

A

Pega – Day 9



# Objectives

- **Ruleset Stack**
- **Rule Resolution**

## Rulesets

Rules are the building blocks of a Pega application. The rule type determines the type of behavior modeled by the rule. In Pega, every instance of every rule type belongs to a **ruleset**. A ruleset is a container or an organizational construct used to identify, store, and manage a set of rules. The primary function of a ruleset is to group rules together for distribution. Understanding the relationship between an application and rulesets is essential in order to understand development and run-time behavior.

Each ruleset has versions.

## Application Rulesets

An application contains a set of rulesets. In the following example, *HRApps* and *HRAppsInt* contain application configuration. The two organizational rulesets — in this example, *TGB* and *TGBInt* — contain reusable organizational assets, such as data structures.

**Note: The rulesets ending in *Int* are used for rules related to integration.**

Add additional rulesets for reusable functionality that you might want to migrate to other applications. For example, the integration wizards create separate rulesets for each integration. The CreditCheck ruleset holds the integration assets for a connector used to perform a background check.

The screenshot displays the 'Definition' tab of the application configuration interface. The top navigation bar includes 'Definition', 'Cases & data', 'Mobile', 'Documentation', 'Integration & security', and 'History'. The main content area is divided into several sections:

- Built on application(s):** Contains an 'Add application' button and a table with columns 'Name' and 'Version'. One entry is visible: '1 HRFW 01.01.01'.
- Current development branches:** Contains 'Add branch' and 'Create branch ruleset' buttons. Below, it states 'No branches added' and has a 'Merge branches' button.
- Presentation:** Features a 'Skin\*' dropdown menu set to 'HRApps' and a checked 'Render in HTML5' option.
- Application rulesets:** Contains an 'Add ruleset' button and a list of six rulesets, each with a dropdown menu and a trash icon. The rulesets are:
  - 1 HRApps:01-01-01
  - 2 HRAppsInt:01-01-01
  - 3 TGB:01-01-01
  - 4 TGBInt:01-01-01
  - 5 UI-Kit-7:07-01
  - 6 CreditCheck:01-01

## Production Rulesets

Production rulesets have at least one unlocked ruleset version in the production environment. Production rulesets include rules that are updated in the production environment. The most common use of production rulesets is for delegated rules. However, production rulesets can be used for any use case requiring rules to be updated in a production environment.

Production rulesets are configured in the Advanced tab on the application record. In addition, the production ruleset needs to be specified in the access group.

The screenshot shows the 'Advanced' configuration tab for an application record. It features two main sections for rulesets:


- Component and shared rulesets:** Includes a '+ Add component or shared ruleset' button, a list with one item (numbered '1') in a dropdown menu, and a '+ Add production ruleset' button.
- Production rulesets (customization):** Includes a '+ Add production ruleset' button, a list with one item (numbered '1') in a dropdown menu, and a '+ Add production ruleset' button.

Below these sections, there is a checked checkbox for 'Place properties on thread page only (5-4 or later)' and a 'Log off redirection' dropdown menu set to 'Show Log off screen'.

## Ruleset Stack

The ruleset stack indicates the rulesets that are available to the application for a given operator session. The ruleset stack is available in the operator profile **Operator > Profile**.

**Note: The order of the rulesets is important. The rule resolution algorithm refers to the order of the ruleset in the ruleset list. Rulesets at the top of the list have higher precedence.**

