



This PDF is generated from authoritative online content, and is provided for convenience only. This PDF cannot be used for legal purposes. For authoritative understanding of what is and is not supported, always use the online content. To copy code samples, always use the online content.

GRS Best Practice Guide

New Features by Release

12/20/2025

New Features by Release

New in 8.5.3

Role-Based Access Control (8.5.302)

Support for Role-Based Access Control at the Rules Package Level

Important

GRS requires Genesys Administrator 8.1.305.04 (minimum) for configuring package level permissions.

You can expand the graphics by clicking on them.

Background

Previously, GRAT used Configuration Server Roles to provide only global access control to all packages in a given node of the business hierarchy. The privileges, like **Modify Rule Package**, **Delete Rule Package**, **Modify Rule**, **Delete Rule** and so on, are granted to users via roles. With this approach, if a user is granted the **Modify Rule Package** (for example) privilege, then they can modify all the rule packages defined in a node of the GRAT business hierarchy.

Release 8.5.302 now provides package-level overrides to these global roles—role privileges can be restricted to specific rule packages by applying Role-Based Access Control at the rule package level. The new Rule Package Level Roles (roles created specifically for use with rule packages only) can be mapped to rule packages to override the global-level roles. These Rule Package Level Roles will have no effect if not mapped to a rule package.

New Role Permission—View Rule Package

View access for specific rule packages can now be controlled by using the new role permission **View Rule Package**. The new permission is applicable to only the rule package level.

Existing Role Permissions

All of the existing role permissions except **Create Rule Package** and template-related permissions are applicable at the rule package level too.

Example

In 8.5.302 you can now assign role permissions at both global/node level and at rule-package level to achieve the following outcome:

- Department A
 - Rule package 1
 - Rule package 2
 - Sales
 - Rule package 3
 - Department B
 - Rule package 4
-
- User A—Can see Department A but not Department B
 - User B—Can see Department B but not Department A
 - User C—Can see rule package 1, but rule package 2 is hidden
-

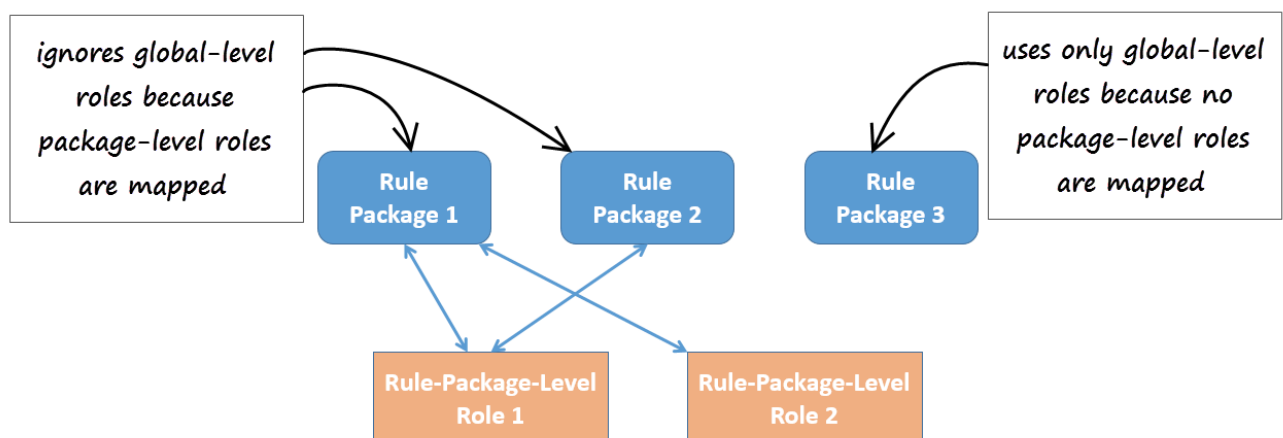
Location

To distinguish these new roles from global-level roles, they are placed in a new folder:

[Tenant] > Roles > GRS Rule Package Level Roles

Package-Level Overrides

Where package-level roles are mapped to a rule package, they override global-level roles.

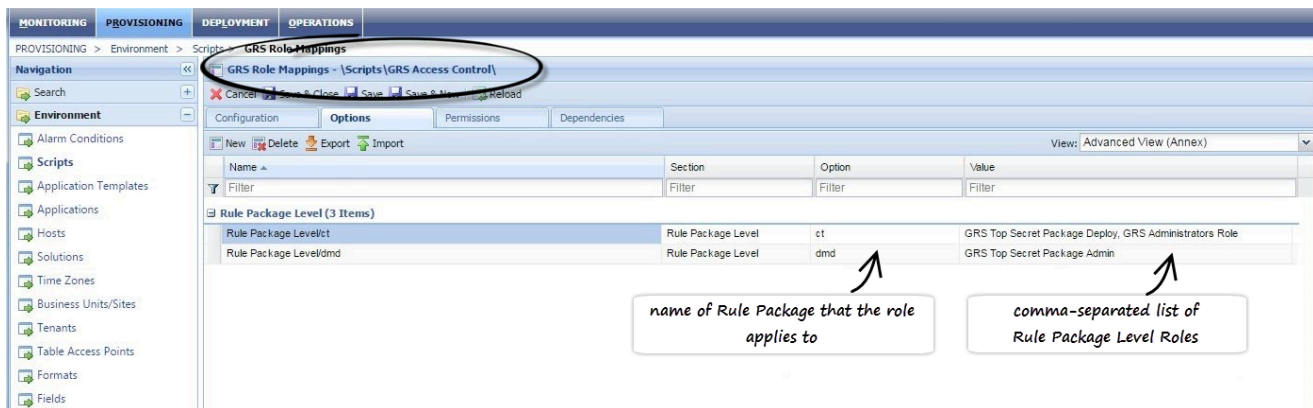


Managing the Mapping of Roles

The mapping of the rule packages to Rule Package Level Roles is managed in Genesys Administrator or Genesys Administrator Extensions, in the options under section **Rule Package Level** of the **\Scripts\GRS Access Control\GRS Role Mappings** script. The example below is from Genesys Administrator.

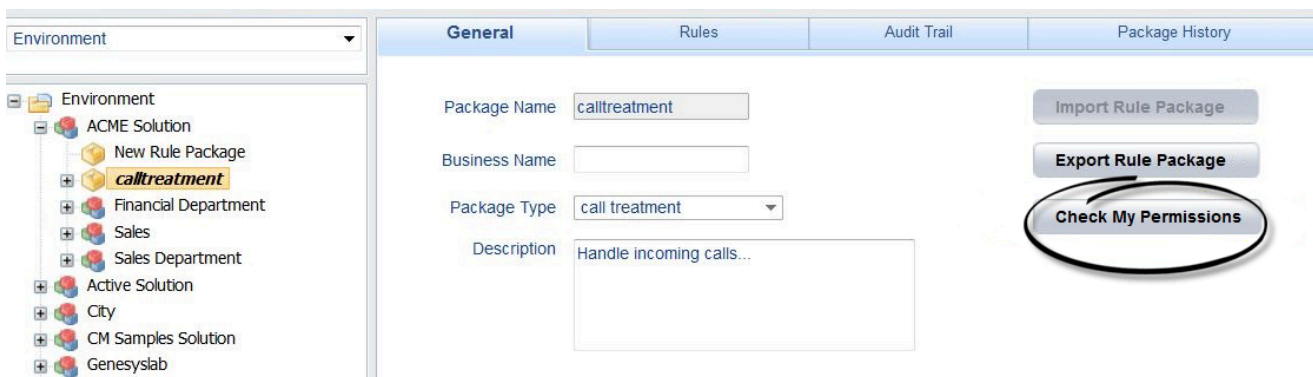
Important

Because the delimiter in the list of roles is a comma, you can't use commas in the names of any role.



Viewing GRAT User Permissions

To enable GRAT users to view their current list of permissions, a **Check My Permissions** button is now also available at the rule-package level and shows the permissions at selected package level.



Support for Deployment to GWE Clusters (8.5.302)

In the case of GRE (Genesys Rules Engine) cluster deployment, GRAT gets the GRE nodes' information from the cluster and deploys the rule package individually to each of the cluster nodes.

But a GWE (Genesys Web Engagement) cluster is different—GWE cluster nodes are not available as connections. So the deployment must be targeted to the cluster host itself—that is, a host in the Configuration Server information of the GWE cluster. See **the Deployment Guide** for more detail.

Support for GRAT Clustering (improved in 8.5.302)

Background

Before release 8.5.301, Genesys recommended maintaining a cold-standby backup GRAT server, connected to the same database repository as the primary, which could be initialized in the event of a primary GRAT server failure or disconnection.

A configuration option called `clear-repository-cache` could be set to force GRAT to clear and rebuild the local cache/search indices on startup. This allowed the backup server to synchronize with the repository, even if it had been shut down for months. Depending on the number of rule packages deployed, synchronization might be fast or slow.

