IBM Security

Identity Governance
Tech Enablement

Skipping Steps in Workflow

Using Rules

Fabrizio Petriconi

IBM

Version Control (hidden)

Ver	Date	Author	Update Summary
1	16/04/18	David E	Direct copy of Fabrizio's module
2	14/05/18	David E	Minor update from Fabrizio

Aim of this module:

Show how Rules can be used to turn a linear workflow into a branched one

Agenda

- The Concept of Skipping Steps in a Workflow
- Examples of using Rules to Alter Flow
  - Skipping activities in a single flow
  - Branching to different flows based on logic/parameters
- Hiding redundant Menu Items

Module Outcomes

At the end of this module you should:

- Understand the concept of using rules to skip steps in a workflow
- Understand how to implement skips and branches
- Understand how to hide menu items

IGI Components

The Concept of Skipping Steps in a Workflow

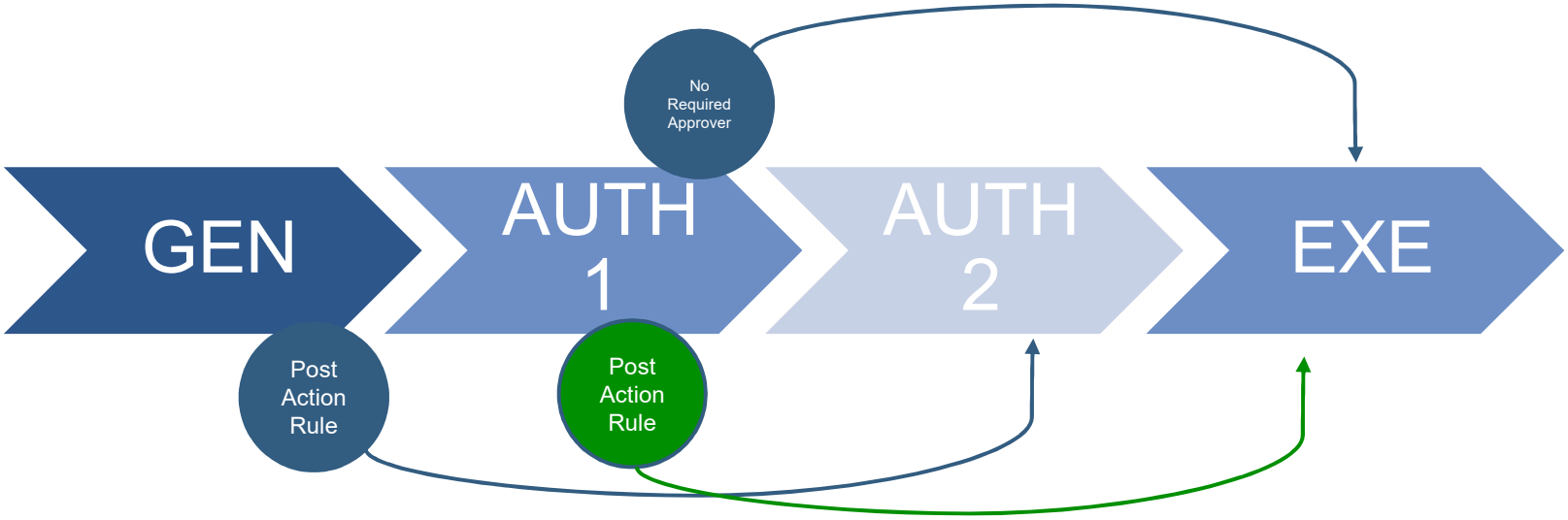
IGI Workflow: Basic Process

- A basic process is generally a sequence
  - A linear flow of activities executed in sequence

- We can alter the linear flow by:
  - GEN activity exposes SoD escalation – so Escalation process called
  - Reminder – may escalate to an Escalation process
  - Time escalation
- How can we implement a non-linear flow?
  - How can we branch to different activities based on some metrics or logic?
  - IGI does not have branching built into the workflow engine
  - Need to implement some custom solution using Rules

IGI Workflow: Implementing Skips and Branches

- If we want to implement skips and branches, need to implement on top of the linear flow
  - Skip – can skip over one approver step (AUTH activity)
  - Branch – can jump over one or more approver steps to create multiple flows
- This is done by automatically approving steps based on logic
  - Implemented as Rules (normally Post-Actions on GEN/AUTH activities)
  - Can be parameterized using variables (properties) on objects, like entitlements



IGI Workflow: Implementing Skips and Branches

- Say you have a workflow with three sets of approvers;
  - GEN -> $AUTH_1$ -> $AUTH_2$ -> $AUTH_3$ -> EXE
  - Normal IGI workflow processing is linear

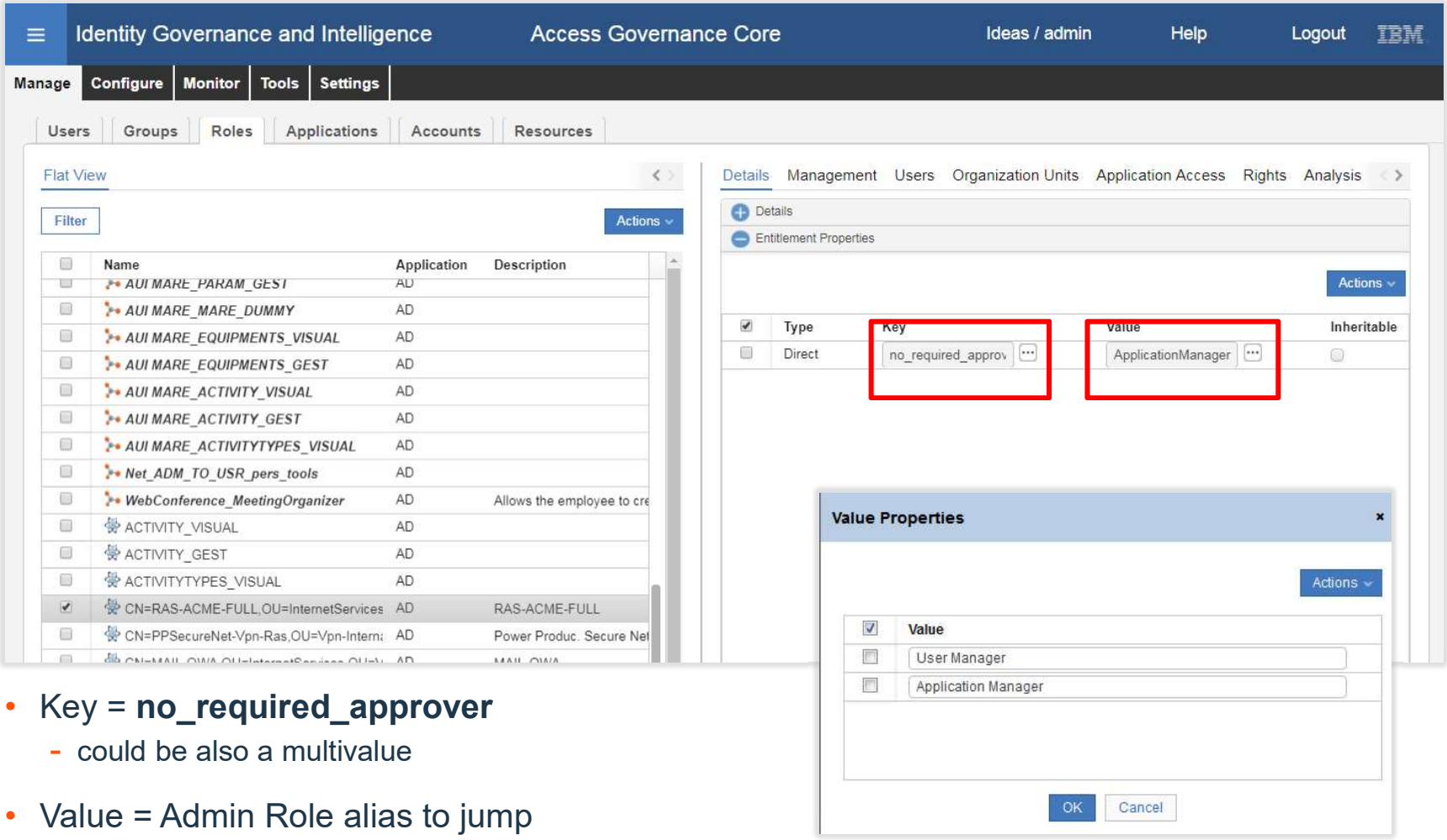
- Can use auto approval based on some condition to implement branching
- For example if we wanted to branch to $AUTH_1$ for ABC condition and $AUTH_2$ & $AUTH_3$ otherwise:

