are contained to process the Data Row Map.
- Run the Finalize Rule (if one exists) once the iterator is exhausted.

The following diagram shows the data flow in the importer:

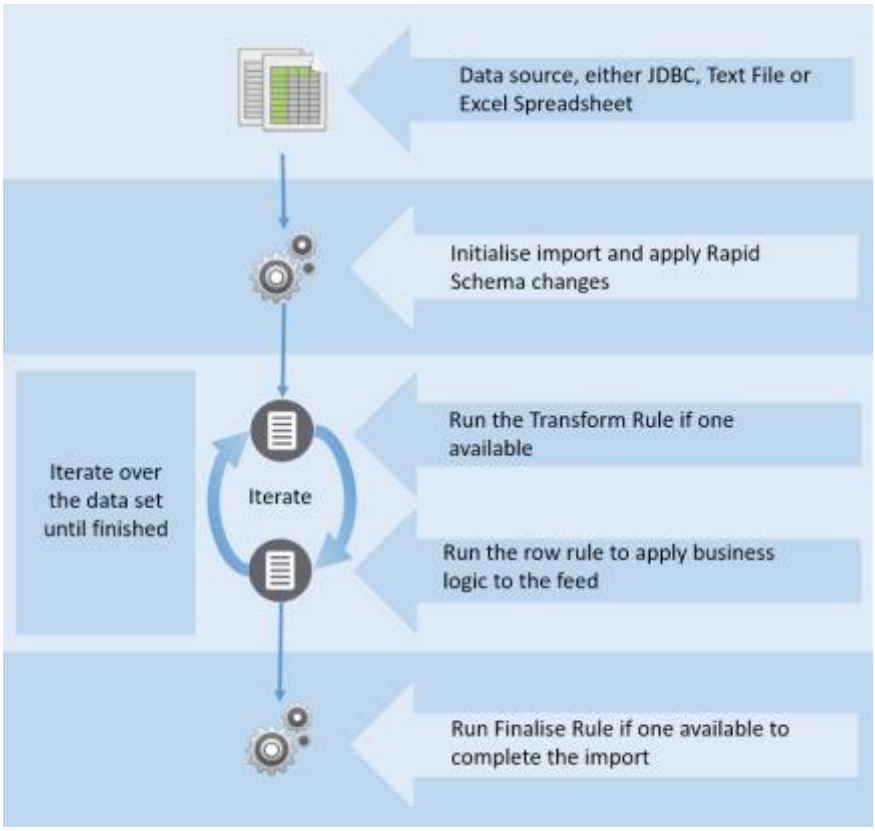
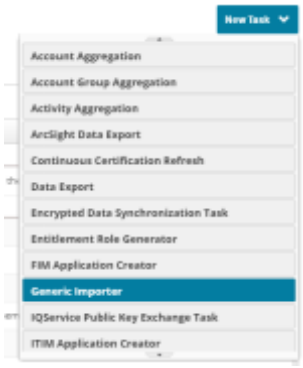


Figure 1: GenericImporter overview

To enable the Generic Importer you must first set up a task; this will be discussed in the next section.

1.1 Setting up the Task Definition

In IdentityIQ, go to the task configuration and select a new task, and select “Generic Importer” from the drop down. A New Task form will be presented with several options to complete. The Options are broken down into 4 sections:



- Generic
- JDBC
- File
- Excel

The options with the “Generic:” prefix apply to all iterators, and their options are:

Name	Description
Generic: Import Driver Class Task argument name: `genericImportDriverClass`	**Required:** The class to use to iterate through the data feed, Supported classes for the generic import can include: TextFileImport, JdbcImport and ExcelSaxImport *NB: The importer will initially search the “sailpoint.services.standard.task.genericImport” name space so any stock iterator can be referred to with their class name only.*
Generic: Group by Task argument name: `importGroupBy`	**Optional:** “Group By” allows the Controller to group the records by field name specified in CSV format, e.g: name, department *NB: The GenericImporter does not have the facility to sort the feed first, only use the Group By function if the data feed is sort in the order you wish the data to be grouped.*
Generic: MV Field Task argument name: `importMultiValueFields`	**Optional:** After the “Group By” entry is populated, the “MV Field” is a CSV format file which allows the fields to be transformed into a list of values based on the data returned by the “Group By” function.
Generic: Init Rule name Task argument name: `importInitRule`	**Optional:** A Rule which is run before the iterator is executed.
Generic: Transform Rule name Task argument name: `importTransformRule`	**Optional:** A Rule which can transform the row if the source feed cannot be transformed easily using the Generic: Transform iterators header function.
Generic: Row Rule name Task argument name: `importRowRule`	**Required:** Row Rule which is executed for each record in the iterator.
Generic: Finalize Rule name	**Optional:** A finalize Rule which is executed after the iterator has been