What's New?

In release 8.5.301, you can now configure clusters of GRAT servers, which deliver much greater resilience and availability, enabling almost instant switchovers between GRAT nodes that are members of the cluster. All cluster members connect to the same database repository.

No single GRAT node is considered primary—they are all equal partners in the n-node cluster.

Each node maintains a journal of changes, which resides on a separate repository database table. Nodes can periodically poll the database to ensure that they mirror updates made on the other nodes. You can also add new nodes to the cluster and synchronize them with their peers.

- **New GRAT Cluster Application Template**

The GRAT cluster configuration object must be based on a new application template—`GRAT_Rules_Authoring_Tool_Application_Cluster_<version>.apd`—that is delivered with this release. GRAT instances become members of the cluster when their Application object is added to the **Connections** tab in the cluster's Application object.

- **High Availability**

An n-node cluster configuration can be used to deliver High Availability—for example, if a customer wants to use one GRAT server as the primary server and another as the secondary/backup warm-standby server, a load balancer could be configured to send all traffic to the

primary server. The warm-standby instance periodically polls the journal table and keeps its cache/index files synchronized. In the event of a failure (or planned maintenance) of the primary GRAT server, the load balancer switches and sends traffic to the hot-standby secondary/backup GRAT server. The user would have to log in again, and any unsaved changes from their old session would be lost. However, they could resume in seconds instead of waiting for the cold-standby GRAT to become available.

- **Load Balancing**

A load balancer can evenly distribute the load across the available GRAT nodes in the cluster, or it can distribute the load based on other criteria, such as the geographic location of the browser. However, once a node has been selected, the load balancer must ensure that the same node is used for the duration of the session (session stickiness).

The load balancer must provide session stickiness for the GRAT user interface requests by sending all the requests pertaining to a session to the same GRAT node that initiated the session after successful login. Similarly, in the case of the GRAT REST API, after successful login, the load balancer must send the subsequent requests to the same GRAT node that handled the login request. More information on the GRAT REST API Authentication mechanism is available [here](#).

- **Scheduled Deployments of Rules Packages**

Scheduled deployments of rules packages can now be viewed, edited and cancelled by GRAT users on GRAT instances that did not originally perform the scheduling—previously, only the original GRAT user/instance that performed the scheduling could do this. Any GRAT instance that modifies the scheduled deployment takes over the responsibility for handling that deployment. The deployment history, including a new **Deployed From** field that indicates which node last scheduled the deployment, is now available to all the nodes in the cluster (visible in the **Deployment History** tab).

Deployment History											
Deployment ID	Package Name	Package Version	Snapshot Name	Location	Deployed From	Deployed By	Status	Deployed/Scheduled	Comments	Scheduled	
145329748303	acceptance.tes	40	LATEST	GRE_8530100	GRAT_853010	demo	Successful	20-Jan-2016 15:44			
145329740548	acceptance.tes	36		GRE_8530100	GRAT_853010	demo	Successful	20-Jan-2016 15:43			
145329737619	acceptance.tes	35	LATEST	GRE_8530100	GRAT_853010	demo	Successful	20-Jan-2016 15:43			
145329427965	acceptance.tes	27	LATEST	GRE_8530100	GRAT_853010	demo	Successful	20-Jan-2016 14:51			
145329348766	acceptance.tes	17	LATEST	GRE_8530100	GRAT_853010	demo	Successful	20-Jan-2016 14:38	First deploy for acceptance test		

Important

The new **Deployed From** field is also visible in standalone GRAT instances—it will display the ID of the standalone GRAT instance.

Deploying the GRAT Clustering Feature

1. Import the `GRAT_Rules_Authoring_Tool_Application_Cluster_<version>.apd` template into your Genesys configuration environment.
2. Create a GRAT cluster Application object based on the new template. Adjust the configuration options as needed.
3. For each GRAT node to be added to the cluster, add its Application ID as a connection in the cluster's Application object.
4. Add the DAP (Database Access Point) Application object to be used by the GRAT cluster as a connection in the cluster's Application object.

Important

An existing standalone GRAT database can be used.

5. If you are re-using an existing standalone GRAT instance in the cluster, re-deploy the GRAT application to ensure that its local cache is re-initialized.

Important

- Any change in the cluster configuration will only take effect upon re-start of the GRAT servers.
- For high availability, GRAT instances must have a high-speed connection to the database. Slow connections may result in the type of issues commonly seen when the local repository cache is corrupted.

Removing a GRAT Node from the Cluster

1. Remove the GRAT node's Application ID from the list of connections in the cluster's Application object.
2. Manually remove the GRAT's entry in the LOCAL_REVISIONS table in the GRAT database (this is a single row, identified by the GRAT Application's DBID in the JOURNAL_ID column).

Warning

If this step is not performed, the clean-up thread (janitor) will not be effective.

Removing a GRAT Node from the Cluster (post-8.5.302)

In release 8.5.302, the manual cleanup of LOCAL_REVISIONS table (by the clean-up thread (janitor)) after node removal is not required. The cleaning task is now automated and controlled by the following new configuration options in the **settings** section of the GRAT cluster application:

- local-revisions-janitor-enabled
 - When this is enabled, at regular intervals, the clean-up task cleans the local revisions table by removing the entries for cluster nodes which are no longer part of the cluster. If this is not done then journal table Janitor will not be effective.
- local-revisions-janitor-sleep
 - Specifies the sleep time of the clean-up task in days (only useful when the clean-up task is enabled, default is 15 days)
- local-revisions-janitor-first-run-hour-of-day
 - Specifies the hour at which the clean-up task initiates its first run (default = 2, which means 02:00)

CONFIGURATION OPTION DETAIL

Support for A/B/C Split Test Scenarios

[+] FULL DETAILS

Overview

[+] DETAILS

What problem does A/B/C Split Testing solve?

If your organization has a complex set of rule packages and rules to define business decision-making,

if you want to be able to compare alternative outcomes and scenarios, then rather than just changing a condition and hoping for the best you probably want to phase changes in slowly in order to measure the impact over time.

A/B/C Split Testing (hereafter, **Split Testing**) allows you to compare the business outcomes of alternative rule scenarios before rolling out significant changes to the way you make your business decisions. With Split Testing you can make, test and review changes incrementally to test their effects before committing to a particular change set.

For example, you might want to test several subtle changes in the way applications for credit cards are treated, in order to identify which has the best business outcome. You could split test for one income level against another, or one age range against another, or between different customer segments, or indeed any conditions in your rule package.

Flexible Approaches to Split Testing

Split Testing offers several approaches. For example, you can choose to:

- Unconditionally force either path A or B or C.
- Tie Split Testing to certain conditions (whether the customer is from a particular city, of a certain age, not a Gold customer, and so on).
- Weight the paths for A, B and C by percentage. This allows you to take a more statistical approach and use larger test data sets.
- Tie Split Testing to a business calendar (for example only do split testing on Fridays after 5 PM or Saturday morning 9 AM - 12 PM).

New Split Testing Template

A new template ships with GRS 8.5.3—GRSSplitTest_template.xml—and this provides some basic Facts (Conditions and Actions) for the Split Testing feature. To implement the feature, GRAT users must import this template (from the **Samples** folder) and attach it to all rules packages for which the Split Test functionality is required.

Important

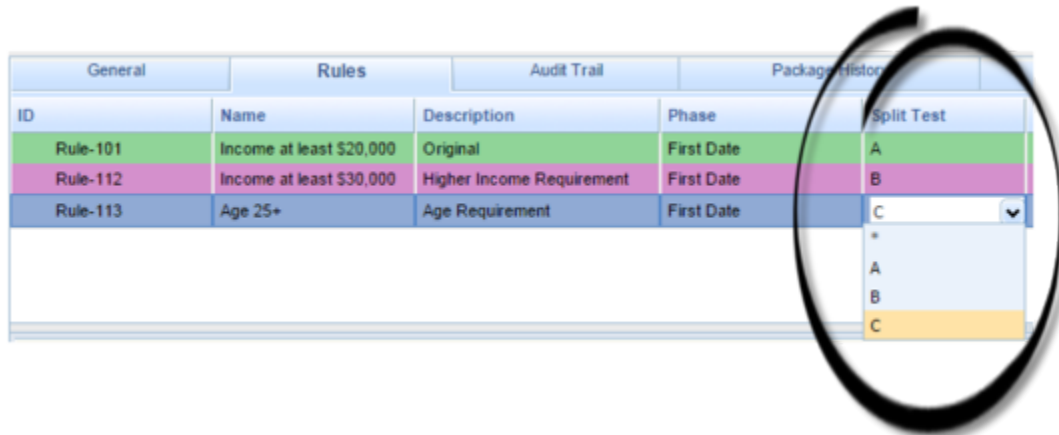
The new template—GRSSplitTest_template.xml—is shipped with type samples. This means it can only be added to rule packages of type samples, because GRAT only shows templates in the list with compatible types. To use this template with other rule package types, you can import the template from GRAT into GRDT, change the name (for example, GRSSplitTestForMyType) and the type (to match your rule package type) and publish to GRAT. Then you can use with another package type.

