



Identity Governance  
Tech Enablement

# Skipping Steps in Workflow

## *Using Rules*

  
Fabrizio Petriconi



## 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





# 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

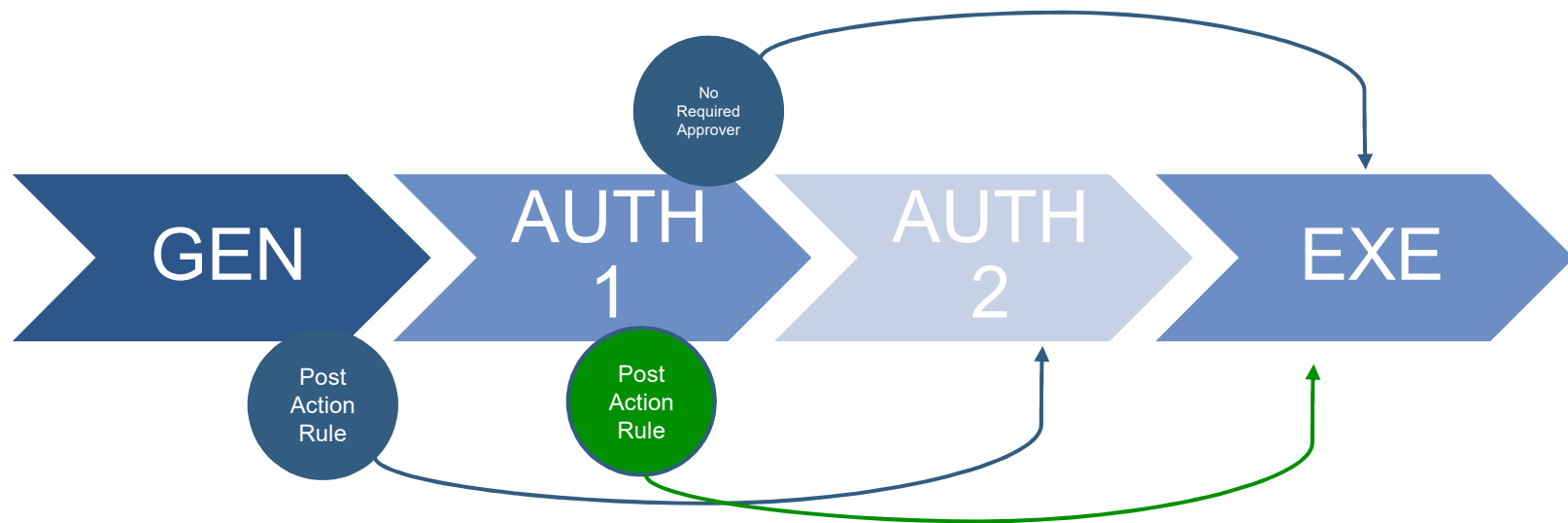


- We can alter the linear flow by:
  - GEN activity exposes SoD escalation – so Escalation process called
  - Reminder – may escalate to an Escalation process
  - Time escalation
- How can we implement a non-linear flow?
  - How can we branch to different activities based on some metrics or logic?
  - 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<sub>1</sub> -> AUTH<sub>2</sub> -> AUTH<sub>3</sub> -> 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<sub>1</sub> for ABC condition and AUTH<sub>2</sub> & AUTH<sub>3</sub> otherwise:



- If ABC condition, perform AUTH<sub>1</sub> but skip (auto approve) AUTH<sub>2</sub> and AUTH<sub>3</sub>



- If !ABC condition, skip (auto approve) AUTH<sub>1</sub> and only do AUTH<sub>2</sub> and AUTH<sub>3</sub>