	Name Administrator	Organization TGB
	Position	Division Div
	ID Admin@TGB	Unit Unit
	Phone	Work Group default@TGB
<b>Skills</b>		<b>Access</b>
Name	Rating	Application HRApps 01.01.01
		Access group HRApps:Administrators
		Portal Layout Developer
		Work Pool TGB-HRApps-Work
<b>Time</b>		<b>Server</b>
Calendar USDefault		Name pega
Time Zone America/New_York		Directory /webwb
		Node 10.61.9.195
		URI /prweb/tCayxcRmcbOsH1DHIDwnH8JjTS1zbn6S*/!STANDARD
<b>Locale Settings</b>		
<b>Current Locale</b>		
<b>Roles</b>		<b>Rulesets</b>
HRApps:Administrator		Admin@TGB:
PegaRULES:SecurityAdministrator		HRApps:01-01-01
		HRAppsInt:01-01-01
		TGB:01-01-01
		TGBInt:01-01-01
		UI-Kit-7:07-01
		CreditCheck:01-01
		HRFW:01-01
		HRFWInt:01-01
		PegaHR:01-01
		PegaHRInt:01-01
		Pega-ProcessCommander:07-10
		Pega-DeploymentDefaults:07-10
		Pega-DecisionArchitect:07-10
		Pega-LP-Mobile:07-10
		<b>Application RuleSets</b>
		HRApps (Customization RuleSet)
		HRAppsInt
		TGB
		TGBInt
		UI-Kit-7
		CreditCheck
		HRFW
		HRFWInt
		PegaHR
		PegaHRInt
		Pega-ProcessCommander
		Pega-DeploymentDefaults
		Pega-DecisionArchitect
		Pega-LP-Mobile
		Pega-LP-ProcessAndRules
		<a href="#">Availability</a>
		<a href="#">Change Password</a>
		<a href="#">Close</a>

## Ruleset Stack

**The ruleset stack is assembled by Pega when an operator is logging in.**

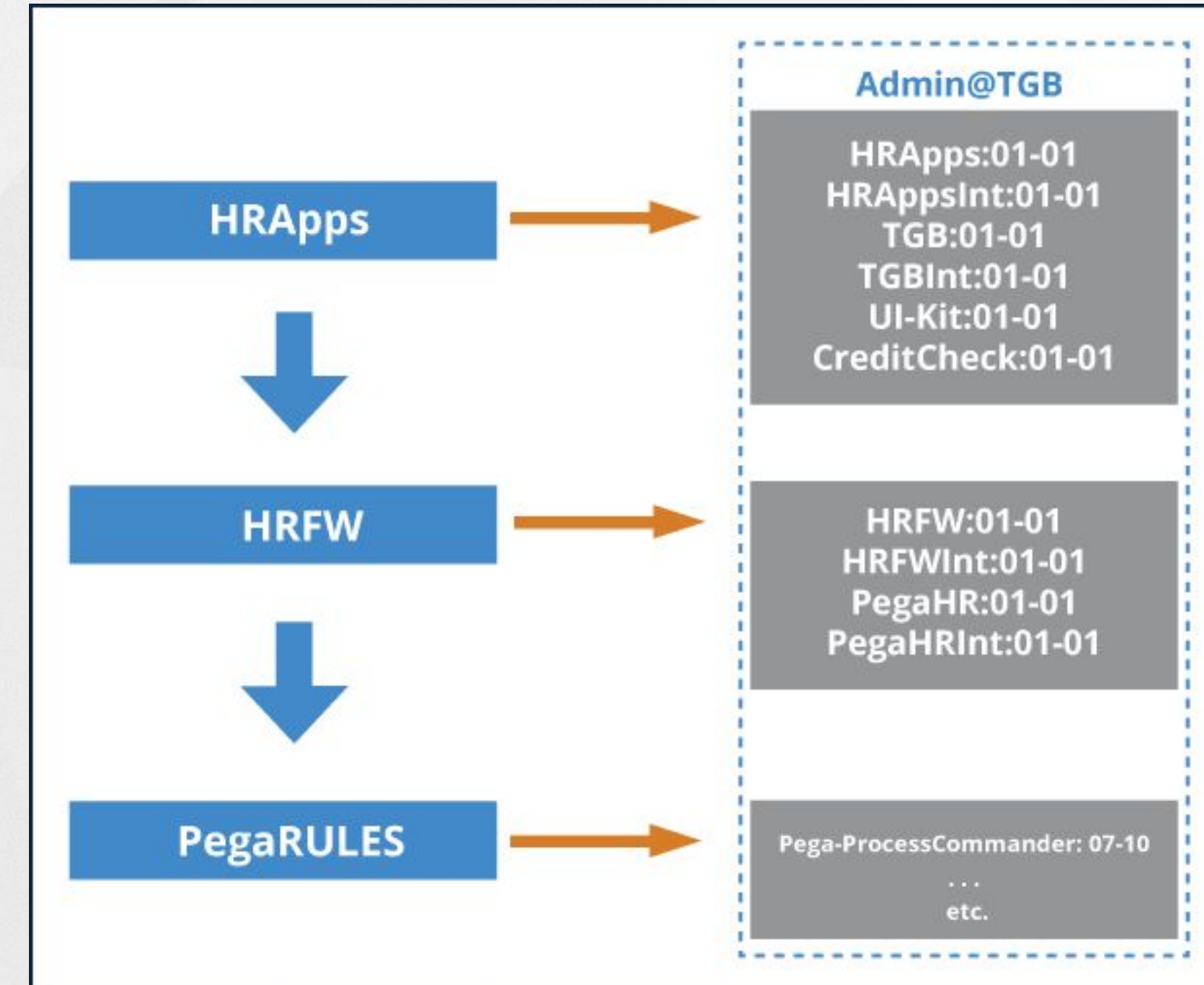
The process begins by locating the versioned application rule referenced on the access group of the operator. In rare configurations, the access group is provided as part of the requestor definition, organization, or division record.