  - If ABC condition, perform $AUTH_1$ but skip (auto approve) $AUTH_2$ and $AUTH_3$

  - If !ABC condition, skip (auto approve) $AUTH_1$ and only do $AUTH_2$ and $AUTH_3$

Using Arguments/Properties to Drive Flow in Rules

- If you want the Rules to be more flexible, could use variables
- For example, could have a Property on an Entitlement and then check that in a rule when deciding whether to auto-approve (skip) the next step or not



- Key = **no_required_approver**
  - could be also a multivalue
- Value = Admin Role alias to jump

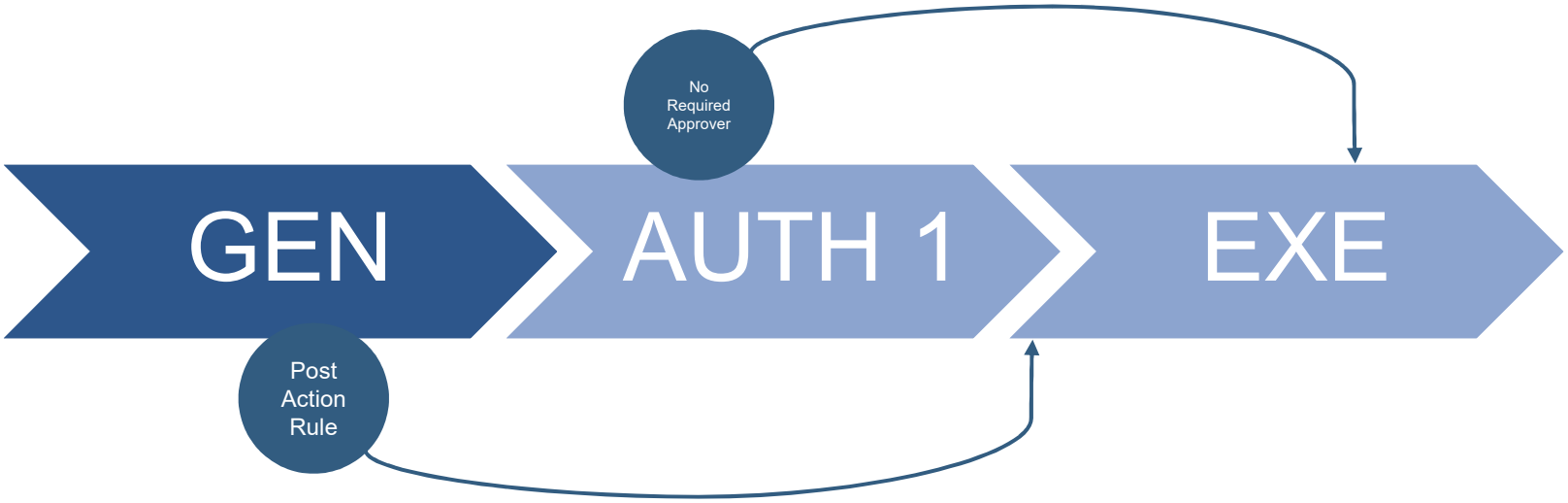
Examples of Implementing Skips and Branches in IGI Workflow

- We will show three examples:
  - All variations on the same thing
- Example 1 – Skipping a Step
  - Simple single-approval workflow with jump for all users of type “Employee”
  - If user type = Employee, skip the approval step
- Example 2 – Skipping a Step
  - Simple dual-approval workflow with jump for all users of type “Employee” for a certain approval role
  - If next approval role = “APPROVER01” and user type = “Employee”, skip the approval step
- Example 3 – Branching with Multiple Processes
  - Implementing a branch for different user types
  - Pre-actions will route to different workflow processes
  - %%%