# 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

The screenshot displays the IBM Identity Governance and Intelligence (IGI) interface. The top navigation bar includes 'Identity Governance and Intelligence', 'Access Governance Core', 'Ideas / admin', 'Help', and 'Logout'. Below this, a secondary navigation bar shows 'Manage', 'Configure', 'Monitor', 'Tools', and 'Settings'. The main content area is divided into two panes. The left pane, titled 'Flat View', shows a table of entitlements with columns for 'Name', 'Application', and 'Description'. The right pane, titled 'Details', shows the 'Entitlement Properties' for a selected entitlement. In the 'Details' pane, the 'Key' is 'no\_required\_approv' and the 'Value' is 'ApplicationManager'. A 'Value Properties' dialog box is open, showing a list of values: 'User Manager' and 'Application Manager'. The 'Value' column is checked, and the 'Application Manager' value is selected.

Name	Application	Description
AUI MARE_PARAM_GEST	AD	
AUI MARE_MARE_DUMMY	AD	
AUI MARE_EQUIPMENTS_VISUAL	AD	
AUI MARE_EQUIPMENTS_GEST	AD	
AUI MARE_ACTIVITY_VISUAL	AD	
AUI MARE_ACTIVITY_GEST	AD	
AUI MARE_ACTIVITYTYPES_VISUAL	AD	
Net_ADM_TO_USR_pers_tools	AD	
WebConference_MeetingOrganizer	AD	Allows the employee to cre
ACTIVITY_VISUAL	AD	
ACTIVITY_GEST	AD	
ACTIVITYTYPES_VISUAL	AD	
CN=RAS-ACME-FULL,OU=InternetServices	AD	RAS-ACME-FULL
CN=PPSecureNet-Vpn-Ras,OU=Vpn-Intern	AD	Power Produc. Secure Net
CN=MAIL_OWA,OU=InternetServices,OU=...	AD	MAIL_OWA

Type	Key	Value	Inheritable
<input checked="" type="checkbox"/>	Direct	no_required_approv	ApplicationManager

Value
<input checked="" type="checkbox"/> User Manager
<input type="checkbox"/> Application Manager

