SSF Tools: IdentityIQ LogiPlex Connector User Guide

LogiPlex Connector: IdentityIQ 7.0, 7.1+

This document explains the inner workings, installation and use of the LogiPlex Connector. The LogiPlex Connector is a special connector that provides functionality like the standard Logical connector and multiplex applications.

Document Revision History

Revision Date	Written/Edited By	Comments
April 2018	Menno Pieters	Initial Release
March 2019	Menno Pieters	Adapter mode added

Table of Contents

LogiPlex Connector Overview

This document explains what the LogiPlex connector is, its inner workings, installation and usage. The LogiPlex Connector is a special connector that provides functionality which is a mixture of the standard Logical connector and multiplex applications.

1. Introduction

The LogiPlex Connector is a possible alternative for logical applications, using features of the multiplex application type. The name of this connector is composed of the connector types that inspired the creation: the logical and the multiplex connector types.

From the *Direct Connectors Administration and Configuration Guide*:

> *The SailPoint Logical Connector is a read only connector developed to create objects that function like applications, but that are actually formed based on the detection of accounts from other, or tier, applications in existing identity cubes.*
>
> *For example, you might have one logical application that represents three other accounts on tier applications, an Oracle database, an LDAP authorization application, and a custom application for internal authentication. The logical application scans identities and creates an account on the logical application each time it detects the three required accounts on a single identity.*
>
> *You can then use the single, representative account instead of the three separate accounts from which it is comprised for certification, reporting, and monitoring.*

The Logical Connector has a number of performance drawbacks: it requires information to be aggregated and then iterates over identities to find suitable matches. This is often a very time-consuming task. The LogiPlex Connector tries to overcome these issues by processing logical applications on the fly, during aggregation. This approach allows the use of performance-enhancing connector features like partitioning, optimized aggregation and delta aggregation.

The Logical Connector acts as a wrapper around any normal connector. It can operate in two modes:

1. **Classic Mode:** It must be set up to point to a pre-configured application and will use all features of the target application for aggregation and provisioning. It uses two additional rules to process information when aggregating and provisioning account or group information.
2. **Adapter Mode:** an existing, pre-configured application is modified to use the LogiPlex adapter to wrap the normally used adapter. The same additional rules that are used in classic mode are still used to control the LogiPlex-behavior.

When aggregating, account and group information is enriched with additional attributes to indicate the name of the logical sub-application. This mechanism is the same as used for so called multiplex applications.

A possible downside of the LogiPlex Connector is that it may in some cases be a little harder to set up than a standard Logical Connector. The LogiPlex Connector requires more BeanShell coding to work than most Logical Connector configurations. For the Logical Connector the tiers can be configured by just selecting the relevant entitlements. Furthermore, the LogiPlex Connector is less suitable for combining information from multiple tiers, although the aggregation split rule can be used to perform side lookups.

2. Application Details

2.1. Application Type

The application type is “LogiPlex Connector” in classic mode or any other standard or custom application type in adapter mode.

3. Terminology

In this document, we will use three different terms for the application definitions involved in the aggregation and provisioning of LogiPlex data.

3.1. Master Application

The master application is a standard application, like a directory server (LDAP, Active Directory), a database or delimited file application. This application needs to be set up first and proven to be working correctly.

3.2. Main Application

The main application is the base LogiPlex application. The *Main* Application will be configured to point to the *Master* Application and use the connector defined in the *Master* Application to perform the actual work.

3.3. Sub Application

A Sub Application is any application derived from the *Main* Application, based on information read from accounts. Unless the aggregation task is configured to prevent this, Sub Applications will be generated on the fly, while aggregating data.

A Sub Application will have the Main Application defined as a Proxy.

3.4. Inner Working

The idea behind this connector is that we would like to use multiplex-like behavior to achieve the logical account grouping that a Logical Connector provides: sub-applications of a master application. As an example, consider two sets of groups in a directory server giving access to a web application and a server application.

A multiplex application only allows an account to be redirected to a single sub-application, while we would like the same account to be related to the web application as well as the server application, listing only the relevant group memberships.

The LogiPlex Connector features a rule option that allows inspection of a freshly read account or group object (ResourceObject) and returns one or more slightly modified instances of this object in a map.

The LogiPlex Connector uses a special iterator class that wraps the iterator of the original connector but uses the split rule to retrieve the sub-application accounts.