Example 1 – Skipping a Step

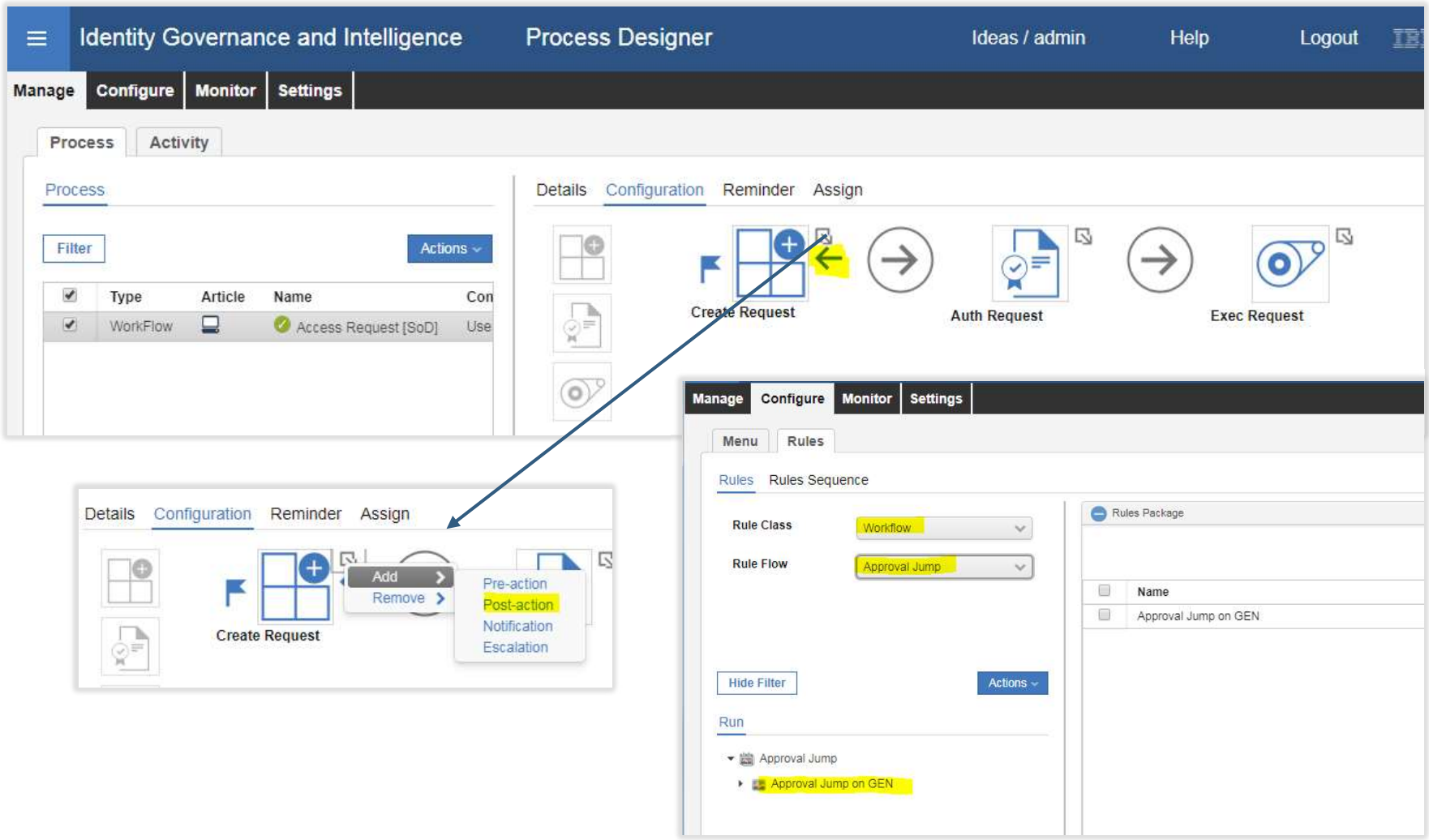
Example 1 – What Process Are We Trying to Implement?

- Have a simple single-approval workflow
  - Shows how to implement skipping a step in a workflow
- We want to skip the approval step for some users, i.e. employees
  - In this case can safely hard-code the user type in the rule
  - Only need to apply a rule to determine whether the authorize activity (approval node) can be skipped
- In this example we will:
  - Implement a (custom) Post-Action Rule on the GEN activity



Example 1 – Adding a Rule as Post-Action to GEN Activity

- We need to create a Post-action Rule to perform the approval skipping



Example 1 – The Rule

```
when
    req : SwimRequestBean(  )
then

String operator = "System";
String userTypeToSkip = "Employee";

logger.info("Skip Approver on GEN by user type.");

BeanList<SwimRequestBean> childrenList = RequestFindRule.findRequestChildren(sql, req.getId(), null);
for (SwimRequestBean childReq : childrenList) {

  SwimRequestBean fullReq = RequestFindRule.findRequestDetail(sql, childReq);

  String authAlias = "";
  for (SwimAuthorizationBean auth : fullReq.getAuthorizations()) {
    if (auth.getState() == AuthorizationStatus.AUTHORIZABLE.getCode() ) {
      authAlias = auth.getApproverType_name();
      break;
    }
  }

  if (authAlias.isEmpty()) {
    continue;
  }

  UserBean ub = new UserBean();
  ub.setId(fullReq.getBeneficiary_id());

  BeanList<UserBean> ubList = UserAction.find(sql, ub);
  if (ubList != null && !ubList.isEmpty()) {

    ub = ubList.get(0);
    String userType = ub.getPersontype_name();

    if (userType != null && userType.equals(userTypeToSkip) ) {
      SwimRequestBean swimReq = RequestAuthorizationRule.authorizeRequest(sql, fullReq, authAlias, fullReq.getPermission(), operator);
      logger.info("Request " + swimReq.getId() + " automatically approved: user " + ub.getCode() + " has type " + userType + ",
approver not need");
    }
  }
}
```

Hardcoded the user type to skip (e.g. skip if type = “Employee”)

Get the admin roles for the AUTH activity (approval step)

Get the beneficiary (the person this request is for) and
IF they are of type “Employee” (i.e. the Persontype_name of the user equals the hardcoded user type,
THEN skip over the approval step (i.e. automatically authorize the request).