The ruleset list consists of rulesets referenced on the application form. The built-on applications are processed repeatedly until the PegaRULES application is located.

The ruleset stack is built by stepping through the built-on applications until the PegaRULES application is located. First the PegaRULES ruleset list is added to the ruleset stack. Then the built-on application are processed recursively adding each application's ruleset to the ruleset stack on top of the previously added rulesets.

For example, the *PegaRULES* application is at the base, the *HRFW* application is built on top of *PegaRULES*, and the *HRApps* application is built on top of *HRFW*.

If you are allowed to check out rules, a **personal ruleset** is added to the top of the stack. The personal ruleset has the name of the operator ID and contains the rules checked out by the operator.

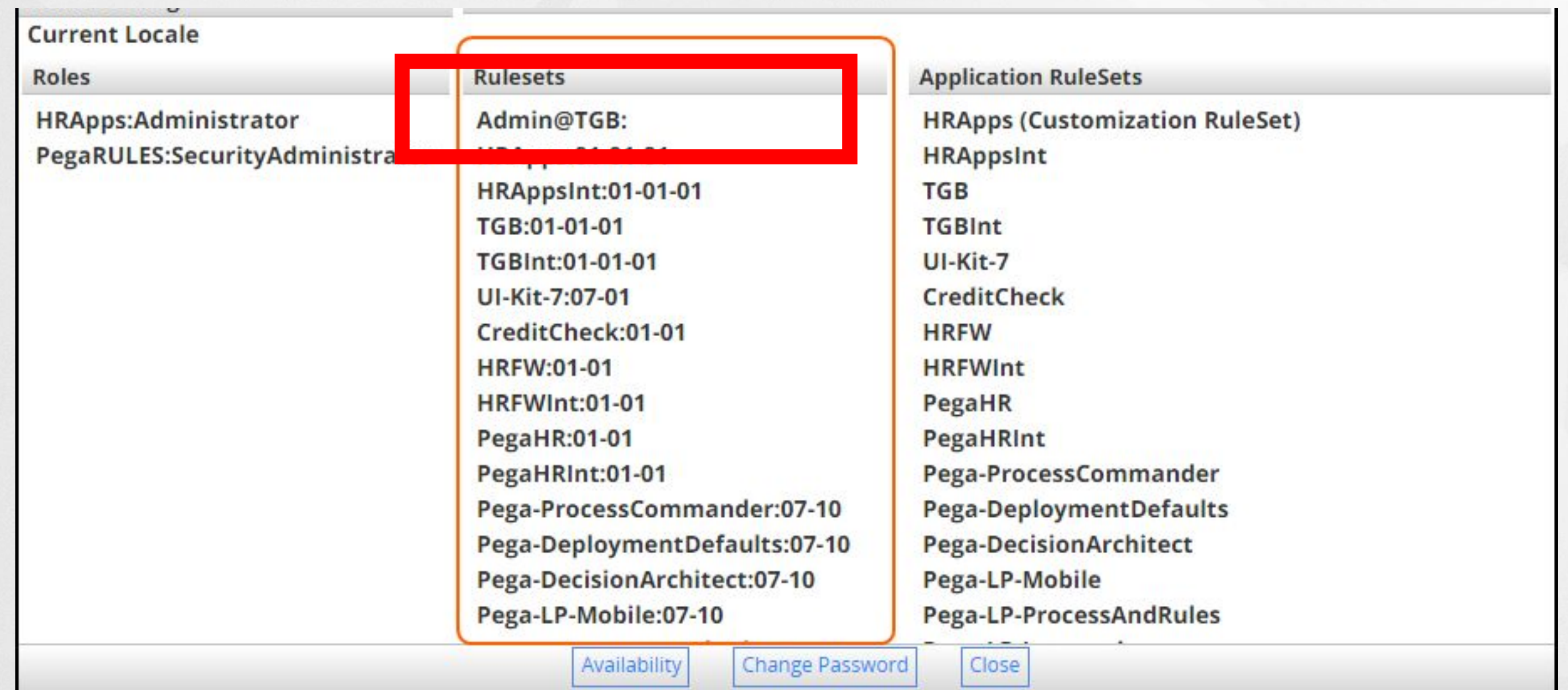


## Personal Ruleset

A **personal ruleset** contains copies of rule instances. Personal rulesets are used **with the check out and check in** features. A personal ruleset is sometimes called a private ruleset.

Applications are typically developed by a team. Multiple team members may check out rules to work on the same application in a coordinated way. The check-out feature ensures that one rule is not being edited by different team members at the same time.

The system enforces use of check-out and check-in operations when a ruleset has the check-out facility turned on. Before you can change that rule, you must perform either the standard **check-out** or **private check-out** operation.





## Check-Out a rule

When you check out a rule, you are creating a private copy of the rule that you can modify and test.

On the Security tab on the ruleset, select **Use check-out?** to enable check-out.

On the Operator record Security tab, operators need to have the **Allow Rule Check out** selected in order to update rules in rulesets that require check-out.

The check-out button displays on rules that are in unlocked rulesets. If a developer checks out a rule, no one else may check the rule out until it is checked back in by the developer.

