Birthright Provisioning

IdentityIQ Versions: 6.0, 6.1, 6.2

This document describes how to automate provisioning of accounts or entitlements for any new Identity created from an authoritative aggregation. This is often called "birthright provisioning" because these entitlements are granted to everyone simply on the basis of having an Identity in the system. In some cases, birthright entitlements may vary based on attributes recorded on the Identity or the authoritative application account.

Document Revision History

Revision Date	Written/Edited By	Comments
Dec 2012	Jennifer Mitchell	Initial Creation; current IdentityIQ version: 6.0p2 (though these options are not specific to 6.0+)
May 2013	Jennifer Mitchell	Corrected problem causing locking contention issue with example workflow
August 2013	Jennifer Mitchell	Updated to reflect compatibility with version 6.1; no changes to content
Feb 2014	Jennifer Mitchell	Updated to version6.2; small adjustments to content for clarity, but no material changes (process is same in 6.2 as before)

Table of Contents

How to Provision Birthright Entitlements

Customers commonly want automated provisioning of specific entitlements to occur when a new Identity is created based on an authoritative aggregation. IdentityIQ supports this is multiple ways. The two most commonly implemented methods of accomplishing this are:

- Using Roles (recommended method)
- Using Joiner Lifecycle Event

There are pros and cons to each of these methods.

Method	Pros	Cons
Roles	• Easier to set up (just requires role creation with assignment rules) • Uses built-in IdentityIQ functionality to do the core work • No custom workflow development	• No option for hiring manager (or other user) to have input into which entitlements to auto-provision for individual new employees
Joiner Lifecycle Event / Workflow	• Can include presentation of custom forms that gather additional info from hiring manager (or others) to determine which entitlements to auto-provision	• Requires development of custom workflow to implement functionality that is otherwise already available in the core product (through other method) • Workflow developer must understand provisioningPlan structure and provide all inputs required to build it correctly

Birthright Provisioning Using Roles

IdentityIQ can auto-provision the creation of new accounts and the assignment of entitlements to new and existing accounts using business roles with assignment rules. This method is generally recommended because it relies on the built-in functionality of IdentityIQ to do the actual processing; it can be used as long as the set of birthright entitlements to provision is static or can be calculated based on Identity attributes.

NOTE: These steps assume the *Create* provisioning policy for the application to which accounts are being provisioned contains all the required elements for provisioning a new account. If any attributes in the provisioning policy are specified as Review Required or if any required attributes are not calculated in the policy, a form will be presented to one or more users before the automated provisioning will complete as described in step 5.

1) Define a Business Role with an assignment rule that matches based on values on the authoritative application and/or Identity attributes. Common examples for this are two business roles like:
    - "All Active Employees" - matching on status from HR = "A" and employee type = "EMP"
    - "All Active Contractors" - matching on status from HR = "A" and employee type = "CTR"

    Typically the HR status and employee types are configured as Identity Attributes that are mapped from an HR feed.

2) Add required IT roles to those business roles that represent the birthright access. For example, define an IT role with one or more entitlements on the AD application and make it a required role for the business role(s) (different IT roles can be defined for the different business roles as needed). The IT roles must have the entitlements to provision specified in profiles.

3) Aggregate the authoritative HR feed. A new employee or contractor is found, and a new Identity cube is created with a Link on the HR application.

4) Run the Identity Cube Refresh task with the **Refresh assigned, detected roles and promote additional entitlements** and **Provision assignments** options checked. This causes IdentityIQ to evaluate the business role assignment rule and automatically assign the "All Active Employees" or "All Active Contractors" role to the new Identity cube.

5) Now IdentityIQ can see that this cube should have an AD account (according to the required roles on that assigned business role), so it creates a ProvisioningPlan (with "OP" = "create") based on the attributes specified in the AD application's provisioning policy. A provisioning policy is basically an outline of the fields that need to be populated when creating or modifying an account on a system.   Because of the **Provision Assignments** option on the Identity Refresh task run in step 4, that provisioning plan is automatically processed to provision the birthright entitlements; no custom workflow writing was required.