Changes in GRAT

[+] DETAILS

New Split Test Column in Rule Packages

A new Split Test column displays at the rule package level.



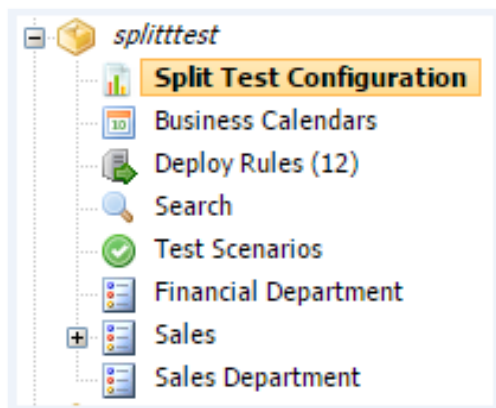
ID	Name	Description	Phase	Split Test
Rule-101	Income at least \$20,000	Original	First Date	A
Rule-112	Income at least \$30,000	Higher Income Requirement	First Date	B
Rule-113	Age 25+	Age Requirement	First Date	C

When the Split Test template has been imported into GRAT, this drop-down field will show the values imported from the template (A, B and C are the default values in the shipped template). If the template has not been imported, only the wildcard (*) symbol will be available.

When a path is selected in this column, the rows color-code for easier viewing. The wildcard symbol (*) means that the Split Test condition is ignored, so this rule will always fire.

New Split Test Configuration Node

A new node—Split Test Configuration—appears in the left navigation tree. You can use this node to create new (and display existing) Split Test rules that will apply at the rule package level. In rules created by clicking this new node, the **Split Test** column is hidden to avoid confusion.

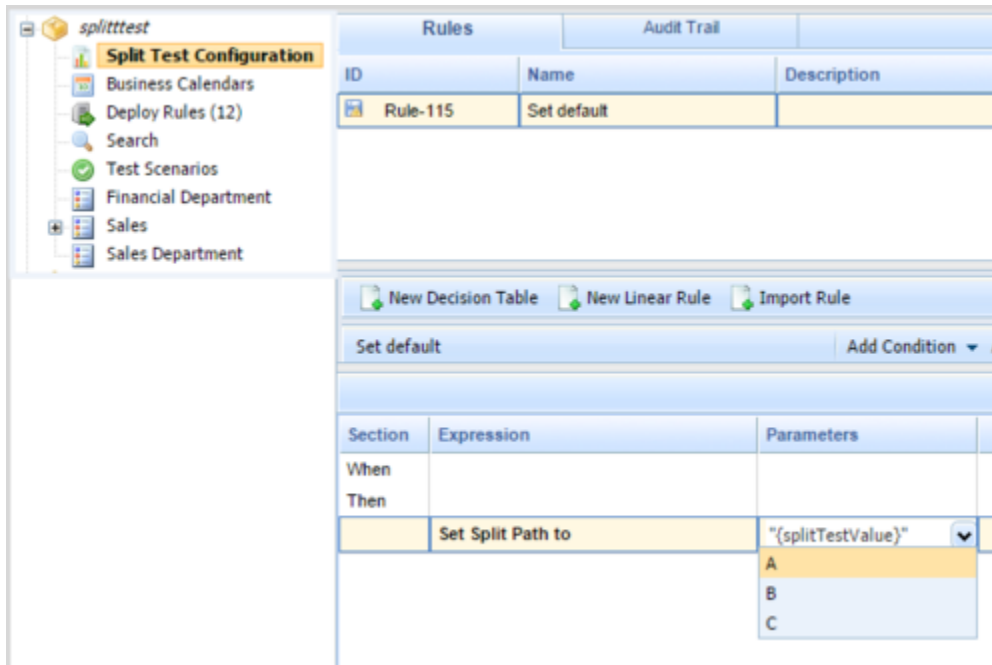


What Can I Do with Split Test Rules?

[+] Force the Split Test Path to A, B or C

Unconditionally force either path A or B or C

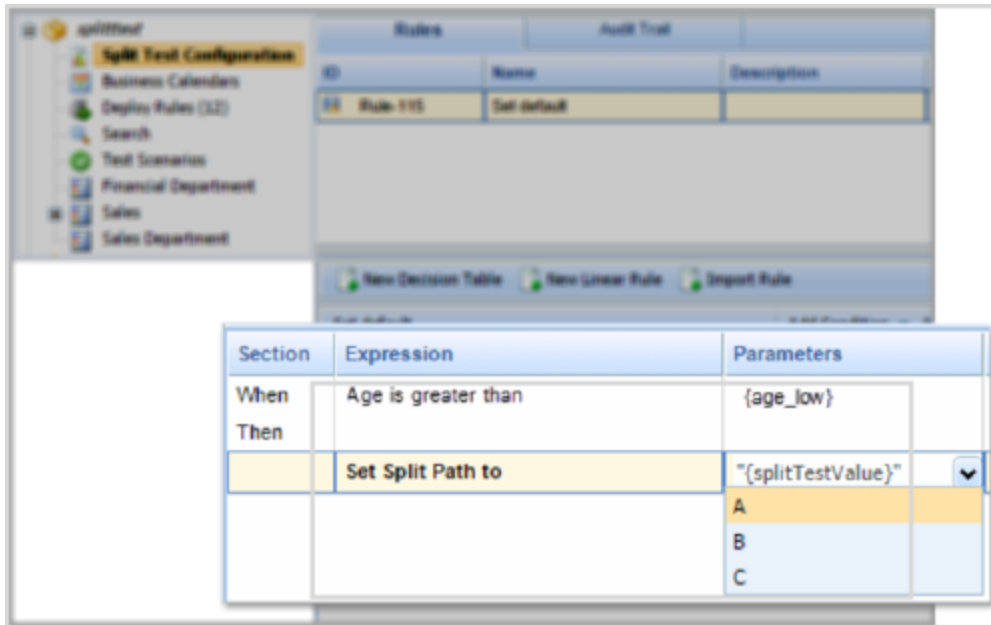
In rules created by clicking this new node, the `splitTestValue` can be selected as a Parameter for the rule. You can use this simple rule to force the Split Test path to any of the values in the template (the default shipped values in the template are A, B and C, but you can change these in the template if required).



[+] Tie a Split Test Rule to a Condition

Tie Split Testing to a Condition

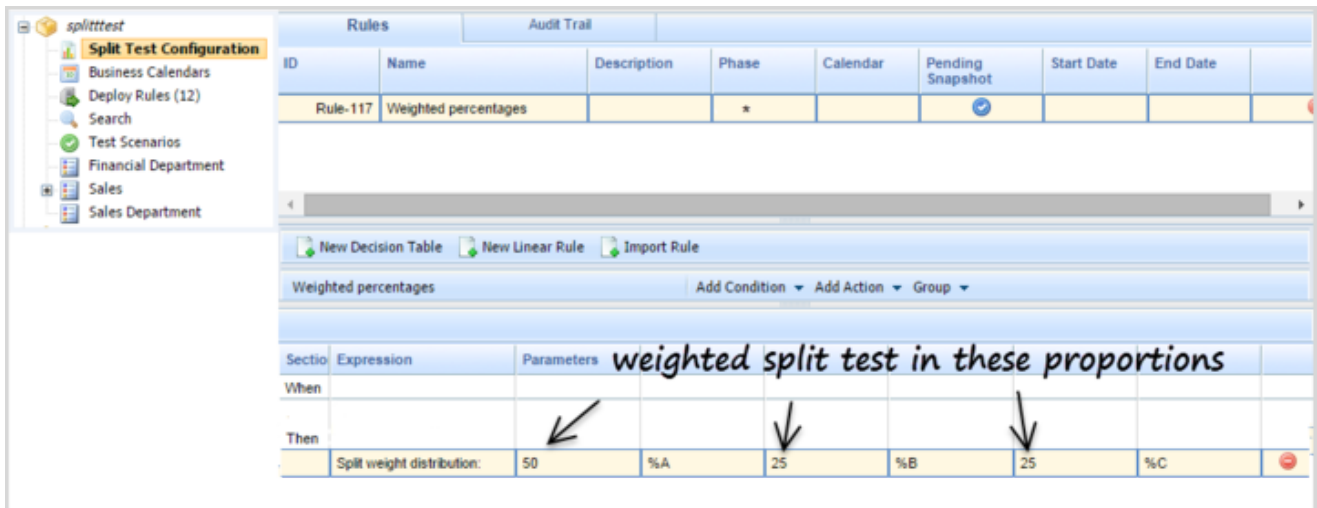
To add more flexibility and precision to your use of Split Test rules, you can add to your rule any other condition available from the template(s) attached to the rule package. The example below has a lower age limit:



[+] Set Up a Weighted Distribution

Weight the paths for A, B and C by percentage

You could add a condition to perform Weighted Distribution Split Testing. In this example the Split Test rule distributes the incoming rule evaluation requests across the A, B and C paths in the percentage proportions 50/25/25. This should look something like this:



[+] Tie a Split Test Rule to a Business Calendar

Tie Split Testing to a Business Calendar

You can also tie Split Testing to a Business Calendar. For example, you might prefer to keep Split Testing outside normal business hours, or maybe run Split Testing in response to specific customer offers or other commercial events. To do this you can either use an existing Business Calendar or create a new one for your specific purpose. You then add this calendar to the rule to which you want to apply Split Testing.