If a rule is not available for check-out, it is already checked out by someone else, or in a locked ruleset version.

When the ruleset is in a locked ruleset version, the **private edit button** is displayed instead of the check-out button. Private edit is a special case of the standard check-out that allows a developer to prototype or test changes to a rule that is not available for standard check-out. When an operator checks out or selects private edit for a rule, a copy of the rule is placed in the personal ruleset.

Versions Security Category History

**ENTER PASSWORD VALUES**

To update this RuleSet

**RULE MANAGEMENT**

Local customization?  **Use check-out?**

Work class used for approval process Work-RuleCheckIn

Work flow to use for approvals ApproveRuleChanges

**DEFINE PASSWORDS**

To Add a RuleSet Version

To Update a RuleSet Version

To Update this RuleSet

Profile Work Security History

**Access Settings**

**Allow rule check out**

Use external authentication

Starting activity to execute

Data-Portal.ShowDesktop

License type

Named ▾

## CHECKING IN A RULE

When a rule is checked in, the checked-in rule replaces the original base rule. Add a comment describing the changes to the rule. The check-in comments can be viewed on the rule History tab.

Use the bulk action feature to check-in, open, or delete several checked out rules at the same time. The bulk action feature is located in the Private Explorer menu or under the check mark icon in the header.

Rules Checked Out by "Administrator"

Displaying 3 records [Clear Comments](#)

[Select All](#) [Clear All](#)

SELECT	RULE TYPE ▲	NAME	APPLIES TO	DESCRIPTION	RULESET	IS PRIVATE CHECKOUT?	CHECK-IN COMMENTS	STATUS
<input type="checkbox"/>	Declare Expression	.As...	TGB-...	Assessmen...	HRApps:01-01-01	No	<input type="text"/>	
<input type="checkbox"/>	Section	Col...	TGB-...	Collect Per...	HRApps:01-01-01	No	<input type="text"/>	
<input type="checkbox"/>	When	IsE...	TGB-...	IsEmploye...	HRApps:01-01-01	No	<input type="text"/>	

Bulk Prefix

## Rule Resolution - Definition

**Rule resolution** is the search algorithm that the system uses to find the best or most appropriate rule instance to apply in a situation.