- 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
  - %%%

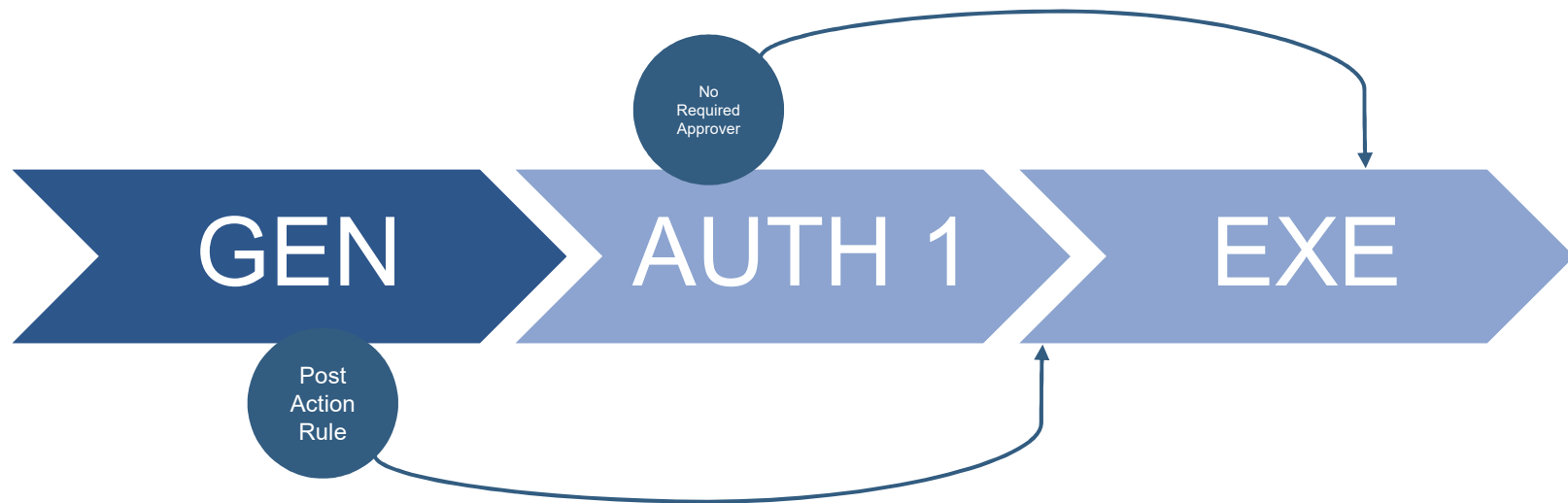


# Example 1 – Skipping a Step



## 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

The screenshot displays the IBM Identity Governance and Intelligence Process Designer interface. The top navigation bar includes 'Identity Governance and Intelligence', 'Process Designer', 'Ideas / admin', 'Help', and 'Logout'. Below this, a secondary navigation bar shows 'Manage', 'Configure', 'Monitor', and 'Settings'. The main workspace is divided into two panes: 'Process' and 'Activity'.

In the 'Process' pane, a table lists activities:

Type	Article	Name	Con
Workflow	Access Request [SoD]	Use	

The 'Activity' pane shows a workflow diagram with three activities: 'Create Request', 'Auth Request', and 'Exec Request'. A yellow arrow points from the 'Create Request' activity to a detailed configuration window.

The configuration window for 'Create Request' shows a context menu with options: 'Add', 'Remove', 'Pre-action', 'Post-action', 'Notification', and 'Escalation'. The 'Post-action' option is highlighted.

The 'Rules' configuration window is also shown, displaying the 'Rules Sequence' section. It includes a 'Rule Class' dropdown set to 'Workflow' and a 'Rule Flow' dropdown set to 'Approval Jump'. The 'Run' section shows a list of rules, including 'Approval Jump' and 'Approval Jump on GEN'.