3.4.1. Classic Mode

In the classic mode, the master application and main application are two different application definitions. The master application is set up and must be confirmed working. Then a second definition is created of the type “LogiPlex Application” which is pointed to the pre-existing master application.

Parts of the master application’s XML definition, like the provisioning policies, need to be copied over to the main application. The basics and rules of the LogiPlex main application can be configured in the user interface.

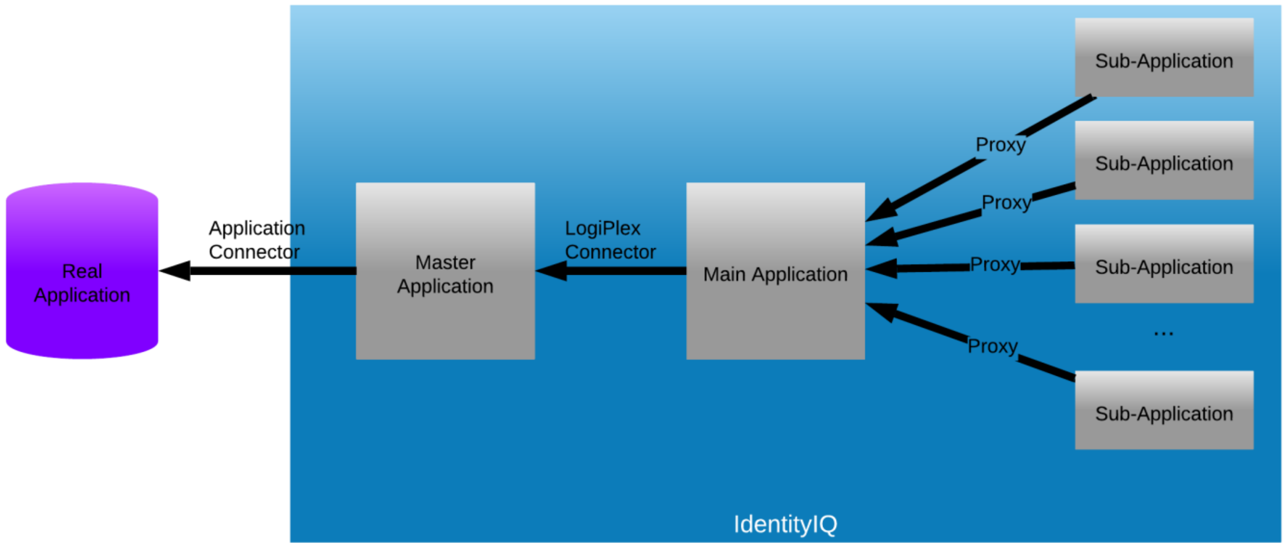


Figure 1: Relations between application definitions and connectors in “classic mode”

3.4.2. Adapter Mode

In the adapter mode, the master and main application become effectively one application definition. The master application is set up and must be confirmed working. Once that is done, a few changes need to be made to the application definition's XML:

- The real connector class is replaced by the class of the LogiPlex connector,
- The real connector class name is placed in a special attribute used by the LogiPlex connector,
- The aggregation (split) rule must be specified in the attributes map,
- Optionally a provisioning (merge) rule can be specified.

Adapter mode requires more manual changes to the XML definition but allows the master application to be updated with aggregation statistics and other information necessary for subsequent aggregations, especially when delta aggregation is used.