Rule resolution occurs whenever a rule is needed to accomplish processing of a case. As you create applications, the choices you make when defining the values for the key parts of a rule are based on how you want the rule to be found by rule resolution.

**Rule resolution** applies to all but a few rule types — classes that inherit from the Rule- base class.

Rule resolution does not apply to rules that are instances of classes derived from any other abstract base class such as *Data-, System-, or Work-*.

The following are examples of instances of rules derived from the abstract System- base class;

- Operator IDs (Data-Admin-Operator-ID)
- Email listeners (Data-Admin-Connect-EmailListener)
- Operator's favorites (System-User-MyRules)
- The rule check-in process (Work-RuleCheckIn)

## The inputs and outputs of rule resolution

### Inputs

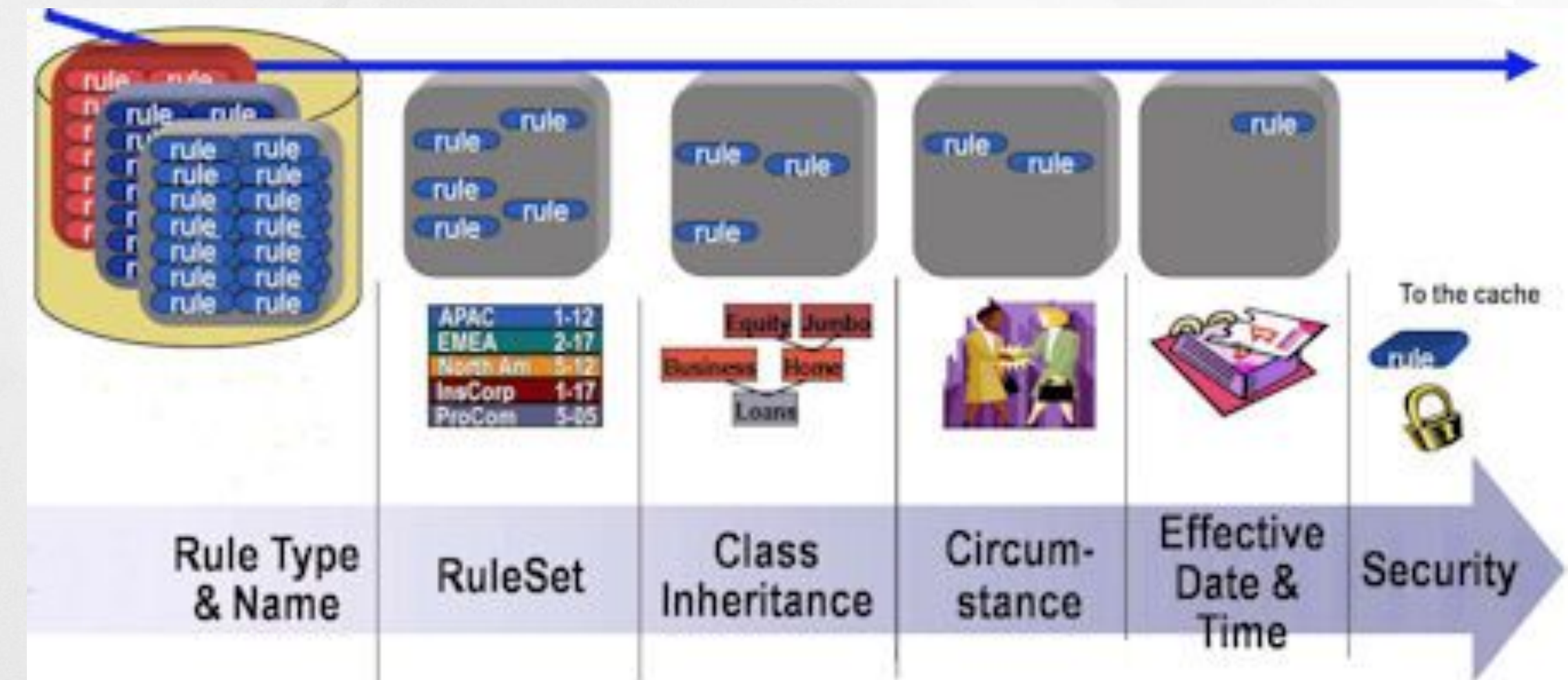
- The key parts of a rule instance being sought, such as its **Applies To class and name**
- The **user's RuleSet list**, assembled when the user logs in
- The **class hierarchy** — The structure of parent classes and subclasses below the ultimate base class
- The **user's access roles and privileges** held, determined by the access group
- Security and access control rules such as **Access of Role to Object** rules and **Privileges**
- **Rule availability** — Which rules are available, blocked, final, withdrawn or not available
- Whether **time and date limitations** affect which rules are available for this session
- In some cases, the **value of a circumstance property**