The screenshot shows two panels. The top panel, titled 'Business Calendars', contains a table with the following data:

ID	Name	Week Starts on	Week Ends on	Start Time	End Time	Time Zone
Calendar_140	Split Testing	Saturday	Saturday	8:00 AM	12:00 PM	(UTC-4) Eastern Standard T

The bottom panel, titled 'Split Test Configuration', shows a 'Rules' table with the following data:

ID	Name	Description	Phase	Calendar	Per Sn
Rule-117	Weighted percentages		First Date	Split Testin	

An arrow points from the 'Split Testin' dropdown in the 'Calendar' column of Rule-117 to the 'Split Testing' entry in the 'Business Calendars' table.

So, when the parameters set for the Business Calendar apply, the weighted Split Testing begins and ends automatically.

The screenshot shows the 'Rules' configuration for Rule-117. The 'Calendar' column is circled, and a handwritten note says 'when calendar constraints apply...'. Below the table, there is a section for 'Weighted percentages' with a table showing the split weight distribution:

Section	Expression	Weight	Label	Weight	Label	Weight	Label
When	Today is a working day						
Then	It is normal business hours						
	Split weight distribution:	50	%A	25	%B	25	%C

Handwritten notes include '...perform weighted split test in these proportions' with arrows pointing to the weights 50, 25, and 25.

Using Test Scenarios to Check Your Split Test Logic

[+] Using Test Scenarios to Check Your Split Test Logic

GRAT's Test Scenario feature allows you to run pre-deployment checks on the logic of any rule package that you want to deploy. Because A/B/C Split Testing enables an additional level of logic to be built into a rule package, it's advisable to run more complex Split Test logic through a Test Scenario before deploying it to a production rules engine.

For example, if you wanted to check the Weighted Distribution example above before deploying the rule package to a production engine, you could create a test scenario to have 10 rows of identical data that only qualify when running path **A**. Each row in the test scenario simulates a separate rule evaluation and will therefore apply the Split Test Configuration rules first. If we see a green tick, then the original **A** path was taken. If we see a red cross, then either the **B** or **C** path was taken.

Because we configured the **A** path for 50% of the time, we should see about 50% green ticks every time we hit the **Run Test Scenario** button. Here are the outcomes of running the test scenario twice:

ID	Name	Results	Income	age	offerCreditCard
TSR-127		✓	20000	20	✓
TSR-139		✓	20000	20	✓
TSR-138		✓	20000	20	✓
TSR-137		✓	20000	20	✓
TSR-136		✗	20000	20	✗
TSR-135		✗	20000	20	✗
TSR-134		✗	20000	20	✗
TSR-133		✓	20000	20	✓
TSR-132		✗	20000	20	✗
TSR-131		✗	20000	20	✗

ID	Name	Results	Income	age
TSR-127		✗	20000	20
TSR-139		✗	20000	20
TSR-138		✓	20000	20
TSR-137		✓	20000	20
TSR-136		✗	20000	20
TSR-135		✓	20000	20
TSR-134		✓	20000	20
TSR-133		✗	20000	20
TSR-132		✓	20000	20
TSR-131		✓	20000	20

After testing that the A/B/C split logic is correct, you can deploy your rule package. At this point, the A/B/C split logic will run as you have it configured during normal rule evaluation. To adjust the A/B/C configuration (for example, make the **A** path 25% instead of 50% and the **B** path 75% instead of 50% and so on), you must make the changes in GRAT and then redeploy the rule package.

Allowing an Application to Override Split Test Configuration Rules

[+] Allowing an Application to Override Split Test Configuration Rules

In addition to setting the A/B/C path in the Split Test Configuration section, it is possible for the invoking application (GVP, IVR, ORS and so on) to set the value. This can either be used as a default, or can be used to override the logic in the Split Test Configuration.

If you want the invoking application to be able to override the normal logic in Split Test Configuration, then you can simply add the new `Split Test Path is not set` condition to your rule.

Section	Expression	Parameters
When	Split test path is not set	
Then	Set Split Path to	A

This new condition will check whether the value has already been passed in by the invoking application. If it has been passed in, then the Split Test logic will be bypassed. If not, then the normal Split Test rule logic will apply.

New in 8.5.2

Support for GRE Clustering

Background

Before release 8.5.2, successful deployment to a GRE cluster required a successful deployment to every node in the cluster, otherwise the deployment was rolled back and none of the nodes was updated.

What's New?

- **Partial deployments**—The deployment process can now handle scenarios in which nodes are down or disconnected. GRAT continues deploying directly to the clustered GREs, but now the deployment continues even if it fails on one (or more) of the cluster nodes. A new deployment status—`Partial`—will be used for such deployments. Users will see the failed/successful deployment status for each node by clicking on the status in GRAT deployment history.

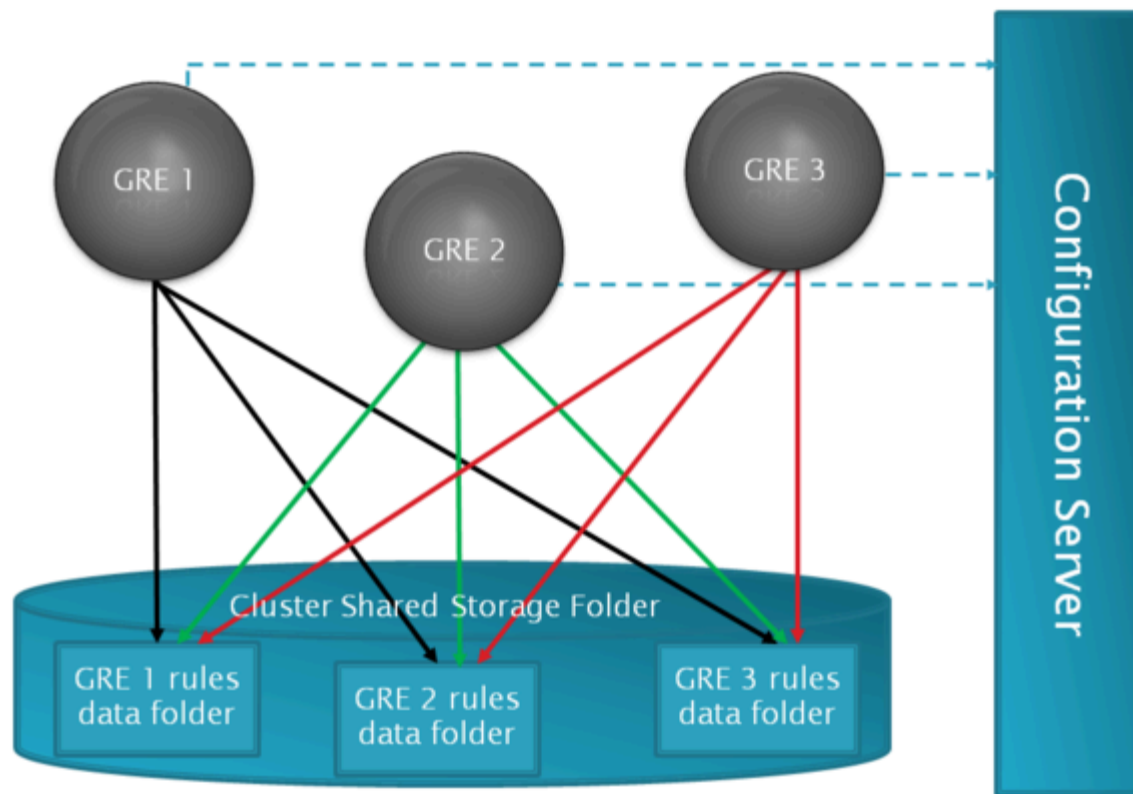
[+] CONFIG OPTION

- `allow-partial-cluster-deployment`
 - Default value—`false`
 - Valid Values—`true`, `false`
 - Change Takes Effect—Immediately
 - If set to `true` it enables GRAT to perform a partial deployment for a cluster, otherwise the old

behavior (false) of failing the entire cluster deployment if any single node fails.