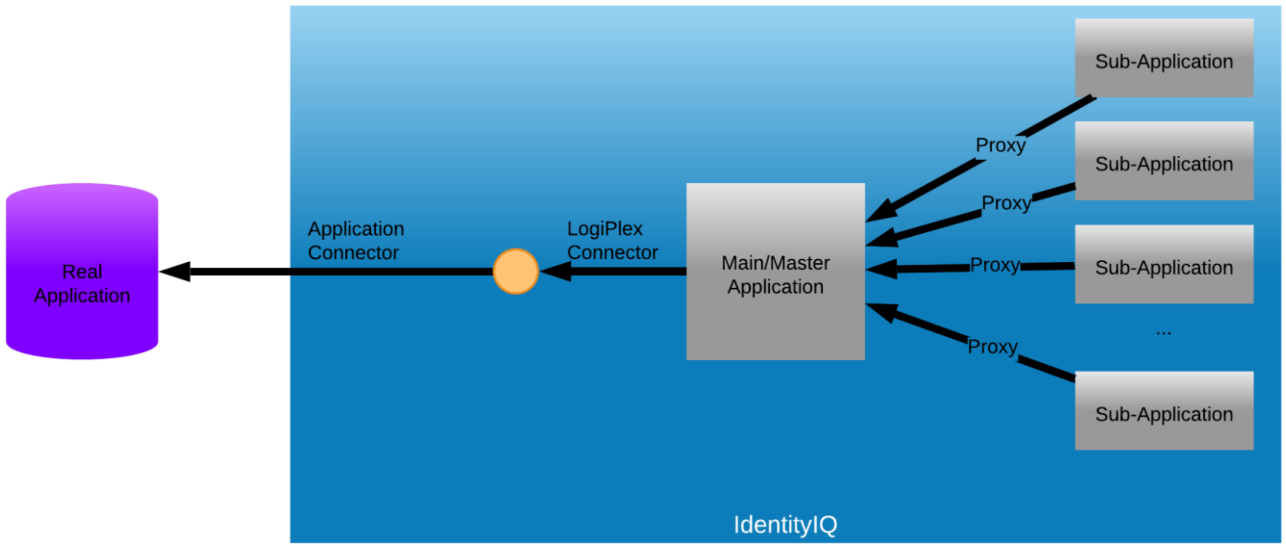


Figure 2: Relations between application definitions and connectors in "adapter mode"

Installation and Configuration – Classic Mode

4. Implementation Overview

The implementation of the LogiPlex connector in classic mode require the following steps:

- Deployment of the connector and supporting files and configurations,
- Set up the master application and confirm connectivity,
    - Test the configuration and schema
- Define and configure a LogiPlex application:
    - Set the master application,
    - Optionally define a prefix for auto-generated sub-applications,
    - Define a split rule to determine which entitlements belong to which sub-application
    - Optionally define a merge rule
    - For provisioning: configure a provisioning on the main application
    - For generated sub-applications:
        - Manually copy configuration options,
        - Manually copy and update the provisioning policy.

5. Installation Files

If you are deploying IdentityIQ 7.2 or greater with the Services Standard Deployment (SSD), the LogiPlex Connector will be deployed by default, and there are no extra steps to take to install it. Deployment is controlled by this property in the `build.properties` file:

```
deployLogiPlexConnector=true
```

Setting it to false will prevent the connector being deployed.

The IdentityIQ LogiPlex Connector consists of the following Java class files:

Filename	Description
LogiPlexConnector.java	Main LogiPlex Connector Java class

The configuration files are:

Filename	Description
ConnectorRegistry-LogiPlexConnector.xml	Connector registry merge file to describe the connector.

The configuration interface files, placed under `define/applications/` are:

Filename	Description
logiPlexAttributes.xhtml	Base configuration
logiPlexAttributesInclude.xhtml	Additional attribute configuration

logiPlexRulesForm.xhtml	Connector Rules configuration

Note that the default version of logiPlexAttributes.xhtml will work with IdentityIQ 8.0 and later, but not with earlier versions. For IdentityIQ 7.x another version of the file is supplied, called logiPlexAttributes7x.xhtml; you will need to remove logiPlexAttributes.xhtml and rename logiPlexAttributes7x.xhtml to replace it. If you are deploying with SSD v6.1 or later, this will automatically be done for you.

Some sample rules are also provided in the SSD under the folder `config/SSF_Tools/LogiPlex_Connector/Samples`. These will not be deployed by default.

6. Configuration

6.1. Settings

The connector has just two configuration settings:

Setting	Type	Description
Master Application	Application	The application definition which defines the connector of the real target application
Split Application Prefix	String	If set, every application selected by the split rule on aggregation will be prefixed with this string. If the string does not end in a dash, a dash will be added as a separator. E.g. if the prefix is set to “LGX”, the sub-application “Expenses” will become “LGX-Expenses”. If the prefix is set to “ABC-”, it will become “ABC-Expenses”

Connection settings must be configured on the master application.

6.2. Rules

The connector has two additional rules options. Since we cannot extend the types of rules supported by IdentityIQ, as rule types are defined as an enum, we have to re-use existing rule types.