Task argument name: `importFinalizeRule`	exhausted.
Generic: Logger to use instead of default Task argument name: `importLoggerName`	**Optional:** Custom logger name, to use instead of the sailpoint.services.standard.task.genericImport.GenericImporterController logger.
Generic Use log level for custom logger Task argument name: `importLoggerLevel`	**Optional:** If a customer logger is specified, the logging level can be specified here. The logging level can be: ALL, DEBUG, ERROR, FATAL, INFO, OFF, TRACE and WARN.
Generic: Transform iterators header Task argument name: `importManualHeader`	**Optional:** String command to transform the schema. This is discussed further in another section.

The setup the different iterators are detailed in the next sections.

Please note that with all iterators, the values will be passed to the GenericImporter as "String" values and will ignore the data types from the source feed.

1.1.1 Delimited Text File Iterator

The Delimited Text File Iterator will iterate a CSV Text File. The entry "Generic: Import Driver Class" should be set to "TextFileImport" to use the settings below.

Name	Description
File: File name Task argument name: `text_fileName`	Filename of the text file to be iterated over.
File: Delimiter Task argument name: `text_fileDelimiter`	The row delimiter for the file. If none is specified then "," is the default.
File: File has a header Task argument name: `text_hasHeader`	Set to "true" if the import file has a header
File: Import remark token Task argument name: `text_remarkToken`	Remark token, will ignore this line if encountered
File: Encoding Task argument name: `text_fileEncoding`	Encoding use to read the file

Notes about this iterator:

- If no header is detected, all column names will be numbers, using a zero-based index

1.1.2 JDBC Iterator

The JDBC Iterator will iterate a JDBC RecordSet. The entry "Generic: Import Driver Class" should be set to "JDBCImport" to use the settings below.

Name	Description
JDBC: Driver Class Task argument name: `jdbc_driverClass`	JDBC Driver class, such as: `oracle.jdbc.OracleDriver`
JDBC: URL Task argument name: `jdbc_url`	JDBC URL for the database resource. E.g. `jdbc:oracle:thin:@ldap://database:3389/DB`
JDBC: Username Task argument name: `jdbc_user`	User name used for authentication for the JDBC resource
JDBC: Password Task argument name: `jdbc_password`	Password used for authentication for the JDBC resource, this value can be encrypted, e.g. `1:327377w734YQZExYyz==`
JDBC: SQL Query Task argument name: `jdbc_sqlQuery`	The SQL Query used to iterate through. If using groupBy generic function, ensure the result set is sorted by the fields listed in the groupBy entry.

Notes about this iterator:

- Headers will always be detected, based on the schema the JDBC query returns

1.1.3 Excel Iterator

The Excel Iterator will iterate an Excel Worksheet. The entry "Generic: Import Driver Class" should be set to "ExcelSAXImporter" to use the settings below

Name	Description
Excel: The excel full path and file name Task argument name: `excel_filename`	The Excel filename
Excel: Does the worksheet have a header row Task argument name: `excel_hasHeader`	Indicates there is a row of data in the spreadsheet which indicates the header.
Excel: Sheet name Task argument name: `excel_sheetName`	The Excel worksheet to iterate through
Excel: Header row number Task argument name: `excel_headerRow`	The line number used in Excel for the header.

Notes about this iterator:

- If no header is detected, all column names will be numbers, using a zero-based index
- The Excel spreadsheet must be in MS Office Open XML file format (OOXML), which is usually represented as an ".xlsx" file.