- **A new "smart cluster" Application template**—A new Application template—`GRE_Rules_Engine_Application_Cluster_<version>.apd`—is implemented to support the new functionality. To configure a cluster with the new features, use this template. Members of the cluster must be of the same type (Genesys Rules Engine applications—the new features are not applicable to Web Engagement engines) and must have minimum version numbers of 8.5.2. Genesys recommends not creating clusters of GREs with mixed 8.5.1/8.5.2 versions. A new shared deployment folder from which rule packages can be synchronized can also be defined. When the cluster is configured to auto-synchronize, the GREs will auto-synchronize when newer rule packages are detected in the shared deployment folder. Auto-synchronization is enabled or disabled using configuration options in the `GRE_Rules_Engine_Application_Cluster` object in the Genesys configuration environment.
- **Auto-synchronization of cluster nodes**—Newly provisioned nodes in the cluster, or nodes that have disconnected and reconnected, can be auto-synchronized with other nodes in the cluster. For a clustered GRE:
 - Where the cluster has the new option `auto-synch-rules` set to `true` (new option), a cluster shared folder is now used to store rules package data. Each clustered GRE node has its own deployment folder in the cluster shared folder. The shared folder will enable synchronization of the cluster GREs after network or connection disruption or when a new GRE is added to the cluster.
 - Where the cluster has the new option `auto-synch-rules` set to `false` (default), the deployed rules files will be stored in the location defined in `deployed-rules-directory`. In such cases a manual redeployment will be required if deployment status is partial or if a new node is joining the cluster.

[+] **DIAGRAM** `auto-synch-rules=true`



[+] CONFIG OPTIONS

- `auto-synch-rules` (set on cluster application)
 - Default Value—`false`
 - Valid Values—`true`, `false`
 - Change Takes Effect—At GRE (re-)start
 - Description—Set this to `true` to enable a GRE in cluster to start the periodic auto-synch and auto-deployment process. When this value is set to `true`, options `shared-root-directory` must be provided and `deployed-rules-directory-is-relative-to-shared-root` must be set to `true` for auto-synch to work.
- `auto-synch-rules-interval` (set on cluster application)
 - Default value—5
 - Valid Values—Integer
 - Change Takes Effect—At GRE (re-)start
 - Description—The interval in minutes between the end of the last synchronization check/auto deployment and the start of a new synchronization check.
- `synch-rules-at-startup` (set on cluster application)

- Default value—false
 - Valid Values—true/false
 - Change Takes Effect—At GRE (re-)start
 - Description—Set this option to true to have the GREs synchronize and deploy rules at startup. It is ignored if auto-synch-rules is set to true (that is, when auto-synch-rules is true then auto-synch is always performed at startup. This is useful if rules synchronization is required only at startup when auto-synch-rules is set to false.
- shared-root-directory
 - Default value—No default
 - Valid Values—String
 - Change Takes Effect—Immediately
 - Description—Specifies the shared root directory. When this option is used and option deployed-rules-directory-is-relative-to-shared-root is set to true, the effective deployed rules directory used by GRE is made by prepending this string to the path specified in deployed-rules-directory. It can be used to specify the mapped path to the shared location used for the auto-synch feature for rules. Having this option empty (or not set) effectively allows setting an absolute path in the deployed-rules-directory even when deployed-rules-directory-is-relative-to-shared-root is set to true. It may be a mapped folder backed by a service like Amazon S3 or simply an OS shared folder. Examples:
 - If shared-root-directory = C:\shared and deployed-rules-directory = \GRE1, then the effective deployed rules directory path used by GRE is C:\shared\GRE1 .
 - If shared-root-directory = \\10.10.0.11\shared and deployed-rules-directory = \GRE1, then the effective deployed rules directory path used by GRE is \\10.10.0.11\shared\GRE1 .
 - If the shared folder is mapped on drive Z, the shared-root-directory will be Z:, deployed-rules-directory may be \GRE1, then the effective deployed rules directory path used by GRE will be Z:\GRE1.
 - deployed-rules-directory-is-relative-to-shared-root
 - Default value—false
 - Valid Values—true, false
 - Change Takes Effect—Immediately
 - Description—Indicates whether to use the shared root directory as the root directory for the deployed-rules-directory (true). If GRE belongs to a cluster that has auto-synch-rules or just synch-rules-at-startup enabled then this option must be set to true so that GRE can participate in the auto-synch process. This can be used even when GRE does not belong to a cluster. If this option is set to false, auto-synch will not work.

[+] CONFIGURING SHARED FOLDERS

You must provide access to enable GRE to read and write to its own deployment folder and read data from the other node's deployment folders. If the GRE deployment folder does not exist, GRE will try to create it at start-up. When the new option deployed-rules-directory-is-relative-from-shared-root is enabled, the value provided for deployed-rules-directory is considered relative to the value of the shared root directory value.

For example, if the deployed-rules-directory-is-relative-from-shared-root option is true:

- If the value of the shared-root-directory option is **/shared/cluster-A** and the value of the

deployed-rules-directory option is **/foo/GRE_1** then GRE will try to use **/shared/cluster-A/foo/GRE_1** as the deployed rules directory.

- If the value of the shared-root-directory option is "" (that is, empty or not set) and deployed-rules-directory option is **/foo/GRE_1**, then GRE will try to use **/foo/GRE_1** as the deployed rules directory.

If required, for example in cloud deployments, Customer/Professional Services must make sure that the shared folder are set up in HA mode.

Folder Sharing Schema

Below is an example of how clustered GREs see other GRE node's deployed rules folders. In the example, below **/sharedOnGre1**, **/sharedOnGre2** and **/sharedOnGre3** all are pointing to the same shared folder, but the shared folder is mapped/mounted differently on each machine.

GRE1

```
shared-root-directory = /sharedOnGre1
deployed-rules-directory = /GRE1_DEPLOYDIR
```

GRE2

```
shared-root-directory = /sharedOnGre2
deployed-rules-directory = /GRE2_DEPLOYDIR
```

GRE3

```
shared-root-directory = /sharedOnGre3
deployed-rules-directory = /GRE3_DEPLOYDIR
```

GRE1 will see other GREs (GRE2 and GRE3) deployed rules folder by using paths as below:

```
GRE2    /sharedOnGre1/GRE2_DEPLOYDIR
GRE3    /sharedOnGre1/GRE3_DEPLOYDIR
```

GRE2 will see other GREs (GRE1 and GRE3) deployed rules folder by using paths as below:

```
GRE1    /sharedOnGre2/GRE1_DEPLOYDIR
GRE3    /sharedOnGre2/GRE3_DEPLOYDIR
```

GRE3 will see other GREs (GRE1 and GRE2) deployed rules folder by using paths as below:

```
GRE1    /sharedOnGre3/GRE1_DEPLOYDIR
GRE2    /sharedOnGre3/GRE2_DEPLOYDIR
```

[+] CONFIGURATION STEPS

To set up the auto-synchronization feature, do the following:

1. Shut down the clustered GREs (which must have been created using the GRE_Rules_Engine_Cluster application template).
2. Set the following configuration options:
 - a. On the GRAT Application object, set `allow-partial-cluster-deployment` to `true`.
 - b. On the new cluster Application object, set `auto-synch-rules` to `true`. You can optionally set `auto-synch-rules-interval` if you require a value that is different value from the default. You can also optionally set `synch-rules-at-startup`—this is useful only when `auto-synch-rules` is set to `false`.
 - c. On each GRE in the cluster, set `deployed-rules-directory-is-relative-to-shared-root` to

true. Set the value of `shared-root-directory` per the description above.

3. Make sure that in each GRE, the concatenated path (the effective deployed rules directory path) `shared-root-directory` PLUS `deployed-rules-directory` makes a valid directory path. If the directory does not already exist, it will be created at GRE start-up. See [Cluster Shared Storage Folder](#) for more details.
 4. Ensure that the clustered GREs have appropriate access rights to create/read files or folders and start them.
 5. If you are migrating from a pre-8.5.2 release, re-deploy each rule package in order for auto-synchronization to work. See [Configuration Notes](#) below for details.
-

Configuration Notes

If GRAT's CME Application ID is replaced (such as in the scenario in [Important](#) below), you must do one of the following for auto-synchronization to work correctly. Either:

- Redeploy all the rule packages to the cluster; or
- Update the configuration—this may be preferable to redeploying all rule packages (for example, because of a large number of rule packages)

Important

Changing of GRAT's configuration Application ID will occur when you have a previous configuration using GRAT 8.5.1 with deployed rule packages and you upgrade to GRAT 8.5.2, and as part of that, create new application objects in CME for GRAT 8.5.2.

Redeploy all the rule packages to the cluster

If auto-synchronization is enabled and deployment to the cluster cannot be performed, follow the steps below to deploy to the GREs individually:

- a. Temporarily disable auto-synchronization in the GREs by setting option `deployed-rules-directory-is-relative-to-shared-root` to false.
- b. Redeploy all the rule packages to the GREs.
- c. Once the rule packages have been deployed to all the GREs, reset `deployed-rules-directory-is-relative-to-shared-root` to true.

If auto-synchronization is disabled and deployment to the cluster cannot be performed, the rule packages can be deployed to all the GREs individually without requiring any additional settings.