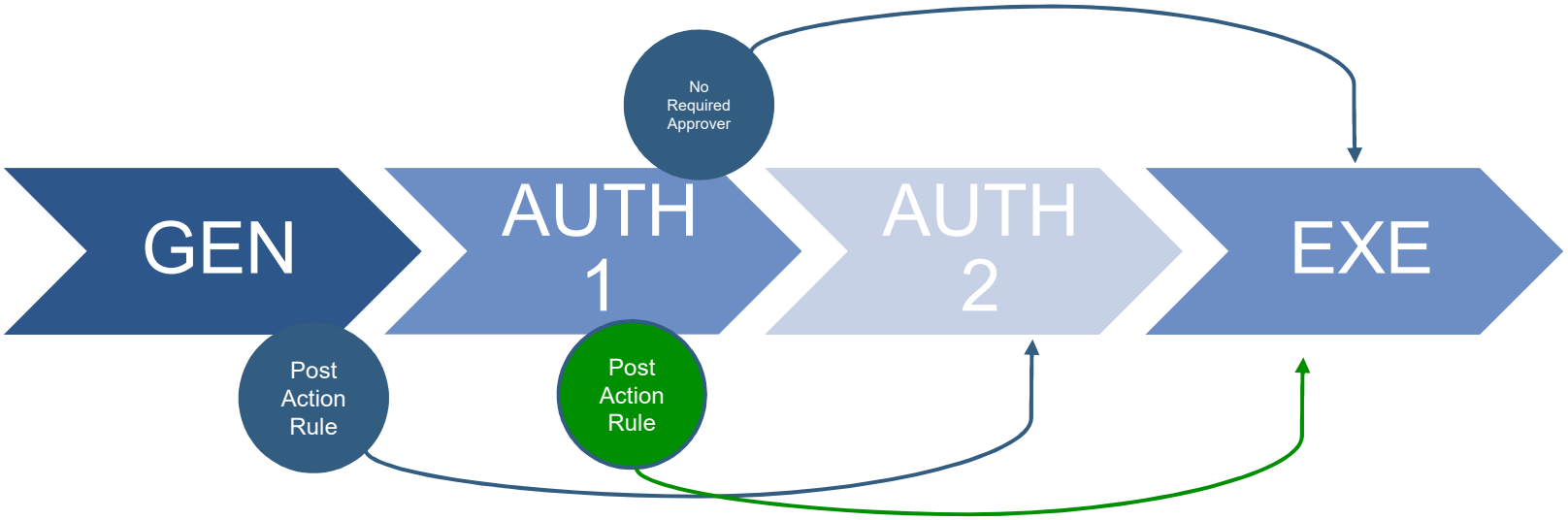
Example 1 – The Rule Explained

- The rule will take place after the Generation step
- Rule will check the beneficiary type and if the involved actor is an employee, all the approval steps will be skipped
- In the rule you need a generic identity (in this case called System) authorized to perform the involved authorization activity.
  - In this example Auth Request step, is configured to be performed by an Application Manager, so System must have the Admin Role of Application Manager
- You can customize the rule to consider other logic, good examples could be identity belonging to specific OU or entitlements flagged with custom properties etc.

Example 2 – Skipping a Step

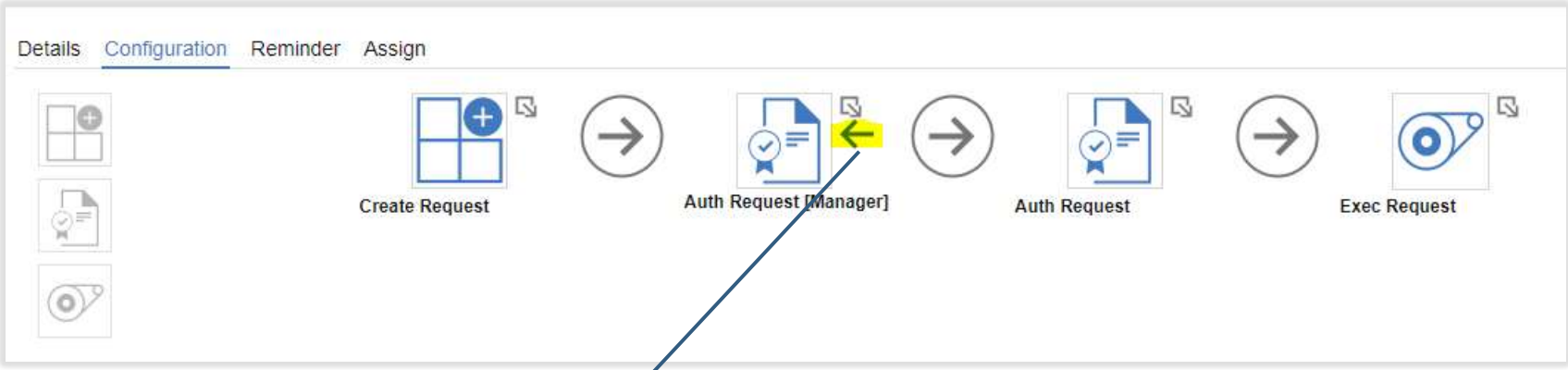
Example 2 – What Process Are We Trying to Implement?

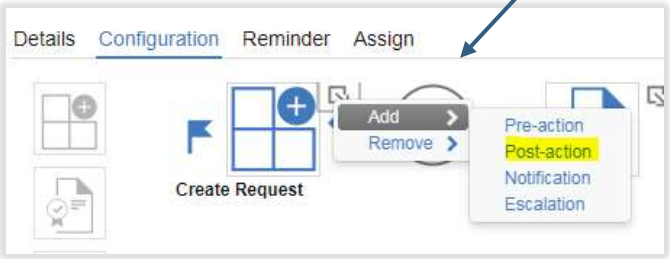
- Look at an example of how branching could be implemented
- Say we wanted to skip a second level of approval for some entitlements based on Admin Role(s)
  - Could set this value for specific entitlements in IGI
  - Then code a rule to check for this value and auto-approve the second-level, thus skipping the step
- In this example we will:
  - Use a (custom) **NoRequiredApprovers** property on entitlement
  - Implement a (custom) Post-Action Rule on the activity

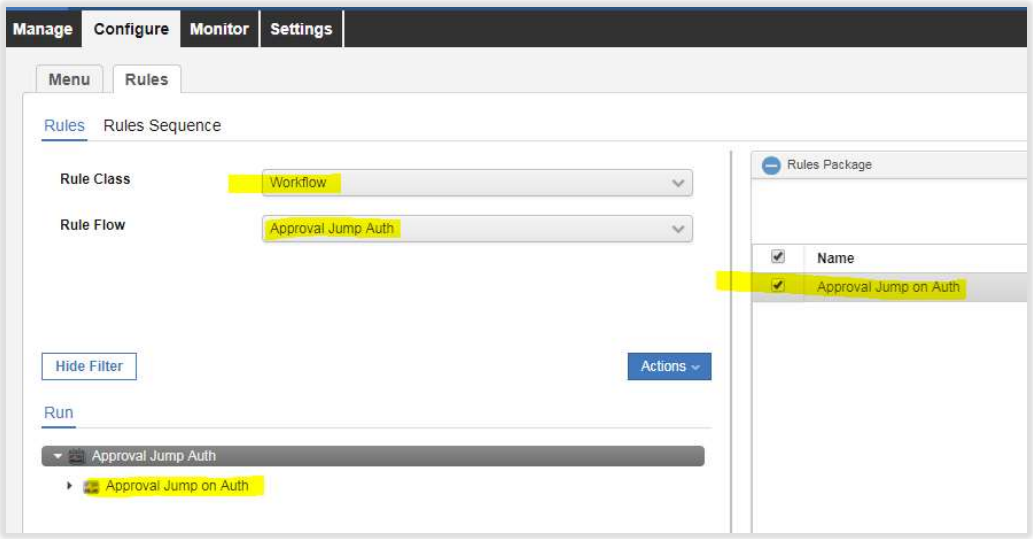


Example 2 – Adding a Rule as Post-Action to 1st AUTH Activity

- We need to create a Post-action Rule to perform the approval skipping







Example 2 – The Rule

```
when
    req : SwimRequestBean(  )
then
String operator = "System";
String userTypeToSkip = "Employee";
String authAliasToSkip = "APPROVER01";

logger.info("SkipApprover on Auth by user type.");

BeanList<SwimRequestBean> childrenList = RequestFindRule.findRequestChildren(sql, req.getId(), null);
for (SwimRequestBean childReq : childrenList) {

  SwimRequestBean fullReq = RequestFindRule.findRequestDetail(sql, childReq);

  boolean applyRule = false;
  for (SwimAuthorizationBean auth : fullReq.getAuthorizations()) {
    if (authAliasToSkip.equals(auth.getApproverType_name())) {
      if (AuthorizationStatus.AUTHORIZABLE.getCode() == auth.getState()) {
        applyRule = true;
      }
      break;
    }
  }

  if (!applyRule) {
    return;
  }

  String authAlias = "";
  for (SwimAuthorizationBean auth : fullReq.getAuthorizations()) {
    if (auth.getState() == AuthorizationStatus.AUTHORIZABLE.getCode() ) {
      authAlias = auth.getApproverType_name();
      break;
    }
  }

  if (authAlias.isEmpty()) {
    continue;
  }
```