Birthright Provisioning using Joiner Lifecycle Event

To assign birthright entitlements without using roles, specify a Joiner Lifecycle Event that runs a custom Joiner workflow. That custom workflow must manually build a ProvisioningPlan and then execute the required actions to process it.

1) Write a custom workflow which creates a provisioning plan to provision the birthright access and invokes the required methods to process the plan. An example workflow is shown in the *Example Joiner Workflow* section below.

   NOTE: To prevent locking contention with the refresh task that launches it, the workflow must be executed in the background. This is accomplished through a `wait="1"` attribute in the Start Step, which causes the workflow to pause for 1 minute. It will be restarted by the next Perform Maintenance task that runs after that one minute delay; in most installations, Perform Maintenance is set to run every 5 minutes.

2) Specify that workflow as the Business Process launched by the Joiner Lifecycle Event. That event is configured (when LCM is licensed and installed) to fire a workflow in response to an Identity Create event. Ensure that the Joiner Lifecycle Event is enabled.

3) Aggregate the authoritative HR feed. A new employee or contractor is found, and a new Identity cube is created with a Link on the HR application.

4) Run an Identity Cube Refresh task with the **Process Events** options checked. This identifies the new identity cube and causes the Joiner Lifecycle Event to fire, launching the custom joiner workflow. That workflow creates and executes the provisioning plan to create a new account on AD (or other system, as specified in that workflow).

Example Joiner Workflow

This example workflow contains a step that builds the provisioning plan manually and two steps that call workflow library methods to compile and provision the provisioning project (compileProvisioningProject and provisionProject, respectively). It is backgrounded using the `wait="1"` attribute in the Start step to prevent locking contention with the Identity Refresh task that launches it.

NOTE: One of the benefits of the lifecycle event auto-provisioning method is the option to present a custom form to the manager or other personnel to determine additional entitlements that might need to be provisioned for some new employees. This example workflow does not include the additional steps required for the form(s) and conditional provisioning; it only illustrates the provisioning of a single birthright account on AD and any birthright entitlements on that application.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Workflow PUBLIC "sailpoint.dtd" "sailpoint.dtd">
<Workflow created="1354829433334" explicitTransitions="true"
id="ff8080813b71b97d013b722089d8000b" libraries="Identity" modified="1366403638601"
name="Joiner Birthright AD" type="IdentityLifecycle">
  <Variable input="true" name="trigger">
    <Description>The IdentityTrigger</Description>
  </Variable>
  <Variable input="true" name="event">
    <Description>The IdentityChangeEvent</Description>
  </Variable>
  <Variable input="true" name="identityName">
    <Description>The name of the identity.</Description>
  </Variable>
  <Variable initializer="script:(identityDisplayName != void) ? identityDisplayName :
resolveDisplayName(identityName)" input="true" name="identityDisplayName">
    <Description>
       The displayName of the identity being updated.
       Query for this using a projection query and fall back to the name.
    </Description>
  </Variable>
  <Variable initializer="string:AccountsRequest" input="true" name="flow">
    <Description>What type of LCM flow is this</Description>
  </Variable>
  <Variable name="plan">
    <Description>The provisioning plan which is built during the workflow.
    </Description>
  </Variable>
  <Variable initializer="string:LCM" input="true" name="source">
    <Description>
       String version of sailpoint.object.Source to indicate
       where the request originated.  Defaults to LCM.
    </Description>
  </Variable>
  <Variable initializer="string:true" input="true" name="trace">
    <Description>
       Used for debugging this workflow and when set to true trace
```

```
        will be sent to stdout.
      </Description>
    </Variable>
    <Variable name="project">
      <Description>
         ProvisioningProject which is just a compiled version of the ProvisioningPlan.
      </Description>
    </Variable>
    <Description>
        Enable birthright accounts when an employee joins the company.
    </Description>
    <Step icon="Start" name="Start" posX="28" posY="10" wait="1">
      <Transition to="Create Birthright Plan"/>
    </Step>