1.2 Rapid Schema Transformation

The Rapid Schema Transformation is passed to the Generic Importer via the Task Definition. The transform is configured in the "**Generic: Transform iterators header**" (*Task argument name: importManualHeader*) and has a command syntax which can augment the way the Data Row Map is presented.

This value is a CSV format value but each CSV value can be interpreted with different meanings.

The basic syntax is for each item in a CSV list. For each item you can specify:

```
Column1
```

Optionally transform the column, to coerce the type:

```
Column1(Integer)
```

Column1 string value will now be transformed as an Integer in the Data Row Map.

Optionally transform column to give a new column name:

```
Column1=NewColumn1
```

Column1 will now be known as NewColumn1 in the Data Row Map.

Optionally transform column to give a new column name and coerce the data type:

```
Column1=NewColumn1(Integer)
```

Column1 will now be known as NewColumn1 and the string value transformed to an Integer in the Data Row Map

A data type conversion may be more complex so additional information could be required to do the transform, such as the Date type. The Date transformation needs the date format the value is in before returning a Java Date object, this is passed as parameters in brackets after the data type is designated:

```
Column1=NewColumn1(Date(dd-mmm-yyyy))
```

Column1 will now be known as NewColumn1 and the string value transformed to a Java Date based on the date format "dd-mmm-yyyy" in the Data Row Map.

A data type conversion is not limited to simple data types; with the Rapid Schema Transform values can be transformed into any SailPoint IdentityIQ first class object which has a setName(String), getName(String) method:

```
Column1=NewColumn1(Identity)
```

- *Column1 will be renamed as NewColumn1*
- *NewColumn1 will be transformed to the Identity type, and the value of NewColumn1 will be used to lookup an Identity object of that name.*

- *If the Identity object exists, the NewColumn attribute will be an Identity object of that name*
- *If the Identity Object does not exist, the NewColumn attribute will be NULL.*

The SailPoint IdentityIQ first class object transform has some additional modifiers which can be used before specifying the object type:

Modifier	Description
+	If an object does not exist, then create the object and return that in the Data Row Map
++	If an object does not exist, then create it AND persist it to the IdentityIQ Database, then return the Identity in the Data Row Map.

`Column1=NewColumn1(+Identity)`

- *Column1 will be renamed as NewColumn1*
- *NewColumn1 will be transformed to the Identity type, and the value of NewColumn1 will be used to lookup an Identity object of that name.*
- *If the Identity object exists, the NewColumn attribute will be an Identity object of that name*
- *If the Identity Object does not exist, a new Identity object will be created and the Identity object name will be set to the value held in the NewColumn string.*

`Column1=NewColumn1(++Identity)`

- *Column1 will be renamed as NewColumn1*
- *NewColumn1 will be transformed to the Identity type, and the value of NewColumn1 will be used to lookup an Identity object of that name.*
- *If the Identity object exists, the NewColumn attribute will be an Identity object of that name*
- *If the Identity Object does not exist:*
    - *a new Identity object will be created and the Identity object name will be set to the value held in the NewColumn string.*
    - *The new Identity Object will be saved back in the IdentityIQ data repository and the transaction will be committed to make it permanent.*

1.3 Rules

The Generic Importer has four separate rules which can be run, The Init Rule at the start of the import, Transform Rule and Row Rule while the data is being iterated over and then finally the Finalize Rule. The diagram below shows the rules and order they are processed.

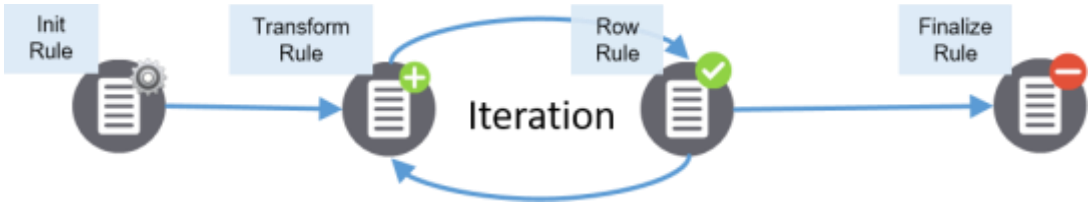


Figure 2: Rules and order they are processed

1.3.1 Init Rule

The “Init” Rule runs at the start of the import, and only runs once. This Rule is optional, and not specifying this rule will still allow the import to execute.