Hardcoded the user type to skip (e.g. skip if type = “Employee”) and the role of the approver (“APPROVER01”)

Get the admin roles for the AUTH activity (approval step) and if it matches the “APPROVER01” then continue to rule, otherwise exit

Example 2 – The Rule (cont.)

```
  UserBean ub = new UserBean();
  ub.setId(fullReq.getBeneficiary_id());

  BeanList<UserBean> ubList = UserAction.find(sql, ub);
  if (ubList != null && !ubList.isEmpty()) {

  ub = ubList.get(0);
  String userType = ub.getPersontype_name();

    if (userType != null && userType.equals(userTypeToSkip) ) {
      SwimRequestBean swimReq = RequestAuthorizationRule.authorizeRequest(sql, fullReq, authAlias, fullReq.getPermission(), operator);
      logger.info("Request " + swimReq.getId() + " automatically approved: user " + ub.getCode() + " has type " + userType + ",
approver not need");
    }
  }
}
```

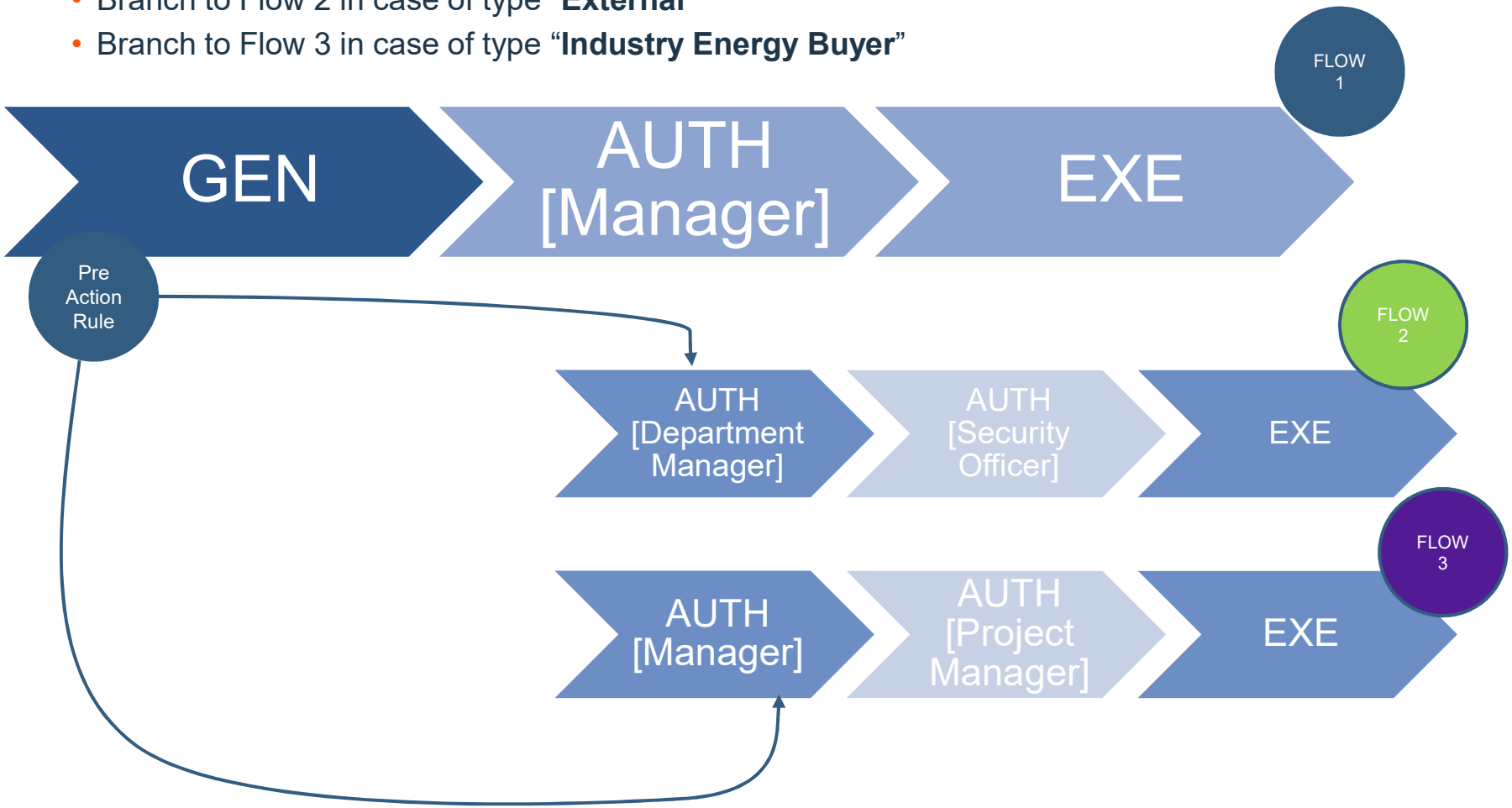
Get the beneficiary (the person this request is for) and IF they are of type “Employee” (i.e. the Persontype_name of the user equals the hardcoded user type,
THEN skip over the approval step (i.e. automatically authorize the request).