Update the configuration

In the Tenant configuration, update option `next-id`, which is available under the Annex settings section in a `Script Schedule-XXXX` (where XXXX is GRAT's configuration Application ID) corresponding to the new GRAT Application, with the value from script corresponding to the previous GRAT Application.

Option path in Configuration Manager:

Configuration > [Tenant Name] > Scripts > Rule Deployment History > Schedule-[Id of GRAT

App] > Annex > settings > "next-id"

Option path in Genesys Administrator:

PROVISIONING > [Tenant Name] > Scripts > Rule Deployment History > Schedule-[Id of GRAT App] > Options (with Advanced View (Annex)) > settings > "next-id"

Example

If the Tenant name is Environment, the new GRAT configuration Application ID is 456 and the old GRAT configuration Application ID is 123.

Using Configuration Manager:

Copy the value of option:

Configuration > Environment > Scripts > Rule Deployment History > Schedule-123 > Annex > settings > next-id

into:

Configuration > Environment > Scripts > Rule Deployment History > Schedule-456 > Annex > settings > next-id

Using Genesys Administrator:

Copy the value of option:

Configuration > Environment > Scripts > Rule Deployment History > Schedule-123 > Options (with Advanced View (Annex)) > settings > next-id

into:

Configuration > Environment > Scripts > Rule Deployment History > Schedule-456 > Options (with Advanced View (Annex)) > settings > next-id

Limitations in the Initial 8.5.2 Release

- The auto-synchronization feature does not include undeploy functionality.
 - A GRE cannot be a member of more than one cluster. This is because GRE checks all the clusters in the Genesys configuration environment to see which one has a connection to the GRE. If there are multiple such clusters, only the first one found is considered; any others are ignored.
 - GRE can operate either singly or as part of a "smart cluster", but not both.
 - High Availability (HA) for the cluster shared folder is not currently implemented. If HA is required, for example in multi-site deployments, Professional Services must make sure that the shared folder is set up in HA mode.
-
-

Support for WebSphere Clustering

Background

Before release 8.5.2 of GRS, it was not possible to configure multiple cluster nodes running on the same machine and controlled by the same cluster manager because separate entries for the same host could not be created in **bootstrapconfig.xml** to represent different GRE nodes. The pre-8.5.2 format of the **bootstrapconfig.xml** allowed for a single node to be defined per host. The xml format was as follows:

```
<xs:complexType name="node">
  <xs:sequence>
    <xs:element name="cfgserver" type="cfgserver" minOccurs="1" maxOccurs="1"/>
    <xs:element name="lcaserver" type="lcaserver" minOccurs="0" maxOccurs="1"/>
    <xs:element name="application" type="application" minOccurs="1"
maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="host" type="xs:string"/>
  <xs:attribute name="ipaddress" type="xs:string"/>
</xs:complexType>
```

What's New?

In GRS 8.5.2, an additional attribute called **servername** has been added to the node definition. This makes it possible to define multiple nodes for the same host. The server name is defined via the WebSphere Application Server (WAS) Deployment Manager when the cluster node is created.

For example, you can replicate the “node” definition for each GRE that is running on the same host. Then, by adding **servername**, you can make the entry unique. Each entry then points to the corresponding Configuration Server application for that GRE instance. In this way, a single **bootstrapconfig.xml** file can be used to define all nodes in the Websphere cluster, whether or not there are multiple GRE nodes defined on a given host.

To ensure backward compatibility, if no node is found within the **bootstrapconfig.xml** that matches both the hostname and **servername** then the node that contains the **hostname** with no server name defined serves as the default.

Editing the bootstrapconfig.xml file

To edit this file, manually extract the **bootstrapconfig.xml** file from the .war file, edit and save the **bootstrapconfig.xml** file, then repackage the **bootstrapconfig.xml** file back into the .war file.

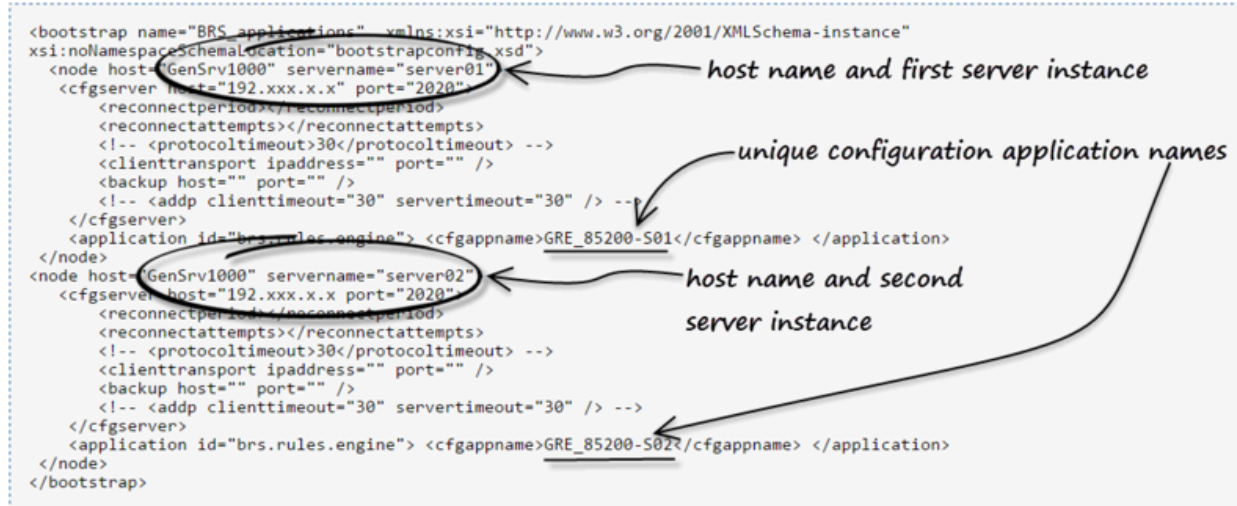
Sample bootstrapconfig.xml files

Important

Terminology—In the **bootstrapconfig.xml** files, the **<node>** element corresponds to an individual member of a WebSphere cluster.

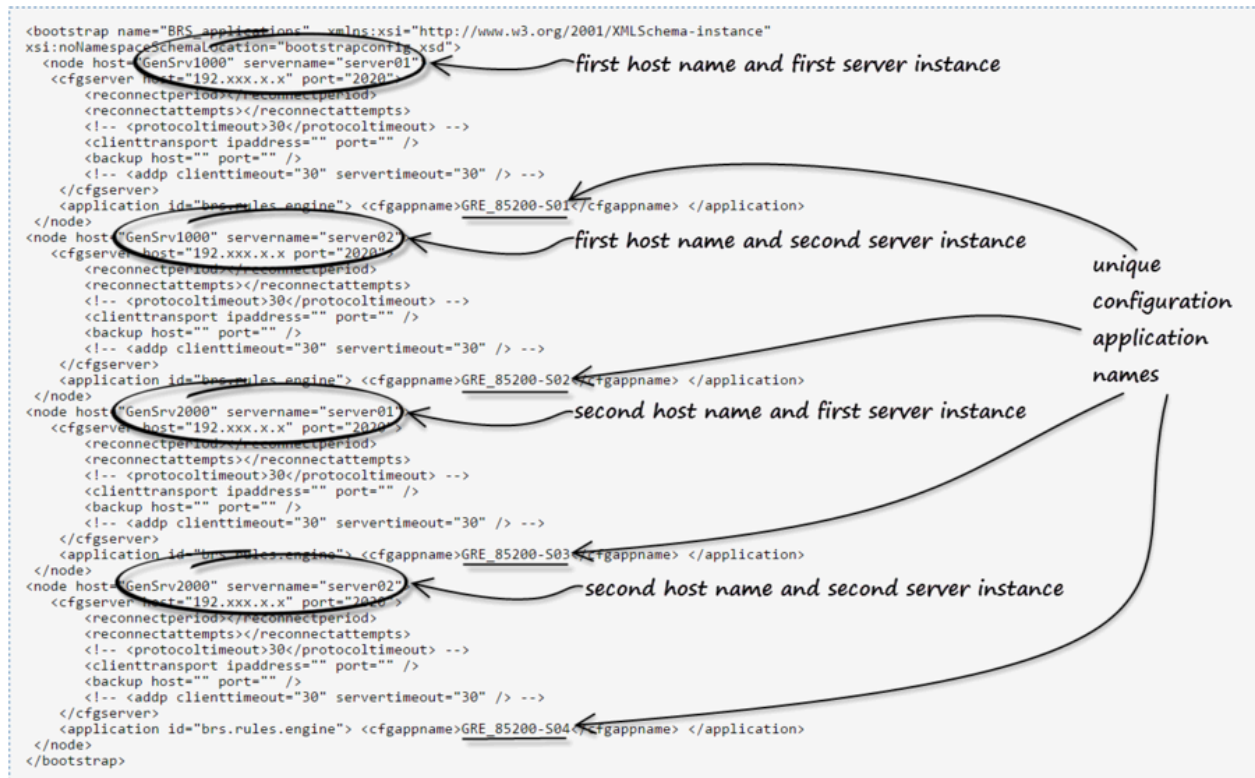
For a cluster with one host and two server instances on that host