## 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.

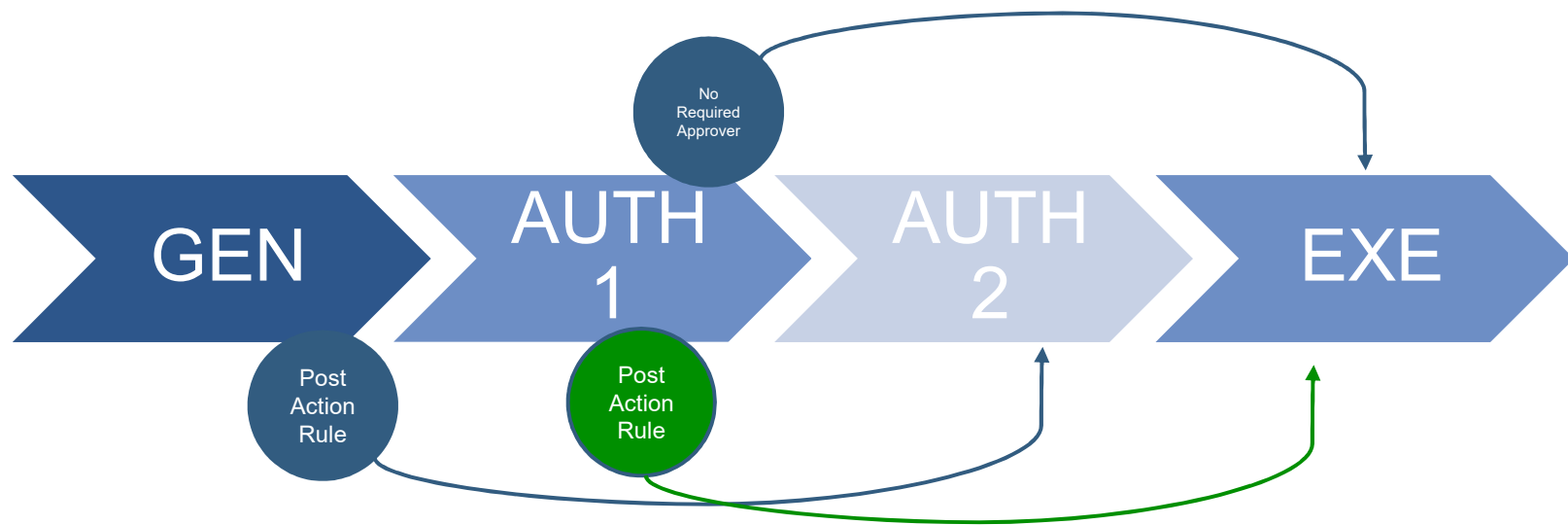


## Example 2 – Skipping a Step



## 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 1<sup>st</sup> AUTH Activity

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

The image illustrates the process of adding a post-action rule to the first authentication activity in a workflow. It consists of three main parts:

- Top Workflow Diagram:** A sequence of activities: 'Create Request', 'Auth Request [Manager]', 'Auth Request', and 'Exec Request'. A yellow arrow points to the 'Auth Request [Manager]' activity, indicating it is the target for the rule.
- Bottom Left Detail View:** A zoomed-in view of the 'Create Request' activity configuration. A context menu is open, showing options: 'Pre-action', 'Post-action' (highlighted in yellow), 'Notification', and 'Escalation'. The 'Add' button is also visible.
- Bottom Right Configuration Panel:** The 'Rules' configuration window. It shows the 'Rule Class' set to 'Workflow' and the 'Rule Flow' set to 'Approval Jump Auth'. The 'Run' section shows a list of rules, with 'Approval Jump on Auth' selected. The 'Rules Package' section on the right shows a list of rules, with 'Approval Jump on Auth' selected.