6.2.1. LogiPlex Split Rule

6.2.1.1. Description

The LogiPlex Split Rule is in fact a rule of type ResourceObjectCustomization, but is run before the actual customization rule is applied. It will receive slightly different inputs and the expected output is also different.

6.2.1.2. Definition and Storage Location

The rule is associated to a LogiPlex application in the UI in the application definition:

Application →Application Definition → select existing or create new application → Rules → LogiPlex Split Rule

6.2.1.3. Arguments

Inputs

Argument	Type	Purpose
object	sailpoint.object.ResourceObject	A reference to the resource object built by the connector
application	sailpoint.object.Application	
applicationName	java.lang.String	The name of the application.
connector	*Not provided*	-
state	java.util.HashMap	A Map that can be used to store and share data between executions of this rule during a single aggregation run.
map	java.util.HashMap <String,List<ResourceObject>>	A pre-prepared map to be filled and returned
util	sailpoint.services.standard.connector. LogiPlexConnector.LogiPlexUtil	An instance of the LogiPlexUtil as used by the iterator, to allow calling convenience methods (see JavaDoc for the connector).

Note: contrary to a normal customization rule, the connector variable is not set.

Outputs

Argument	Type	Purpose
objects	java.util.HashMap	A Map containing application names as keys and corresponding ResourceObject values.

6.2.1.4. Example

This example rule will inspect the account and group objects received from an LDAP server. It will check the name of group memberships. Groups starting with `cn=webapplication` will be related to the **WebApplication** application. Groups starting with `cn=expenses-` will be related to the **Expenses** application. All other groups will be related to the main application. In this example, the choice has been made to also return results for the main application, although this is not required.

```
import sailpoint.tools.Util;
import sailpoint.object.ResourceObject;

public String resolveApplication(String name) {
  if (Util.isNotNullOrEmpty(name)) {
    String lname = name.toLowerCase();
    if (lname.startsWith("cn=webapplication")) {
      return "WebApplication";
    }
    if (lname.startsWith("cn=expenses-")) {
      return "Expenses";
    }
  }
  return application.getName();
}

String applicationName = application.getName();
Map map = new HashMap();

if ("account".equals(object.getObjectType())) {
  List groups = object.getStringList("groups");
  if (groups != null && !groups.isEmpty()) {
    Map groupMap = new HashMap();
    groupMap = util.updateListMap(groupMap, applicationName, null);
    for (String group: groups) {
      String appName = resolveApplication(group);
      if (Util.isNotNullOrEmpty(appName)) {
        groupMap = util.updateListMap(groupMap, appName, group);
      }
    }
    Set keys = groupMap.keySet();
    if (!keys.isEmpty()) {
      for (String key: keys) {
        List appGroups = groupMap.get(key);
        ResourceObject cloneObject = object.deepCopy(context);
        if (!Util.isEmpty(appGroups)) {
          cloneObject.put("groups", appGroups);
        } else {
          cloneObject.remove("groups");
        }
        map.put(key, cloneObject);
      }
    } else {
      map.put(applicationName, object);
    }
  } else {
    map.put(applicationName, object);
  }
} else if ("group".equals(object.getObjectType())) {
  String nativeIdentity = object.getIdentity();
  String appName = resolveApplication(nativeIdentity);
  map.put(appName, object);
} else {
  map.put(applicationName, object);
}

return map;
```

In case a large number of groups need to be considered, good alternatives to check for the destination of groups could be:

- A Custom object containing the names or regular expressions to match,
- Additional information on groups in the source application: aggregate groups first to populate the Entitlement Catalog, then reference the entitlement catalog to determine to which application(s) a group should be linked.

6.2.2. LogiPlex Provisioning Rule

6.2.2.1. Description

The LogiPlex Provisioning Rule is in fact a rule of type CompositeRemediation (Logical Provisioning Rule). It accepts a provisioning plan and can modify the plan for the real target application and then return the modified provisioning plan.

If the rule is not provided, the connector will apply internal default logic. This default logic will:

- Clone the plan and included AccountRequest and ObjectRequest objects.
- For each request object targeted at the main application, just keep the request as is.
- For each account request object targeted at a sub application:
  - Handle create operation as create operations only if there is no corresponding account on the main application,
  - Handle create operations as modify operations, applicable only to entitlements, if there already is a corresponding account on the main application,
  - Handle deletion on a sub-application as a removal for all entitlements,
  - Handle changes on sub-application accounts only for entitlements,
  - Handle enabling, disabling, locking and unlocking only on the main application (ignore for sub-applications).