### Outputs

- The output of the rule resolution is the FIRST rule found that matches all the criteria

## Steps of rule resolution

- Step 1: Check the Cache. If the rule is there, go to Step 8.
- Step 2: Choose instances with the correct purpose
- Step 3: Discard rules where Availability = No
- Step 4: Discard inapplicable RuleSets and Versions
- Step 5: Discard all candidates not defined in a class in the 'ancestor tree'
- Step 6: Rank remaining candidates
- Step 7: Set the cache
- Step 8: Find the best instance and check for duplicates
- Step 9: Check Availability is not Blocked
- Step 10: Verify requestor is authorized to see the rule



## Step 1: Check the Cache

- An in-memory rule cache helps the rule resolution process operate faster. If the system finds an instance (or instances) of the rule in question in the cache, it accepts what is in the cache as the candidate rules and skips many steps in the resolution process.

Using rules already in the cache avoids additional database lookups.

## Step 2: Choose instances with the correct purpose

The **purpose** combines all the key properties of a rule, excluding the "defined on" class.

For an activity rule, the key properties include:

- the *Applies To class* that the activity is defined on
- the activity *name*



*The purpose in this case is the **activity name**.*

## Step 2: Choose instances with the correct purpose

For a **Field Value**, the key properties include:

- the Applies To class, as above
- the Field Name
- the Field Value



*The purpose in this case is the **Field Name plus the Field Value**, for example "pyActionPrompt.ViewHistory".*



## Step 3: Discard rules where Availability = No

The system drops unavailable rules from the temporary list.

The valid values for Availability are:

- Yes
- No
- Blocked
- Final
- Withdrawn

## Availability = Yes

If the Availability of a rule is marked as Yes, then that rule may be used for all processing – work, editing, saving, etc.

You may copy and save this rule into any open RuleSet.

## Availability = Final

Use the *Availability* value of **Final** to signal that this rule is the final version and should not be changed. If the *Availability* of a rule is set to **Final**, that means that you may use this version of the Rule, but you cannot save it into the same RuleSet, or a different RuleSet, *unless you change its name*.

You can save final rule into a higher version of the *same* RuleSet, if the higher version is not locked. This new version of the original rule also has **Final** availability.

For the purposes of rule resolution, a rule with the *Availability* value **Final** is equivalent to the value **Yes**.

## Availability = No / Draft

When a rule is marked **No/Draft**, the system treats it as though it were not present at all; the system ignores the rule and uses other rules.

The **Not Available** label applies to one rule only.

When the system is gathering all the rules to consider for a rule resolution, any rule marked **Not Available** is simply discarded.

## Exercise: Availability - No / Draft



As an example, suppose there is an activity called **Display**, which is defined on classes at several levels in the database:

- Acme-Auto-ClaimsEntry-Accident
- Acme-Auto-ClaimsEntry.DISPLAY
- Acme-Auto
- Acme-.DISPLAY

Which activity will be chosen if the system needs the activity **Display** on the class Acme-Auto-ClaimsEntry-Accident?

### Answer

Acme-Auto-ClaimsEntry-Accident

Acme-Auto-ClaimsEntry.**DISPLAY**      (*chosen rule*)

## Exercise: Availability - No / Draft



The same example:

- Acme-Auto-ClaimsEntry-Accident
- Acme-Auto-ClaimsEntry.DISPLAY
- Acme-Auto
- Acme-.DISPLAY

What should happen if Acme-Auto-ClaimsEntry.DISPLAY will be set to have availability of **No/Draft**?