- The rule is very similar to the earlier one
- It’s just adding the step to check the admin role of the next step and only process if it matches “APPROVER01”

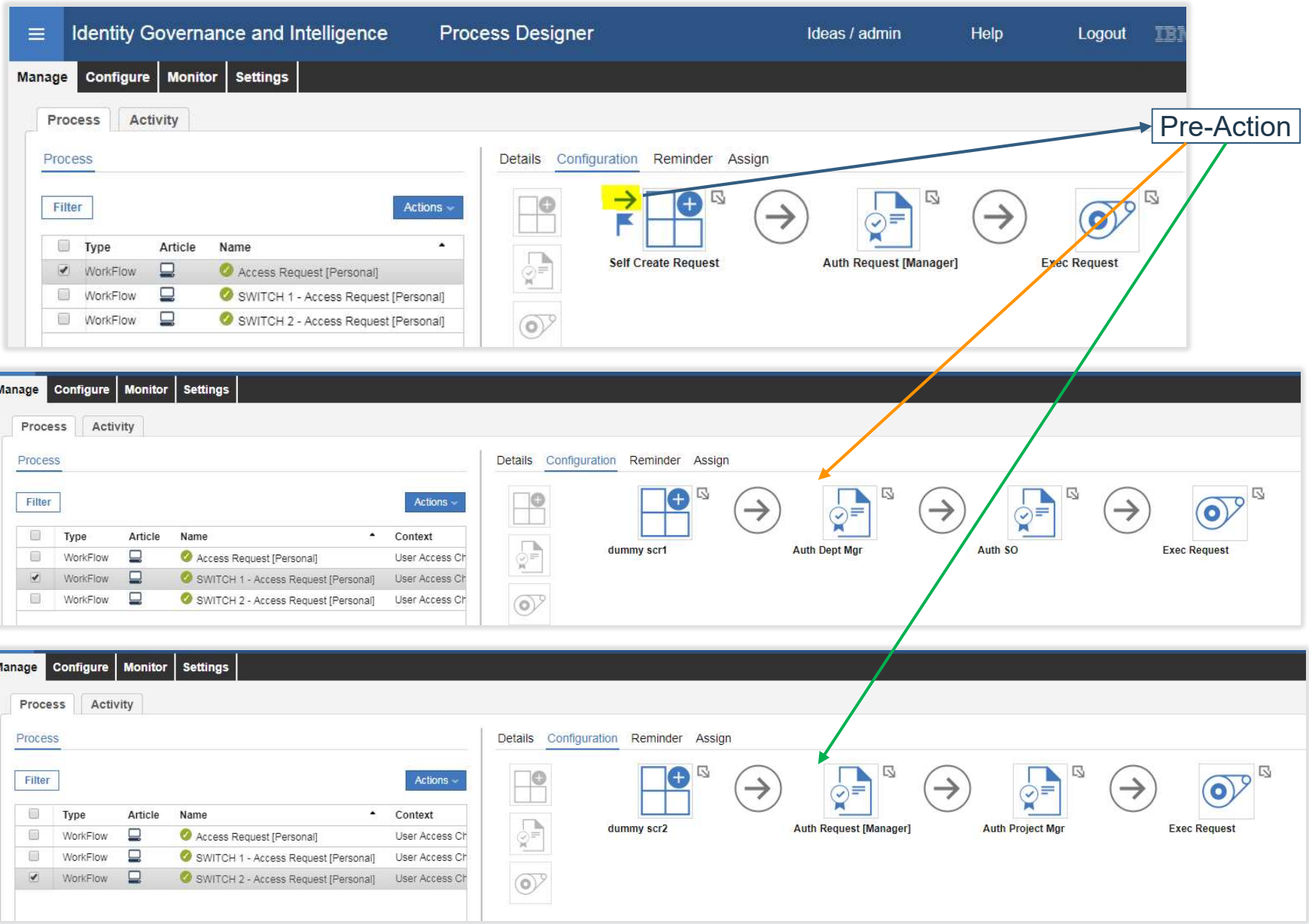
Example 3 – Implementing Branching through Multiple Processes

Example 3 – Implementing Branching

- To Branch you need a Pre-Action rule to customize your behaviour:
  - In this example I want:
    - A standard flow in case user type is “**Employee**” (Flow 1)
    - Branch to Flow 2 in case of type “**External**“
    - Branch to Flow 3 in case of type “**Industry Energy Buyer**”