Variables instantiated in the rule:

Variable name	Description
log	Logger for this BeanShell Rule, logger class
context	SailPoint context
taskResult	sailpoint.object.TaskResult object, from the Task Executor
taskAttributes	sailpoint.object.Attributes object, from the Task Executor

Returns nothing.

1.3.2 Transform rule details

The “Transform” Rule is executed for each record being iterated over. This Rule is optional, and not specifying this rule will still allow the import to execute.

Variables instantiated in the rule:

Variable name	Description
log	Logger for this BeanShell Rule, logger class
context	SailPoint context
taskResult	sailpoint.object.TaskResult object, from the Task Executor.
taskAttributes	sailpoint.object.Attributes object, from the Task Executor
row	A row of data from the input source, represented as a sailpoint.object.Attributes object
transform	A sailpoint.object.Attributes object used to hold the transformed row

Returns:

Variable name	Description
sailpoint.object.Attributes	Returns the transformed row of data to be processed by the Row Rule.

1.3.3 Process Row rule details

The “Row” Rule is executed after the Transform Rule, for each record being iterated over. This Rule is required.

Variables instantiated in the rule:

Variable name	Description
log	Logger for this BeanShell Rule, logger class
context	SailPoint context
taskResult	sailpoint.object.TaskResult object, from the Task Executor.
taskAttributes	sailpoint.object.Attributes object, from the Task Executor
row	A row of data from the input source, represented as a sailpoint.object.Attributes object
transform	A sailpoint.object.Attributes object used to hold the transformed row

Returns Nothing.

1.3.4 Finalize rule details

The “Finalize” Rule runs at the end of the import and only runs one. This Rule is optional, and not specifying this rule will still allow the import to execute.

Variables instantiated in the rule:

Variable name	Description
log	Logger for this BeanShell Rule, logger class
context	SailPoint context
taskResult	sailpoint.object.TaskResult object, from the Task Executor.
taskAttributes	sailpoint.object.Attributes object, from the Task Executor

Returns nothing.

1.4 Transmogrifier Tool

The Transmogrifier tool is a Java class that allows the creation, population or retrieval and update of a SailPoint Object class by merging the Row Data Map to the object.

The Transmogrifier should use the following imports:

```
import sailpoint.services.standard.task.genericImport.Transmogrifier;
```

Details of constructors and useful methods are given below. For full details please see the Transmogrifier Javadoc in <SSD root>/doc/JavaDoc.

1.4.1 Constructors

The Transmogrifer has a number of constructors to help simply implementation, they are:

- (SailPointContext)
- (SailPointContext , Attributes (Row meta data attribute map),)
- (SailPointContext, Attributes (Row meta data attribute map), SailPointObject)

The SailPointContext is the normal “context” variable you will see in all IdentityIQ Rules. You may omit the SailPointContext and the class will attempt to discover the current context.

The Attributes map defined as “row” in the Row Rule where the Transmogrifier is expected to be instantiated.

The SailPointObject is the first class IdentityIQ object to merge Row data with. Please note that the SailPointObject does not need to be instantiated.

1.4.2 mergeObjectWithRow ()

Method will attempt to merge the SailPointObject with the row data map.

If the SailPointObject is null, then using the map it will attempt to discover an object’s name. If one is found, then it will instantiate the object from the IdentityIQ repository instead of creating a new object.

The SailPointObject custom attributes will be queried against the Row Data Map column name, if a match is found then the SailPointObject’s custom attributes will be populated with the value in the Row Data Map.

The SailPointObject setter methods will be queried against to try to match the Row Data Map column names; these names are also transformed to be prefixed with “set” and the first letter of the column name is upper cased, e.g.

Column-name in the Row Meta Data Map is set to “name”

The mergeObjectWithRow method will perform the following

- Attempt to discover a method called “setName” on the Object being operated on.

- If the method is found, then the Row Meta Data map's value is set as a parameter for that method and then executed.

- If the "setName" method is not found on the Object being operated on, then it will attempt to find a method called "name".
  - If the method is found, then the Row Meta Data map's value is set as a parameter for that method and then executed.