## 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”

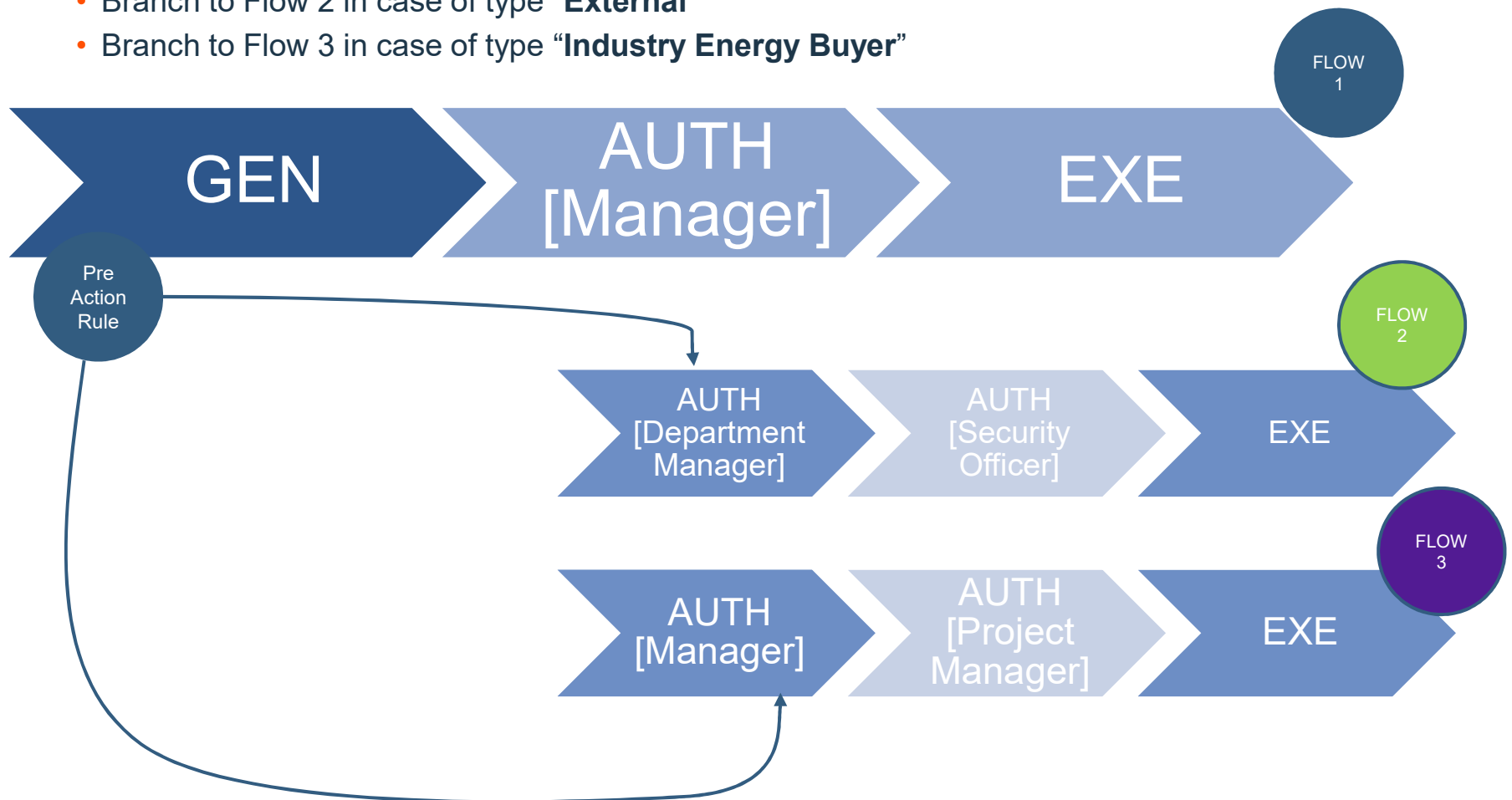


## **Example 3 – Implementing Branching through Multiple Processes**



## 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

The image displays three sequential screenshots of the IBM Identity Governance and Intelligence Process Designer interface, illustrating the implementation of branching in a process flow. The interface includes a top navigation bar with 'Identity Governance and Intelligence', 'Process Designer', 'Ideas / admin', 'Help', and 'Logout'. Below this is a secondary navigation bar with 'Manage', 'Configure', 'Monitor', and 'Settings'. The main workspace is divided into 'Process' and 'Activity' tabs, with a 'Details' tab selected in the 'Configuration' section.

**Top Screenshot:** The 'Process' tab shows a list of activities. The 'Activity' tab shows a process flow starting with 'Self Create Request', followed by 'Auth Request [Manager]', and 'Exec Request'. A blue arrow points from the 'Auth Request [Manager]' activity to a box labeled 'Pre-Action'.

**Middle Screenshot:** The 'Process' tab shows a list of activities. The 'Activity' tab shows a process flow starting with 'dummy scr1', followed by 'Auth Dept Mgr', 'Auth SO', and 'Exec Request'. A green arrow points from the 'Auth Dept Mgr' activity to the 'Pre-Action' box.

**Bottom Screenshot:** The 'Process' tab shows a list of activities. The 'Activity' tab shows a process flow starting with 'dummy scr2', followed by 'Auth Request [Manager]', 'Auth Project Mgr', and 'Exec Request'. A green arrow points from the 'Auth Request [Manager]' activity to the 'Pre-Action' box.

The 'Pre-Action' box is a label with a blue border, positioned to the right of the process flow diagrams. It is connected to the 'Auth Request [Manager]' activity in the top screenshot, the 'Auth Dept Mgr' activity in the middle screenshot, and the 'Auth Request [Manager]' activity in the bottom screenshot by colored arrows (blue, green, and green respectively).