Below is a sample **bootstrapconfig.xml** definition for a GRE cluster running on one host, GenSrv1000, with server instances server01 and server02 on that host:



For a cluster with two hosts and two server instances on each host

Below is a sample **bootstrapconfig.xml** definition for a GRE cluster running on two hosts, GenSrv1000 and GenSrv2000, with server instances server01 and server02 on each host:



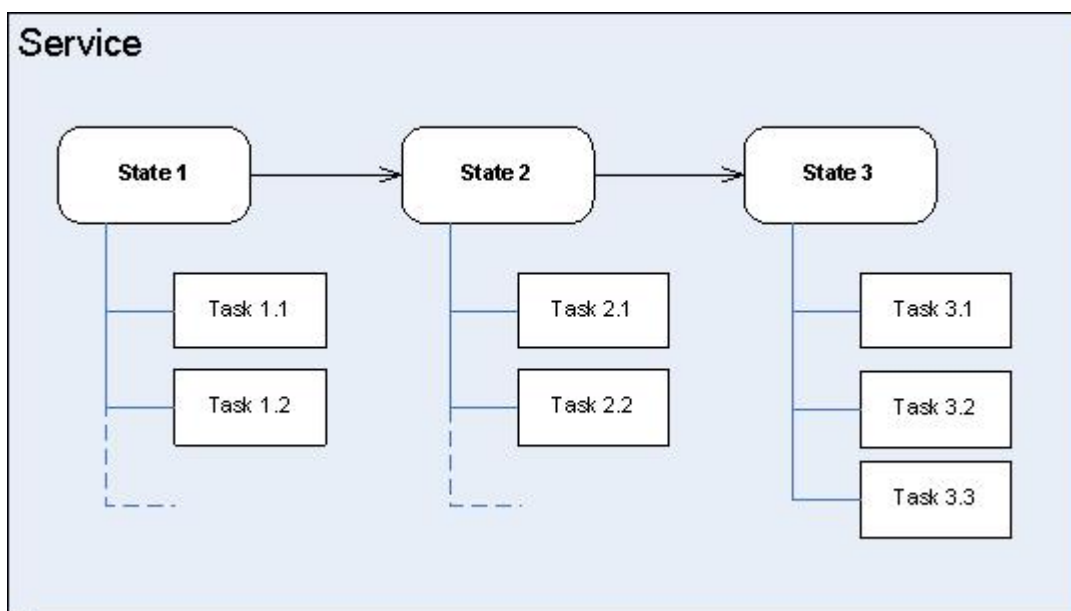
New in 8.5.1

Support for Conversation Manager Templates in Test Scenarios

In the initial 8.5.001 release of GRS, the Test Scenario feature did not support rules that were created using the Conversation Manager (CM) template. This is because the Test Scenario feature in release 8.5.001 works by taking the input data (a set of one or more facts with different fields) that is configured by the user and building the appropriate Fact model, then running the rules under GRAT using that set of data. In release 8.5.1, the Test Scenario feature now supports rules based on the CM template.

Data Structure in CM

With Conversation Manager, the data is in a hierarchical JSON format of **Customer -> Service -> State -> Task**. Any given **Customer** may have one or more **Services**. Each **Service** may be in at most one **State** at a time. Each **State** may have one or more **Tasks**. **Tasks** may also be associated directly with **Services**.



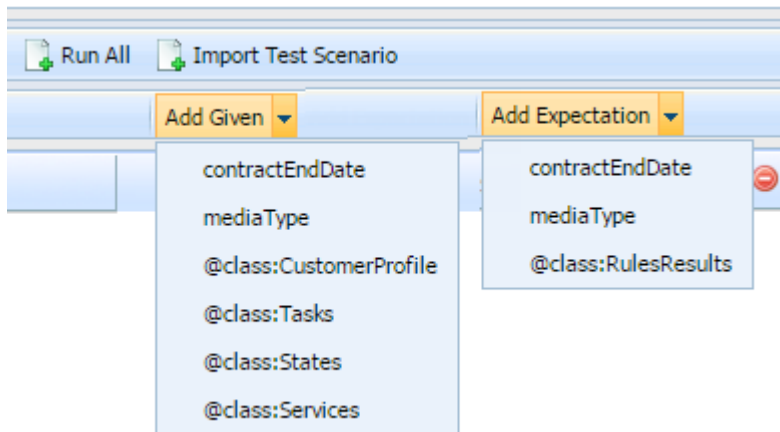
So the Customer, Services, States and Tasks Facts have now been added the lists of Facts that can be defined as **Given** fields, and the RulesResults Fact has been added to the list of Facts that can be defined as an **Expectation**.

Important

The current CM Template is only interested in the Type, Start Time, and Completion Time (if any) of Services, States, and Tasks.

Each of the new values is represented by a JSON string which will be the value for that field.

Now, when the type of rule for which you want to create a test scenario is a Conversation Manager rule (based on the Conversation Manager template), a series of different values for the **Given** and **Expectation** elements that reflect these more complex data structures are available. In the example below you can see the **Customer > Service > State > Task** structure is reflected by the four **@class** entries in the drop-down list of Givens and the **@class:RulesResults** entry in the drop-down list of Expectations.



When you select an **@class** entry, a new column is added. Click on a grid cell under the new column to bring up the edit dialog for that entry. The additional data listed below can be selected as either a **Given** or an **Expectation**.

Additional CM Template Objects

Givens

The list below shows the additional provided data.

- Available by selecting one of the **@class** entries:
 - Add Customer Attribute
 - Add Service
 - Add Service Type
 - Add Service Start Time
 - Add Service Completion Time
 - Add State
 - Add State Type
 - Add State Start Time
 - Add State Completion Time
 - Add Task
 - Add Task Type
 - Add Task Start Time

- Add Task Completion Time
- Available for direct selection from **Givens**:
 - Add Interaction Media Type
 - Add Contract End Date

Expectations

The list below shows the additional expected results:

- Update Customer Attribute
- Request Specific Agent
- Request Agent Group
- Request Place Group
- Request Skill
- Send Communication to Customer
- Block Communication to Customer
- Offer Service Resumption
- Offer Survey to Customer

Edit Dialogs

To create entries for the **Givens** and Expectations of your Conversation Manager test scenario, select the relevant **@class** item and use the sample additional edit dialogs shown below.

Givens

Customer	Parameter	Value	
[-] Customer	+		
[-]	Title		-
[-]	Phone Number		-
[-]	Last Name		-

Edit Services

Service	Parameter	Value	
[-] [Enter service Id]	+		+
[]	Type		-
[]	Completion Time		-
[]	Start Time		-
[]	Custom - String [Enter parameter name]		-
[]	Custom - Integer [Enter parameter name]		-

Edit States

State	Parameter	Value	
[-] [Enter state Id]	+		+
[]	Type		-
[]	Completion Time		-
[]	Start Time		-
[]	Custom - String [Enter parameter name]		-
[]	Custom - Integer [Enter parameter name]		-

Edit Tasks

Task	Parameter	Value	
[-] [Enter task Id]	+		+
[]	Type		-
[]	Completion Time		-
[]	Start Time		-
[]	Custom - String [Enter parameter name]		-
[]	Custom - Integer [Enter parameter name]		-

Expectations

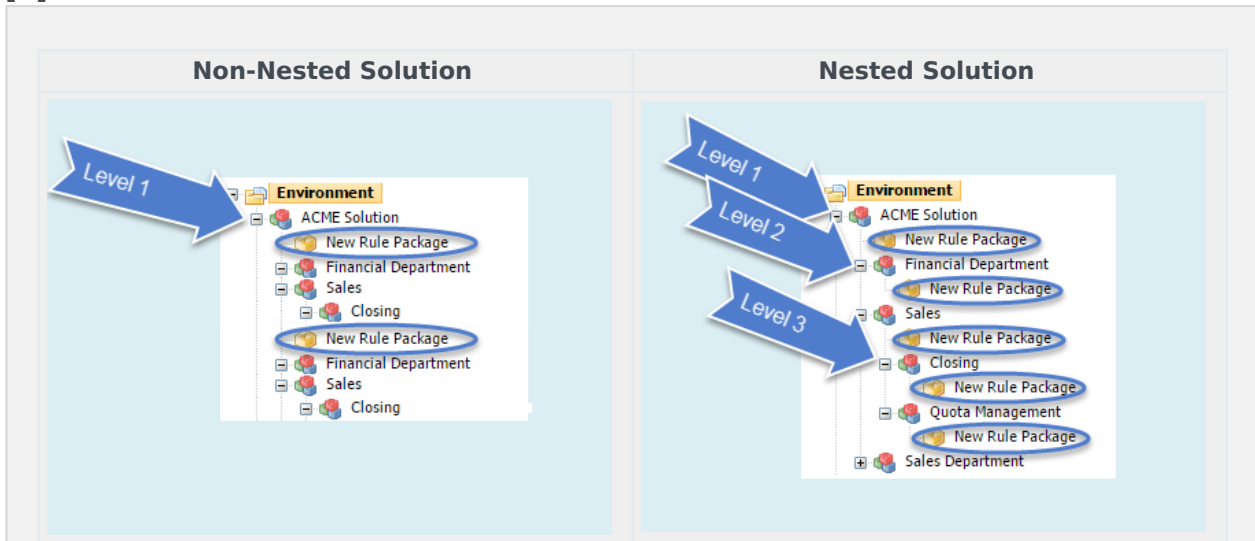
Edit Rules Results

Rules Results	Parameter	Value	
[-] Updated Fields	+		
[]	Title		-
[]	Phone Number		-
[]	Last Name		-
[-] Results	+		
[]	Send Communication	"{mediaType}"	-

Nested Solution Business Hierarchy

In release 8.5.1 of Genesys Rules Authoring tool, if you have permission to create a new rule (Rule Package - Create) you can now add a new Rule Package at any node in the business hierarchy (a *nested* solution), rather than just at the first level.