NOTE: When deleting, enabling, disabling, locking or unlocking the main application account, the same operation is applied to sub-application. This is handled *after* the provisioning plan has been executed and applied internally only.

NOTE: Most often, the rule will clone the original plan and modify the included AccountRequest objects. Note that for these objects and the enclosed AttributeRequest objects, it is important to also copy any arguments, as these may be necessary to link information back to its origins after provisioning.

6.2.2.2. Definition and Storage Location

The rule is associated to a LogiPlex application in the UI in the application definition:

> Application →Application Definition → select existing or create new application → Rules → LogiPlex Provisioning Rule

6.2.2.3. Arguments

Inputs

Argument	**Type**	**Purpose**
identity	sailpoint.object.Identity	Reference to the Identity object for whom the provisioning request has been made
plan	sailpoint.object.ProvisioningPlan	Reference to a provisioning plan against the LogiPlex application
application	sailpoint.object.Application	The LogiPlex application definition

Argument	Type	Purpose
masterApplication	sailpoint.object.Application	The master application definition
connector	sailpoint.connector.AbstractConnector	An instance of the LogiPlex connector
masterConnector	sailpoint.connector.AbstractConnector	An instance of the connector for the master application

Outputs

Argument	Type	Purpose
plan	sailpoint.object.ProvisioningPlan	The modified plan

6.2.2.4. Example

The goal of this rule is to merge changes for sub-applications into the main application. It is important to understand that the plan will, after successful provisioning, also be applied to the identity cube. So, the plan must be modified in such a way that whatever will change ends up in the original plan object.

The easiest implementation is by calling the default logic on the main application, which is demonstrated below. After doing that, it is possible to make some additional changes to the plan.

```
import sailpoint.object.ProvisioningPlan;
import sailpoint.services.standard.connector.LogiPlexConnector;

// Expect: sailpoint.connector.AbstractConnector connector

// Use the default logic from the connector to re-compose the plan.
ProvisioningPlan newPlan = ((LogiPlexConnector)
connector).runDefaultProvisioningMergeLogic(plan, identity);

// After using the default logic, optionally, other changes could be applied.

if (log.isTraceEnabled() && newPlan != null) {
  // Dump the plan.
  log.trace(newPlan.toXml());
}
return newPlan;
```

7. Schema Attributes

The schema(s) can either be generated from the master application or entered manually. To generate the schema(s) from the master application, use the **Discover Schema Attributes** for each object type.

8. Provisioning Policies

Provisioning policies will, unfortunately, not be copied from the master application to the LogiPlex application. The provisioning policy or policies will have to be re-configured or copied as XML in the Debug interface.

If the provisioning policy for the master application is setup as a separate Form object, both the master and the LogiPlex application can reference the same Form object.

The field definition for the “identity attribute” and likely also the “display attribute” on the provisioning policy for sub-applications, may need to be adjusted such that it will not generate a new value, but look for an existing account on the main application. If an account on the main application is found, the values for the “identity attribute” and “display attribute” should be copied from the main application account.

Installation and Configuration – Adapter Mode

9. Implementation Overview

The implementation of the LogiPlex connector in adapter mode require the following steps:

- Deployment of the connector,
- Set up the master application and confirm connectivity,
  - Test the configuration and schema,
  - Define and test provisioning.
- Modify the master application XML to become a LogiPlex application:
  - Replace the connector class with the LogiPlex connector class,
  - Set the original connector class in the attribute XXXX.
  - Optionally define a prefix for auto-generated sub-applications,
  - Define a split rule to determine which entitlements belong to which sub-application
  - Optionally define a merge rule
  - For generated sub-applications:
    - Manually copy configuration options,
    - Manually copy and update the provisioning policy.

10. Installation Files

If you are deploying IdentityIQ 7.2 or greater with the Services Standard Deployment (SSD), the LogiPlex Connector will be deployed by default, and there are no extra steps to take to install it. Deployment is controlled by this property in the build.properties file:

```
deployLogiPlexConnector=true
```

Setting it to false will prevent the connector being deployed.