## Example 3 – Defining the Rule

- Define The Rule

The screenshot displays the IBM Identity Governance and Intelligence Process Designer interface. The top navigation bar includes 'Identity Governance and Intelligence', 'Process Designer', 'Ideas / admin', 'Help', 'Logout', and the IBM logo. Below this, a secondary navigation bar shows 'Manage', 'Configure' (highlighted), 'Monitor', and 'Settings'. The 'Configure' section has a 'Menu' tab and a 'Rules' sub-tab. The main workspace is divided into two panels. The left panel, titled 'Rules Sequence', contains 'Rule Class' (set to 'Workflow') and 'Rule Flow' (set to 'FlowLeap'). Below these are 'Hide Filter' and 'Actions' buttons. The right panel, titled 'Rules Package', contains a table with columns 'Name' and 'Description'. The table lists two items: 'Flow Leap By UserType' and 'FlowLeap'. The 'Flow Leap By UserType' item is highlighted in yellow. Below the table is an 'Actions' button.

Identity Governance and Intelligence Process Designer Ideas / admin Help Logout IBM

Manage Configure Monitor Settings

Menu Rules

Rules Rules Sequence

Rule Class Workflow

Rule Flow FlowLeap

Hide Filter Actions

Run

FlowLeap

Flow Leap By UserType

Rules Package

Actions

	Name	Description
<input type="checkbox"/>	Flow Leap By UserType	
<input type="checkbox"/>	FlowLeap	