[+] FULL DESCRIPTION

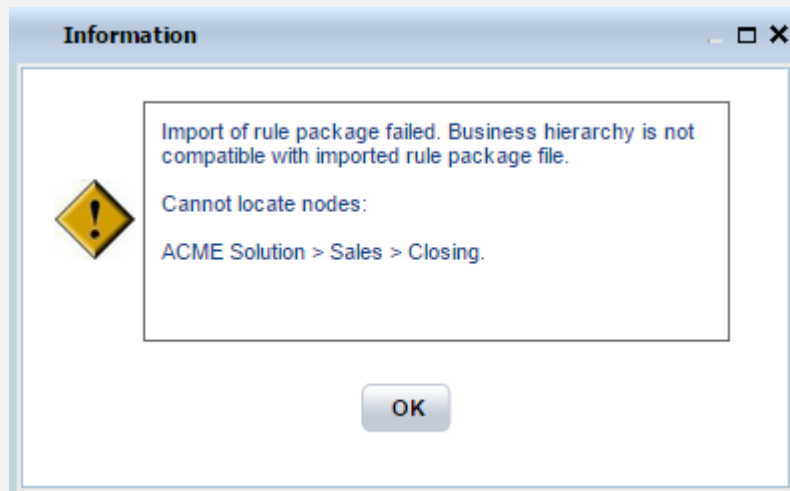


This means that:

- Your business hierarchy can be more easily organized.
- The need for lots of duplication and repetition at the Solution level in more complex business hierarchies is now removed.
- Individual users can be restricted using Role-Based Access Control to specific sub-nodes (for example, Departments and Processes).

Importing Rule Packages

Because rules can be associated with sub-nodes in a nested hierarchy, when a rule package is imported, GRAT ensures that the business structure is compatible, and prevents an import if it is not. If GRAT finds an incompatibility, an error such as the following is displayed:

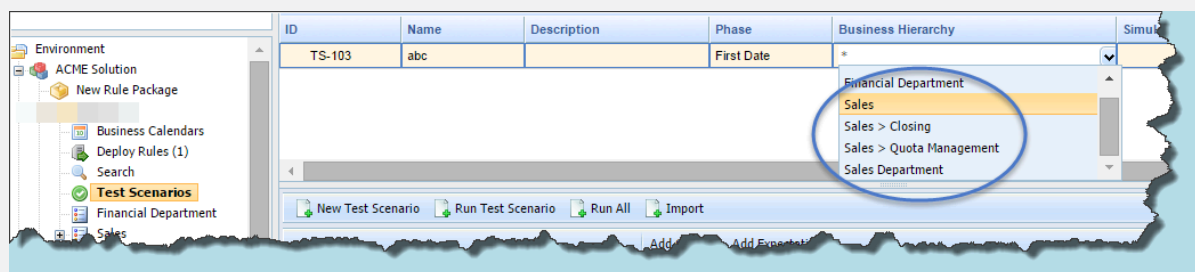


Important

Even if the **Auto-create business hierarchy during import** button is selected, GRAT prevents the same node name from being created anywhere in the hierarchy—uniqueness of business node names across the entire hierarchy is still enforced.

Test Scenarios

For Test Scenarios, the Business Hierarchy drop-down displays the relative path underneath the selected rule package:



Deleting Business Nodes

Be careful never to delete any business structure nodes that contain active rule packages or rules, without first backing up the rule packages as XML files. While it is OK to add new nodes, or to "rename" nodes, proper permissions should be set up to prevent a GA/GAX user from accidentally deleting nodes which could cause rule packages / rules to become unreachable.

Enabling and Disabling the Feature

Because some customers might want to restrict their users to creating rule packages only under the **Solution** node, in release 8.5.100.21 a new configuration option—`enable-nested-solutions`—has been implemented to allow users to enable or disable this feature. Disabling this feature is recommended for iWD users.

- Option name—`enable-nested-solutions`
- Valid values—`true/false`
- Default value—`true`
- Description—Controls whether users can create new rule packages under any node in the hierarchy. For iWD, it is recommended to set this option to `false`.

New in 8.5.001.21

Business Calendar Enhancements

Business calendars have been enhanced in GRS release 8.5.2 to allow:

- Dynamic Timezone Support
- Differentiation between holidays and non-working days.

[+] FULL DESCRIPTION

Dynamic Timezone Support

When the GRAT user configures a business calendar, a timezone is chosen along with the other attributes of the calendar (normal work week, exceptions, holidays). Business calendars have been enhanced to allow the timezone to be provided dynamically at rule-evaluation time. This feature enables you to configure a standard set of business calendars that can be re-used in any timezone.

In this release, the standard methods that can be accessed from within the rule template have been extended to allow the timezone ID to be passed in at rule evaluation time. If the timezone ID is not passed in in this way, then the "saved" timezone is used. If the timezone ID is passed in, then it overrides the saved timezone and the calculations will be done using the provided timezone.

Example

A rule, such as in the example below, asks which business calendar to use in evaluating the rule conditions;

ID	Name	Caller Language is	Type is	Call Type is	Determine Business Hours and Holiday with Calendar
DTR-174		English (en-US)	standard	Agent	English - Agent
DTR-185		English (en-US)	standard	Acquisition	English - Acquisition

and this can answer questions about whether today is a working day (by using the `isWorkingDay` method), or whether this time is in business hours, and so on. Now, you can configure the calling application (that is calling for the rule evaluation) to pass in a timezone ID as a parameter to the relevant business calendar function, and if present, this will override the configured timezone of the business calendar that the rule was created with.

For example, for the `isWorkingDay` method, the addition of the `timezoneID` parameter (highlighted below in the GRST template) enables this override to occur if the parameter is passed in:

Function properties

Name: `isWorkingDay`
Description:

Implementation

```

import org.drools.ide.common.client.modeldriven.cal.BusinessCalendar;
import java.util.Date;

function boolean isWorkingDay(BusinessCalendar cC, Date dNow)
{
    if (cC == null) {
        //LOG.info("isWorkingDay: Business Calendar is not set");
        return false;
    }
    return cC.isWorkingDay(dNow, timezoneID);
}

```

The `isWorkingDay` condition is then evaluated based on the dynamic timezone passed in by the calling application.

New Method Signatures to the BusinessCalendar Object

The following new method signatures have been added to the `BusinessCalendar` object, and can be invoked from within a rule function:

- `public boolean isWorkingDay(Date theDate, String timeZoneID);`

- `public boolean isHoliday(Date theDate, String timeZoneID);`
- `public boolean isException(Date theDate, String timeZoneID);`
- `public boolean isWorkingTime(Date theTime, String timeZoneID);`
- `public int diffWorkingDays(Date date1, Date date2, String timeZoneID);`
- `public int diffWorkingHours(Date date1, Date date2, String timeZoneID);`
- `public int diffWorkingMinutes(Date date1, Date date2, String timeZoneID);`
- `public long diffWorkingSeconds(Date time1, Date time2, String timeZoneID);`
- `public Date beginningOfWorkingDay (Date time, String timeZoneID);`
- `public Date endOfWorkingDay (Date time, String timeZoneID);`

Differentiation between Holidays and Non-Working Days

Business calendars have been enhanced to distinguish between holidays and non-working days. Four new methods have been added to the business calendar object:

- `isHoliday()`—Returns whether this calendar day is a holiday. Holidays are always non-working days.
- `isHoliday(date)`—As for `isHoliday()` but allows you to specify the date to check.
- `isException()`—Returns whether the day is an exception to the standard work schedule. An exception includes either a time change or a holiday.
- `isException(date)`—As for `isException()` but for a specified day.