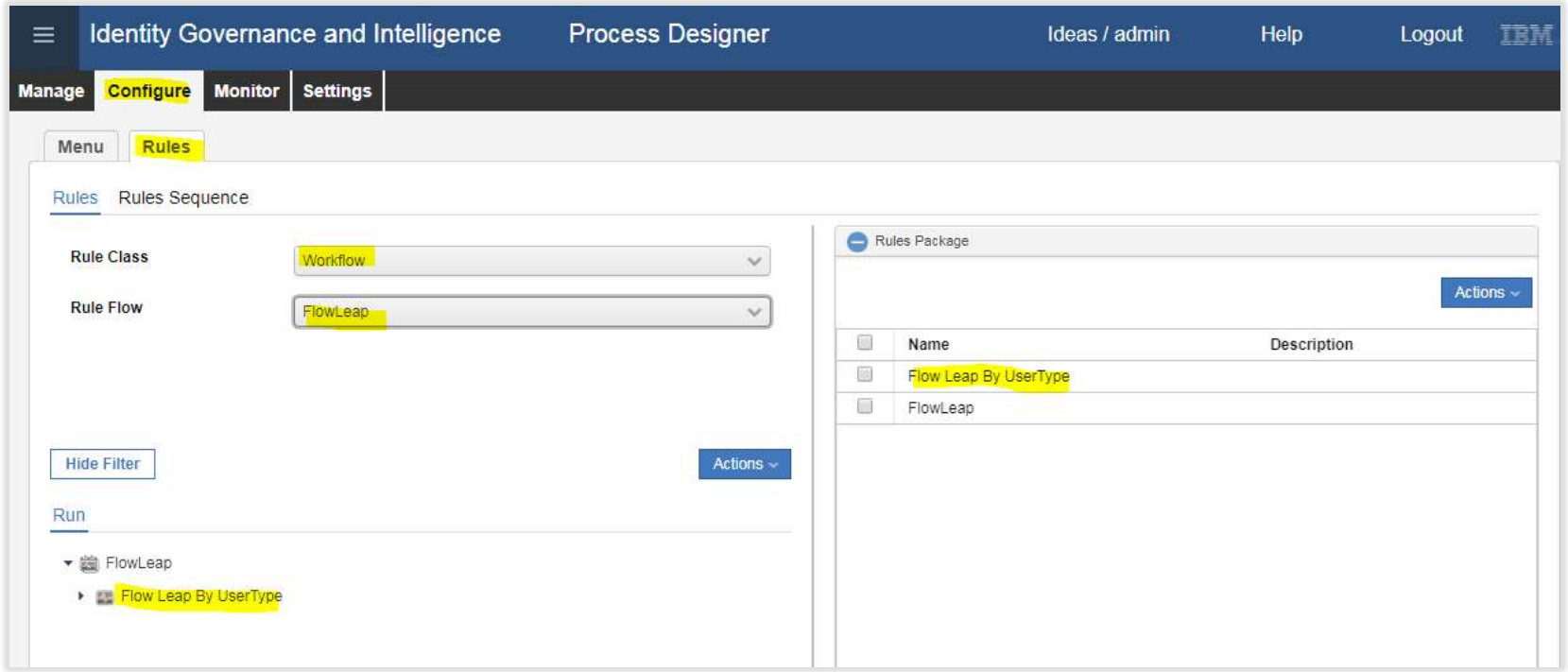


Example 3 – Implementing Branching



Example 3 – Defining the Rule

- Define The Rule



Example 3 – The Rule [1]

```
when
    request : SwimRequestBean(  )
then
//
final String Switch1FlowName = "SWITCH 1 - Access Request [Personal]";
final String Switch2FlowName = "SWITCH 2 - Access Request [Personal]";

String destinationFlowName;

UserBean beneficiaryBean = _UserAction.findUserByCode(sql, request.getBeneficiary_userid());
String beneficirayType = beneficiaryBean.getPersontype_name();

if (beneficirayType.equalsIgnoreCase("External")) {
        destinationFlowName = Switch1FlowName;
} else if (beneficirayType.equalsIgnoreCase("Industry Energy Buyer")) {
        destinationFlowName = Switch2FlowName;
} else {
        // Proceed on standard flow
        return;
}

SwimRequestBean parentReq = new SwimRequestBean();
parentReq.setApplicant_userid(request.getApplicant_userid());
parentReq.setBeneficiary_userid(request.getBeneficiary_userid());
parentReq.setNotes(request.getNotes());
request.setNotes("VOID_REQUEST_TO_REMOVE");

List<SwimEntitlementBean> rAdd = request.getRolesToAdd();
if (rAdd != null && !rAdd.isEmpty()) {
        for (SwimEntitlementBean swimEntitlementBean : rAdd) {
                parentReq.addRoleToAdd(swimEntitlementBean);
        }
}
List<SwimEntitlementBean> rRem = request.getRolesToRemove();
if (rRem != null && !rRem.isEmpty()) {
        for (SwimEntitlementBean swimEntitlementBean : rRem) {
                parentReq.addRoleToRemove(swimEntitlementBean);
        }
}
List<SwimEntitlementBean> rUpd = request.getRolesToUpdate();
if (rUpd != null && !rUpd.isEmpty()) {
        for (SwimEntitlementBean swimEntitlementBean : rUpd) {
                parentReq.addRoleToUpdate(swimEntitlementBean);
        }
}
```

Hardcoded the other two workflow processes to branch to

Find the beneficiary (user who the request is for) and then the user type for that user. Use this to determine the process to run (destinationFlowName), and if not one we're looking for drop out of the rule and continue the normal process

Need to setup a new SwimRequestBean based on the current one, to drive the other process. It includes the roles (entitlements) being added, removed or modified

Example 3 – The Rule [2]

```
// Identify process
CfgProcess cfgProcess = new CfgProcess();
cfgProcess.setName(destinationFlowName);

CfgProcessDAO cfgProcessDAO = new CfgProcessDAO(logger);
cfgProcessDAO.setDAO(sql);

// Identify GEN
CfgProcessactivity cfgPA = null;
try {
        cfgPA = cfgProcessDAO.findGenerator(cfgProcess);
} catch (Exception ex) {
        logger.info("Wokflow Process: " + destinationFlowName + " not defined!");
        throw ex;
}

UserBean systemUB = new UserBean();
systemUB.setCode("System");

GenerateRequest gRequestDAO = new GenerateRequest(logger);
gRequestDAO.setDAO(sql);
gRequestDAO.initialize("ideas", cfgPA.getPermission(), systemUB);

// Generate the parent and child request
parentReq = gRequestDAO.generate(parentReq);

logger.info("InFormal Request generated: " + parentReq.getId() + " in PENDIG");
```

Define the new process to be run and set the Data Access Object (old approach to coding)

Find the GEN activity for the new process, and fail if it doesn't exist

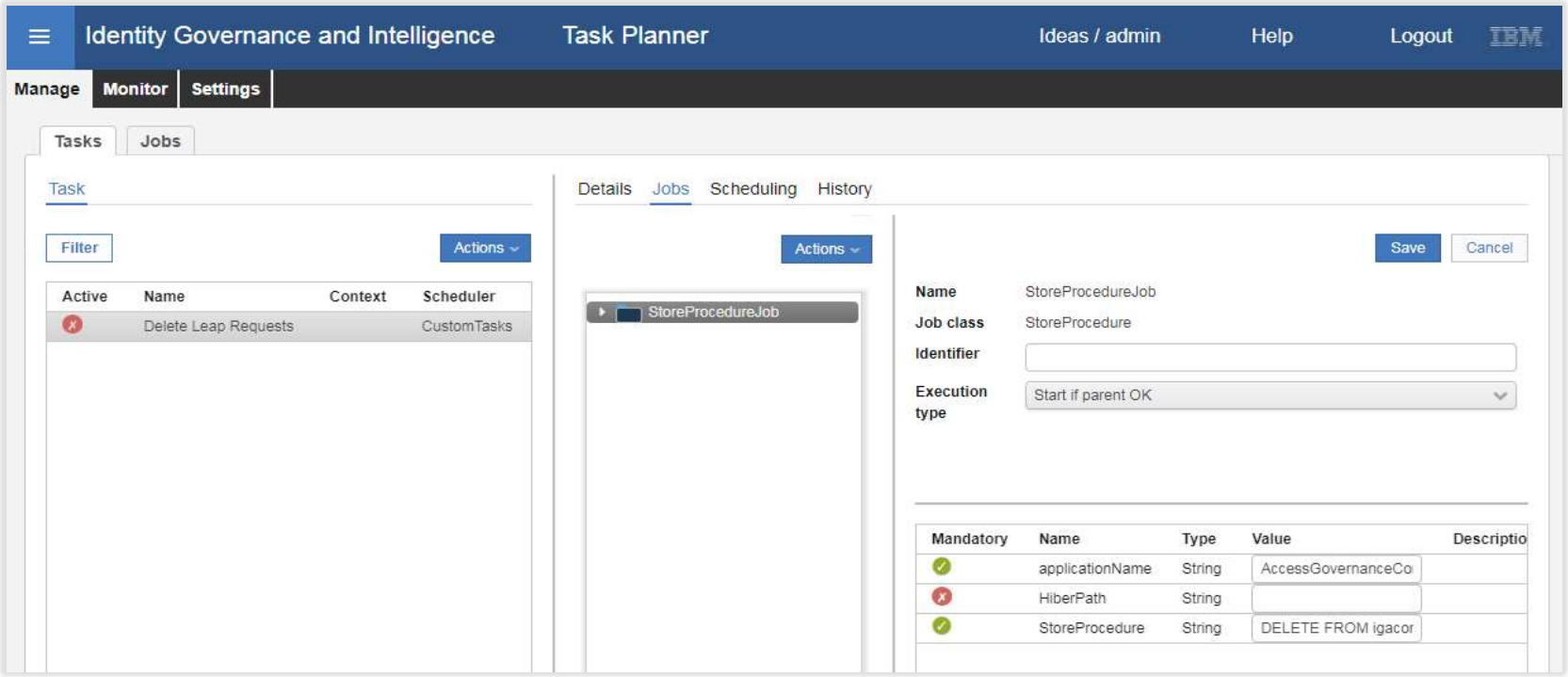
Trigger the new process

Example 3 – The Rule Explained

- The rule will take place before the Generation step
- Rule will check the beneficiary type and will generate another request in the right workflow
- API Limitations unfortunately will generate also the request for the original flow
  - Hopefully API will soon give us the possibility to skip generating the bad request
- To overcome the issue of the double request, we flagged the bad request with a specific note and than we use an easy scheduled job to delete those bad requests (**this will be simplified soon so to avoid this step**)
  - See over