The IdentityIQ LogiPlex Connector consists of the following Java class files:

Filename	Description
LogiPlexConnector.java	Main LogiPlex Connector Java class

Other files (see chapter 5) will also be deployed as part of an SSD deployment, but are not needed when adapter mode is used.

Some sample rules are also provided in the SSD under the folder `config/SSF_Tools/LogiPlex_Connector/Samples`. These will not be deployed by default.

11. Configuration

11.1. Settings

In adapter mode, all settings have to be changed in the XML directly, through the debug interface, an exported XML document, or preferably in the IdentityIQ Deployment Accelerator.

11.1.1. Connector Class

The most important change is the connector class. The following changes are needed:

- In the <Application> tag, change the connector class to the LogiPlex connector class.
- Put the original connector class in the `logiPlexMasterConnector` attribute.

Original (example: Active Directory):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Application PUBLIC "sailpoint.dtd" "sailpoint.dtd">
<Application authoritative="false" connector="sailpoint.connector.ADLDAPConnector"
featuresString="PROVISIONING, SYNC_PROVISIONING, AUTHENTICATE, MANAGER_LOOKUP,
SEARCH, UNLOCK, ENABLE, PASSWORD, CURRENT_PASSWORD" icon="directory1Icon"
name="Active Directory" profileClass="" type="Active Directory - Direct">
  <Attributes>
…
```

Updated:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Application PUBLIC "sailpoint.dtd" "sailpoint.dtd">
<Application authoritative="false"
connector="sailpoint.services.standard.connector.LogiPlexConnector"
featuresString="PROVISIONING, SYNC_PROVISIONING, AUTHENTICATE, MANAGER_LOOKUP,
SEARCH, UNLOCK, ENABLE, PASSWORD, CURRENT_PASSWORD" icon="directory1Icon"
name="Active Directory" profileClass="" type="Active Directory - Direct">
  <Attributes>
…
      <entry key="logiPlexMasterConnector"
value="sailpoint.connector.ADLDAPConnector"/>
…
```

11.1.2. LogiPlex Connector Settings

The following settings should be edited in the XML definition. All of these go into the Attribute section of the application definition.

Attribute	Type	Required	Description
logiPlexMasterConnector	String	Yes	The full class name of the original connector to be used by the LogiPlex connector.

Attribute	Type	Required	Description
logiPlexPrefix	String	No	If set, every application selected by the split rule on aggregation will be prefixed with this string. If the string does not end in a dash, a dash will be added as a separator. E.g. if the prefix is set to “LGX”, the sub-application “Expenses” will become “LGX-Expenses”. If the prefix is set to “ABC-”, it will become “ABC-Expenses”
logiPlexAggregationRule	String	Yes	The name of the rule that will be used during aggregation (the “split rule”) to determine to which sub-application entitlements belong.
logiPlexProvisioningRule	String	No	Optional: the name of the rule that will be used to modify the provisioning plan (the “merge rule”). If no rule is configured, the default behavior will be used.

11.2. Rules

The rules in the adapter mode are the same as used in classic mode. See §6.2.

12. Schema Attributes

The schema(s) should be setup before changing the application into a LogiPlex application. After that, the application schemas can still be modified as usual. If the master application supports discovering the schema, the **Discover Schema Attributes** button can be used for each object type. The request will simply be forwarded to the master connector.

13. Provisioning Policies

Provisioning policies will, unfortunately, not be copied from the master application to the LogiPlex application. The provisioning policy or policies will have to be re-configured or copied as XML in the Debug interface.

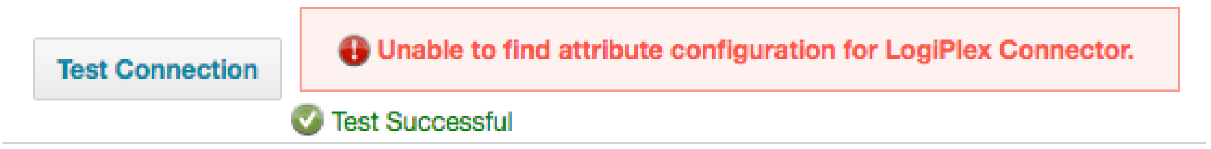
If the provisioning policy for the master application is setup as a separate Form object, both the master and the LogiPlex application can reference the same Form object.

Miscellaneous

14. Known Issues

14.1. Test Connection

When clicking the Test Connection button in “classic mode”, an error will appear along with a success message.