1.4.3 setRemovePrefix(prefix)

This method allows any prefixes to be removed from the column list before it's executed. The remove prefix is a csv value so can remove multiple prefixes in one call

E.g.

```
Column name = identity_name

setRemovePrefix("identity_");
```

Column name will be rendered as "name" in the Transmogrifier.

1.4.4 setIncludeAttributes

Filter the attributes based on an include specification held in a CSV. The set include attribute spec is a CSV which is in the following format:

```
columname1,columnname2,columnname3,attributename,location
```

The csv value can also include wild cards of type "*" and "?" similar to filing system wildcards. The following will give you the same result of the filter spec above

```
column*,attributename,location
```

and

```
columnname?,attribute*,location
```

1.4.5 setExcludeAttributes

Filter the attributes based on an include specification held in a CSV. The set exclude attribute spec is a CSV which is in the following format:

```
columname1,columnname2,columnname3,attributename,location
```

The csv value can also include wild cards of type "*" and "?" similar to filing system wildcards. The following will give you the same result of the filter spec above

```
column*,attributename,location
```

and

```
columnname?,attribute*,location
```

1.4.6 setForceInProvisioningPlan()

When calling the createObjectProvisoningPlan or the createAccountProvisioningPlan the method will filter out row columns based on the application schema. A CSV list of attribute names can be set with this method to ignore the application schema check and force a row column into the provisioning plan. This can be useful when attributes are not exposed in the Application schema but are valid in the target system.

1.4.7 createAccountProvisoningPlan / createObjectProvisoningPlan

These methods required the following augments:

- IdentityIQ Application Name
- Schema name of the object to be managed
- Operation (sailpoint.object.ProvisioningPlan.ObjectOperation)
- Target system's native identity

The Row Data will be examined and a provisioning plan will be created based on the arguments supplied.

Please not that included and excluded attributes will be honoured similar to the mergeObjectWithRow method

If an attribute is to be forced into the plan from the Row map, the setForceInProvisioningPlan value will be honoured.

Example for group management in an LDAP directory.

The following row data is assumed:

```
Cn, objectClass, dn, description, uniqueMember, illegalAttribute <CR>
iiqAdmins2,groupOfUniqueNames,cn=iiqAdmins2,ou=groups,ou=system,IIQ Admin group
, "cn=spadmin,dc=sailpoint,dc=com, "Should not see this attribute in the plan") <CR>
```

Code to merge this row to create an LDAP Group object in a theoretical application called LDAP

```
Transmogrifier t = new Transmogrifier(context, row);
ProvisioningPlan plan = t.createObjectProvisioningPlan("LDAP", "group",
      ProvisioningPlan.ObjectOperation.Create, (String) row.get("dn"));
```

Example for account management in an LDAP directory

The following row data is assumed:

```
cn, sn, objectClass, dn, mail, description, illegalAttribute <CR>
spadmin, Administrator, {inetOrgPerson, organizationalPerson, person, top},
cn=spadmin,ou=users,ou=system, spadmin@sailpoint.com, This is spadmin's ldap account, Should
not see this attribute in the plan

// This is an additional column, which can be added programmatically
// to the original feed
row.put("password", "Letmein99");
```

The following code transforms the above row data into an object in the target system

```
Transmogrifier t = new Transmogrifier(context, row);
// The LDAP application does not have password in the schema, so we can force
// it into the provisioning plan with the next statement
t.setForceInProvisioningPlan("password");

ProvisioningPlan plan = t.createAccountProvisioningPlan("LDAP", "account",
      ProvisioningPlan.ObjectOperation.Create,
      (String) row.get("dn"));
```

1.4.8 ExecuteProvisoningPlan

The ExecuteProvisioningPlan method will execute the created plan against the target system.

2 Examples

This section discusses the samples which are available with the SSD Distribution.

2.1 Template Rules

The Generic Importer does not have a registered Rule type in IdentityIQ, so to aid development 4 template rules are available; the inputs and their types are described as well as IdentityIQ Deployment Accelerator mark-ups to help inside the Eclipse IDE.

1. Template_Init.xml The Init Rule template
2. Template_Transform.xml The Transform Row Rule template
3. Template_Row.xml The Row Rule template
4. Template_Finalise.xml The Finalise Rule template