### Answer

Acme-Auto-ClaimsEntry-Accident

Acme-Auto-ClaimsEntry.DISPLAY

**NOT AVAILABLE**

Acme-Auto-

Acme-.**DISPLAY** (*chosen rule*)

## Availability - Blocked

Unlike rules marked “No,” rules marked Blocked are still considered by rule resolution. If the rule resolution process completes, and the one rule that is the final result of that process is marked Blocked, then the process returns “*not found.*”

### Example

Let’s use rules from our previous example:

- Acme-Auto-ClaimsEntry-Accident
- Acme-Auto-ClaimsEntry.**DISPLAY** **BLOCKED**
- Acme-Auto
- Acme-**DISPLAY**

Which rule will be returned if we attempt to call activity DISPLAY from the class Acme-Auto-ClaimsEntry-Accident?



### Answer

Since *Acme-Auto-ClaimsEntry.DISPLAY* is the most accurate Rule for the purpose, it will be chosen; but since it is marked **BLOCKED**, the process *stops* and returns “*not found.*”

## Blocked vs No

When the system is determining which rule to use, the Availability values of **BLOCKED vs. NO** will change what rule is chosen.

If there were two versions of the activity DISPLAY:

- Acme-.DISPLAY                      AcmeCo:05-02
- Acme-.DISPLAY                      AcmeCo:05-01

What will be if the 05-02 version of the rule were set to NO?

### Answer

The system would use the 05-01 version

What will be if the 05-02 version of the rule were set to BLOCKED?

### Answer

The system would *not* use the 05-01 version, but would return “*not found.*”





## Availability = Withdrawn

Occasionally, you need to delete a rule from an application.

If the incorrect rule is stored in an unlocked RuleSet version, you can easily delete it.

However, if the application has been moved to other systems or the rule is in a locked RuleSet version, it is no longer possible to change or delete that rule.

Instead, you can withdraw the rule.

## Step 4: Discard inapplicable RuleSets and Versions

The system drops from the list rules that belong to RuleSets and Versions that are not enabled for the current requestor.

For instance, if the user's profile includes the RuleSet version

**ThisRuleSet: 05-01,**

rules belonging to **ThisRuleSet: 04-** or **ThisRuleSet: 06-** are *dropped*.

## Step 5: Discard all candidates not defined in a class in the ancestor tree

Only rules found in classes from which the current class descends by either pattern or direct inheritance will be retained in the list.

## Step 6: Rank remaining candidates

1. The system ranks the remaining rules on the list in order of Override (rules stored in an override RuleSet. These rules always move to the top of the ranking.)
  - Class
  - RuleSet
  - Version
  - Circumstance
  - Circumstance Date
  - Date/Time Range
2. After that system removes all candidates with availability = Withdrawn
3. The last sub-step in the ranking phase of the rule resolution algorithm determines the *default* rule candidate. A default rule candidate is the first rule candidate (highest ranked) that has no qualifiers. This default rule candidate is the last possible rule to be executed as it always matches any additional requests for this rule. Additional rule candidates ranked below the default rule candidate are discarded.

## Step 7: Set the cache

The process adds the rules that remain on the list to the cache as being selectable for use.

## Step 8: Find the best instance and check for duplicates

The process searches down the list for the first rule that:

- exactly matches a Circumstanced Rule
- has the correct Circumstanced Date
- is in the correct Date/Time Range
- or is the default Rule, with no qualifiers

When it finds a rule that matches these conditions, the process checks whether the next rule in the list is equally correct. If it is, the process sends a message that there are duplicate rules in the system and stops processing.

If no duplicates are found, the system prepares to use the rule that matched the listed conditions.

## Step 9: Check Availability is not Blocked

The process checks Availability again, to see whether it is set to Blocked for this rule. If it is, the system sends a message that it could not find an appropriate rule to use.

## Step 10: Verify requestor is authorized to see the rule.

The process finally verifies that the requestor has authorization to access the selected rule. If so, the system uses it. If not, it sends a message that it could not find an appropriate rule to use.