Example 3 – Using a Job to Delete Redundant Processes [1]

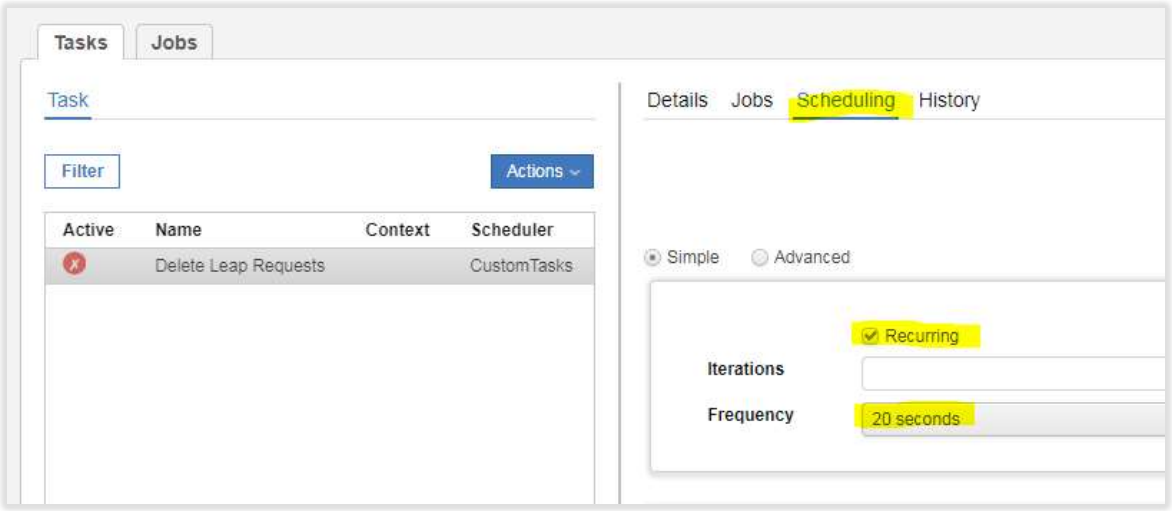
- In Task Planner define a Task that includes a standar StoreProcedureJob



Example 3 – Using a Job to Delete Redundant Processes [2]

- Parameters for StoreProcedureJob:
  - applicationName: **AccessGovernanceCore**
  - storeProcedure:
    DELETE FROM igacore.REQUEST r WHERE r.NOTES LIKE '%VOID_REQUEST_TO_REMOVE'

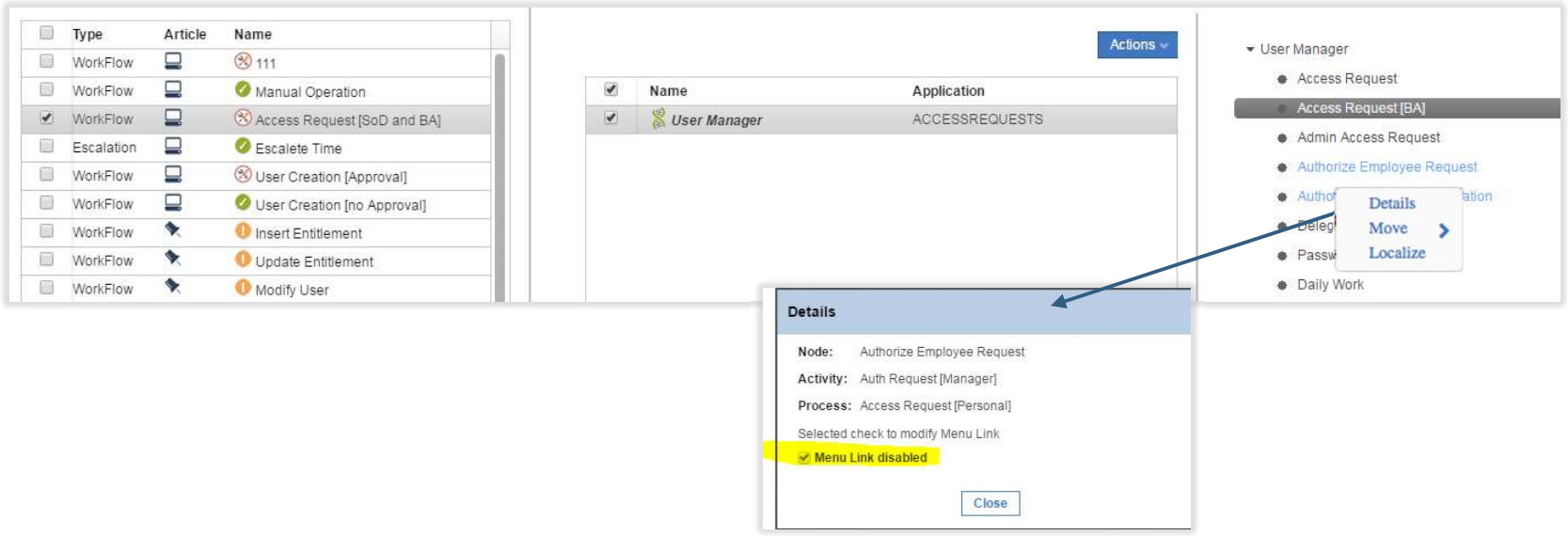
- Scheduling



Hiding redundant Menu Items

IGI Workflow: Daily Work and Hide Auth step

- Best practice is to use **Daily Work** activity for approval
  - A single menu tab should be used for “important” approval or in case of single approval
- To hide menu not required:
  - Put process in maintenance mode and select it
  - Go to the Assign tab and select the activity and role
  - Right-click the menu title for the admin role
  - Select the Details item
  - Use **Menu Link disabled**



- If some of your workflow are used for Branch, you have to hide all the GEN

Close

Module Summary

Questions?

You should now:

- Understand the concept of using rules to skip steps in a workflow
- Understand how to implement skips and branches
- Understand how to hide menu items

IBM Security

THANK YOU

FOLLOW US ON:

- ibm.com/security
- securityintelligence.com
- xforce.ibmcloud.com
- @ibmsecurity
- youtube/user/ibmsecuritysolutions

IBM