## 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.getPersonstype_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("Workflow 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 PENDING");
```

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 then 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

The screenshot displays the IBM Identity Governance and Intelligence Task Planner interface. The top navigation bar includes 'Identity Governance and Intelligence', 'Task Planner', 'Ideas / admin', 'Help', and 'Logout'. Below this, a secondary bar shows 'Manage', 'Monitor', and 'Settings'. The main content area is divided into two sections: 'Tasks' and 'Jobs'.

In the 'Tasks' section, a table lists tasks with columns 'Active', 'Name', 'Context', and 'Scheduler'. One task is visible: 'Delete Leap Requests' with a red 'x' icon, context 'CustomTasks', and scheduler 'CustomTasks'.

In the 'Jobs' section, a table lists jobs with columns 'Active', 'Name', 'Context', and 'Scheduler'. One job is visible: 'StoreProcedureJob' with a blue play icon, context 'CustomTasks', and scheduler 'CustomTasks'.

The 'Details' tab for the 'StoreProcedureJob' is active, showing configuration fields: 'Name' (StoreProcedureJob), 'Job class' (StoreProcedure), 'Identifier' (empty), and 'Execution type' (Start if parent OK). Below these fields is a table with columns 'Mandatory', 'Name', 'Type', 'Value', and 'Description'.

Mandatory	Name	Type	Value	Description
✓	applicationName	String	AccessGovernanceCo	
✗	HiberPath	String		
✓	StoreProcedure	String	DELETE FROM igacor	

## 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

The screenshot displays the IBM Security interface for configuring a task. The 'Tasks' tab is selected, and the 'Task' sub-tab is active. A table lists the task 'Delete Leap Requests' with a status of 'Active' (indicated by a red 'x' icon) and a scheduler of 'CustomTasks'. The 'Scheduling' sub-tab is selected on the right, showing the 'Simple' scheduling option. The 'Recurring' checkbox is checked, and the 'Frequency' is set to '20 seconds'.

Active	Name	Context	Scheduler
<input checked="" type="checkbox"/>	Delete Leap Requests		CustomTasks

Details Jobs **Scheduling** History

☒ Simple ☐ Advanced

☒ Recurring

Iterations

Frequency **20 seconds**



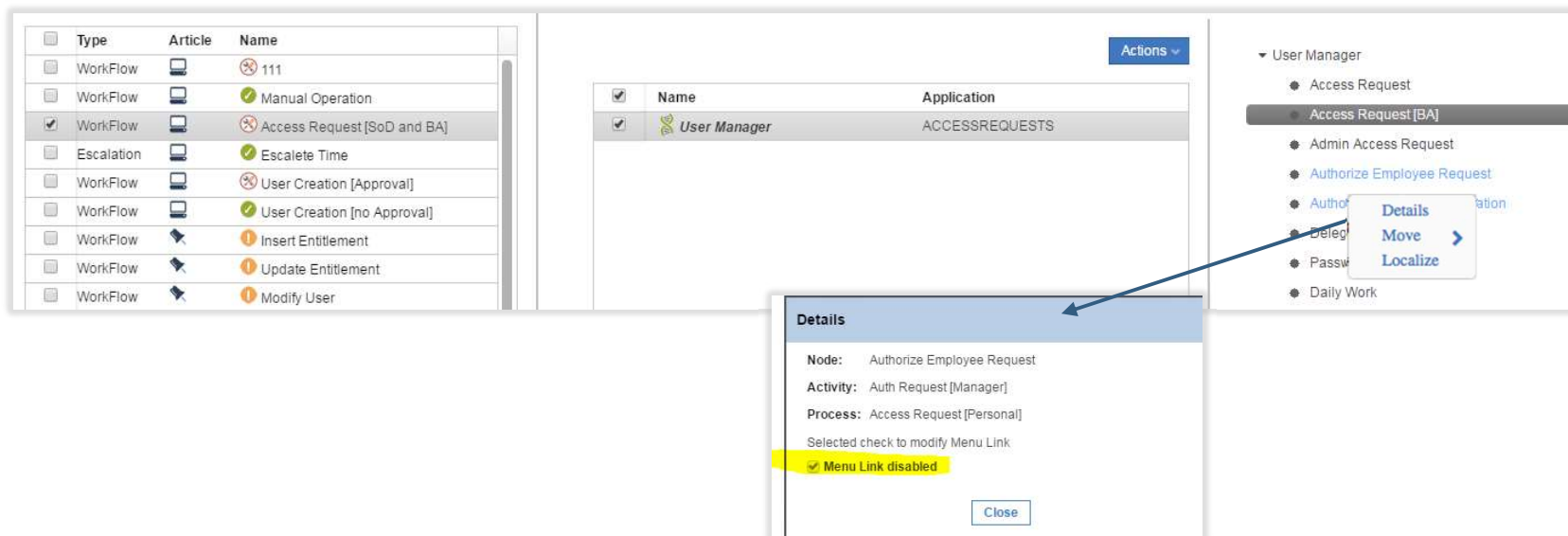
# Hiding redundant Menu Items





# 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



# THANK YOU

FOLLOW US ON:



[ibm.com/security](http://ibm.com/security)



[securityintelligence.com](http://securityintelligence.com)



[xforce.ibmcloud.com](http://xforce.ibmcloud.com)



[@ibmsecurity](https://twitter.com/ibmsecurity)



[youtube/user/ibmsecuritysolutions](https://youtube/user/ibmsecuritysolutions)

© Copyright IBM Corporation 2016. All rights reserved. The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and / or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. IBM, the IBM logo, and other IBM products and services are trademarks of the International Business Machines Corporation, in the United States, other countries or both. Other company, product, or service names may be trademarks or service marks of others.

**Statement of Good Security Practices:** IT system security involves protecting systems and information through prevention, detection and response to improper access from within and outside your enterprise. Improper access can result in information being altered, destroyed, misappropriated or misused or can result in damage to or misuse of your systems, including for use in attacks on others. No IT system or product should be considered completely secure and no single product, service or security measure can be completely effective in preventing improper use or access. IBM systems, products and services are designed to be part of a lawful, comprehensive security approach, which will necessarily involve additional operational procedures, and may require other systems, products or services to be most effective.

IBM DOES NOT WARRANT THAT ANYSYSTEMS, PRODUCTS OR SERVICES ARE IMMUNE FROM, OR WILL MAKE YOUR ENTERPRISE IMMUNE FROM, THE MALICIOUS OR ILLEGAL CONDUCT OF ANY PARTY.