    <Step icon="Message" name="Create Birthright Plan" posX="174" posY="9"
resultVariable="plan">
      <Description>Process the user that joined and create plan for the user. Only
create AD account for new "employees".</Description>
      <Script>
        <Source>
           import sailpoint.object.ProvisioningPlan;
           import sailpoint.object.ProvisioningPlan.AccountRequest;
           import sailpoint.object.ProvisioningPlan.AttributeRequest;
           import java.util.List;
           import java.util.ArrayList;

           ProvisioningPlan plan = new ProvisioningPlan();
           Identity identityObject = context.getObjectByName(Identity.class,
identityName);
           if (identityObject.getAttribute("status").equals("Employee")) {
               log.debug("Employee... Create plan.");
               List accreqs = new ArrayList();
               //create AD account
               AccountRequest acctReq = new AccountRequest();
               acctReq.setOperation(AccountRequest.Operation.Create);
               acctReq.setApplication("AD");
               // Set the bare minimum AD attributes to create an account
               //
               acctReq.add(new AttributeRequest("sAMAccountName",identityName));
               acctReq.add(new AttributeRequest("primaryGroupDN","cn=Domain
Users,cn=Users,dc=training,dc=sailpoint,dc=local"));
               acctReq.add(new AttributeRequest("ObjectType","User"));
               acctReq.add(new AttributeRequest("*password*","newP@$$word"));
               // Add birthright groups
               //
               acctReq.add(new
AttributeRequest("memberOf","cn=TestGroup,cn=Users,dc=training,dc=sailpoint,dc=local")
);
               //  Need this next attribute to create an enabled account in AD
               //  It seems to be required to get the provisioning to properly work.
               //
               acctReq.add(new AttributeRequest("IIQDisabled",false));
               acctReq.setNativeIdentity("cn=" + identityName +
",cn=Users,dc=training,dc=sailpoint,dc=local");
               accreqs.add(acctReq);
               plan.setAccountRequests(accreqs);
               plan.setIdentity(identityObject);
               log.debug("Plan = " + plan.toXml());
           } else {
               log.debug("Contractor... NOP.");
           }
           return plan;
        </Source>
      </Script>
      <Transition to="Compile Project"/>
    </Step>

    <Step action="call:compileProvisioningProject" name="Compile Project" posX="406"
  posY="10" resultVariable="project">
      <Arg name="identityName" value="ref:identityName"/>
      <Arg name="plan" value="ref:plan"/>
      <Arg name="requester" value="string:spadmin"/>
      <Arg name="source" value="string:UI"/>
      <Arg name="optimisticProvisioning" value="string:false"/>
      <Arg name="requireCreateTemplates" value="string:false"/>
      <Arg name="noApplicationTemplates" value="string:true"/>
      <Description>Compile the provisioning plan into a provisioning project.
                   If you need to pass in provisioner options like "noFiltering"
                   or "noRoleDeprovisioning" you must pass them as explicit
                   arguments to the call.
                   The evaluation options "requester" and "source" are commonly
                   set here.
                   You can also pass things into the Template and Field scripts by
                   defining Args in this step.</Description>
      <Transition to="Provision"/>
    </Step>

    <Step action="call:provisionProject" icon="Provision" name="Provision" posX="682"
  posY="10">
      <Arg name="background" value="string:true"/>
      <Arg name="project" value="ref:project"/>
      <Description>Provision the project.</Description>
      <Transition to="end"/>
    </Step>

    <Step icon="Stop" name="end" posX="784" posY="10"/>
  </Workflow>
```

NOTE: The logging for workflows is controlled by the InternalContext, so the debug messages included in this workflow will only be printed if the InternalContext's logging is turned up to Debug level or higher (in log4j.properties file, add entry: `log4j.logger.sailpoint.server.InternalContext=debug` and reload logging configuration).