Unable to find attribute configuration for LogiPlex Connector.

Test Successful

Figure 3: Error during Test Connection

This specific error message can be ignored. The connector works just fine. Other errors may indicate an issue with the setup or the configuration of the *Master* application.

14.2. Aggregation of Sub-Applications

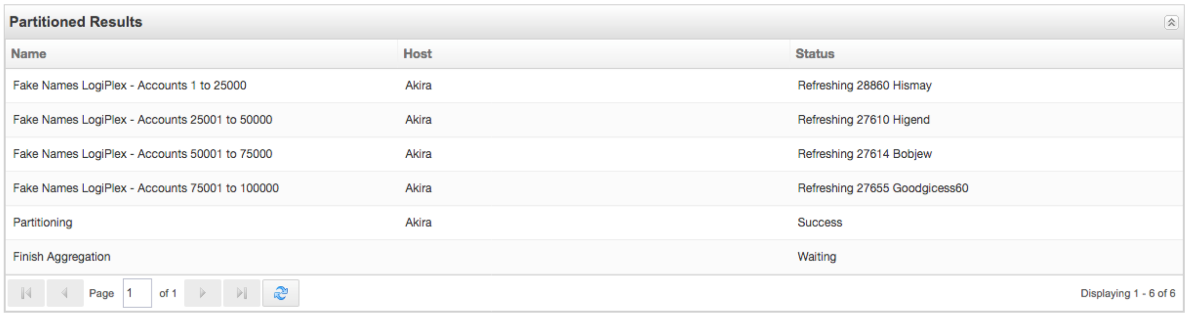
Like with a multiplex application, aggregation should be performed only on the main application. Aggregating directly from a sub-application may not have the desired outcome, depending on how rules are setup.

14.3. Partitioning

Dividing partitions only works correctly if the connector natively supports partitioning. Application level partitioning will work, but the numbers used to divide the partitions will be off. When calculating the number of partitions, the master application will return the raw number of accounts. When the split rule is applied, the number of accounts may be a lot higher.

As shown in the picture below, the number of aggregated accounts is already higher than the expected number of accounts per partition. In this example, we are aggregating a delimited file with 100,000 lines, corresponding to 100,000 accounts.

If partitioning is configured to create 4 partitions, like in this example, it expects to aggregate 25,000 accounts per partition. The LogiPlex Split Rule may return multiple accounts (main and sub) for each line read.



Partitioned Results

Name	Host	Status
Fake Names LogiPlex - Accounts 1 to 25000	Akira	Refreshing 28860 Hismay
Fake Names LogiPlex - Accounts 25001 to 50000	Akira	Refreshing 27610 Higend
Fake Names LogiPlex - Accounts 50001 to 75000	Akira	Refreshing 27614 Bobjew
Fake Names LogiPlex - Accounts 75001 to 100000	Akira	Refreshing 27655 Goodgicess60
Partitioning	Akira	Success
Finish Aggregation		Waiting

Figure 4: Results for Application Level Partitioning

14.4. Delta Aggregation

Depending on the strategy for delta aggregation used by the connector, it may need to store information on the application definition. For example, for a directory server (LDAP) where a changelog container is used (`cn=changelog`), the connector will store the latest processed change number for accounts and groups to the application definition, so it knows where to pick up changes on the next delta aggregation.

For delta aggregation, the application definition for the master connector must be updated to keep that information, which will not succeed in “classic mode” due to the fact that the application that the “Aggregator” uses is the LogiPlex main application.

Customers who require delta aggregation with LDAP, Active Directory and other application types that need delta information to be stored in the application definition, should switch to “adapter mode”. In that case, there will only be one application definition involved, which is correctly updated after each aggregation with delta information.

14.5. Version 7.1 versus 7.2+

There is minor difference between version IdentityIQ 7.1 and 7.2 that requires a small change in the connector code when moving between the versions. The class `ExpiredPasswordException` has been moved from the package `sailpoint.api` to `sailpoint.connector` in 7.2 and up.

So, IdentityIQ 7.1 needs this import:

```
import sailpoint.api.ExpiredPasswordException;
```

And IdentityIQ 7.2 and higher versions needs this import:

```
import sailpoint.connector.ExpiredPasswordException;
```

If you are using the SSD to create your build, the source code modification will be done by the build scripts and you will not need to change anything.