Universal Routing Reference

Universal Routing 8.1.4

9/10/2024

Table of Contents

Supplement to the Universal Routing 8.1 Reference Manual

This supplement provides descriptions of new features introduced in Universal Routing 8.1.4 as part of the Continuous Delivery project. This supplement also describes updates to the *Universal Routing 8.1 Reference Manual*.

The complete documentation set for Universal Routing can be found here.

New Features

The following pages contain new features added to Universal Routing added after the last publication of the *Universal Routing 8.1 Reference Manual*.

This supplement provides descriptions of new features introduced in Universal Routing 8.1.4 as part of the Continuous Delivery project. It also contains updates to the *Universal Routing 8.1 Reference Manual*.

The current documentation set for Universal Routing can be found here.

New 8.1.4 Features

The following features are described in this supplement:

Released in Version	New Feature	Date Released
URS 8.1.400.83	Improved EWT Accuracy	Dec 03, 2021
URS 8.1.400.81	URS now allows specifying attributes in XML elements in SOAP requests. Attributes with assigned values must be defined in the **Key** field in **Request Parameters** of a Web Service object in the IRD strategy. The first parameter in the **Key** field is treated as the element name, and all the following parameters separated by a space are treated as attributes of the element.	August 19, 2021
URS 8.1.400.81	URS now supports Time-based notifications. When opening statistics with StatServer, in addition to the default **ChangesBasedNotification** mode, URS now supports the **TimeBasedNotification** mode. To activate it: • If the statistic is defined as a statistic transaction object in Configuration Layer, specify the notification interval in seconds in the **notify_mode** property in its **Statistic Data** section on the **Annex** tab. And use this statistic in the strategy.	August 19, 2021

Released in Version	New Feature	Date Released
	• If the statistic is used as a statistic type defined within the options of the StatServer application object, specify the notification interval in the **NotifyMode** parameter directly in the strategy, in the following format: Statistic:<StatisticName>|**NotifyMode**:<time_interval>	
URS 8.1.400.81	When there are a huge number of VCB calls, URS now records a standard level log message indicating the number of agents blocked for those VCB calls in the following format: `Std 21012 - Attention! vcb blocking mode is active: N` (where N is the number of agents locked).	August 19, 2021
URS 8.1.400.78	Multithreading Capability	May 18, 2021
URS 8.1.400.78	Timeout to Wait before Sending Negative Response to Web Client	May 18, 2021
URS 8.1.400.75	Optimal Skill Update Mode	December 23, 2020
URS 8.1.400.71	Secure Connections Support Includes SNI	September 01, 2020
URS 8.1.400.67	Support for JavaScript	February 04, 2020
IRD 8.1.400.39	String Manipulation Functions	October 25, 2019
URS 8.1.400.63	The `stat/report` web method is extended with a new parameter, `cpu`, to provide URS CPU consumption data, which can be used for monitoring the health status of URS applications. For more information, see Additional Information on HTTP Report Method.	September 30, 2019
URS 8.1.400.63	A new value, `waited`, has been added to the `report_targets` option. If the `report_targets` option is set to *true* or *waited*, URS attaches the `RTargetsWaited` key into `AttributeUserData` of the T-Server's events. The value of the new key is a comma separated list of targets the interaction is waiting for. This data can be used by the default routing strategy if the processing of an interaction	September 30, 2019

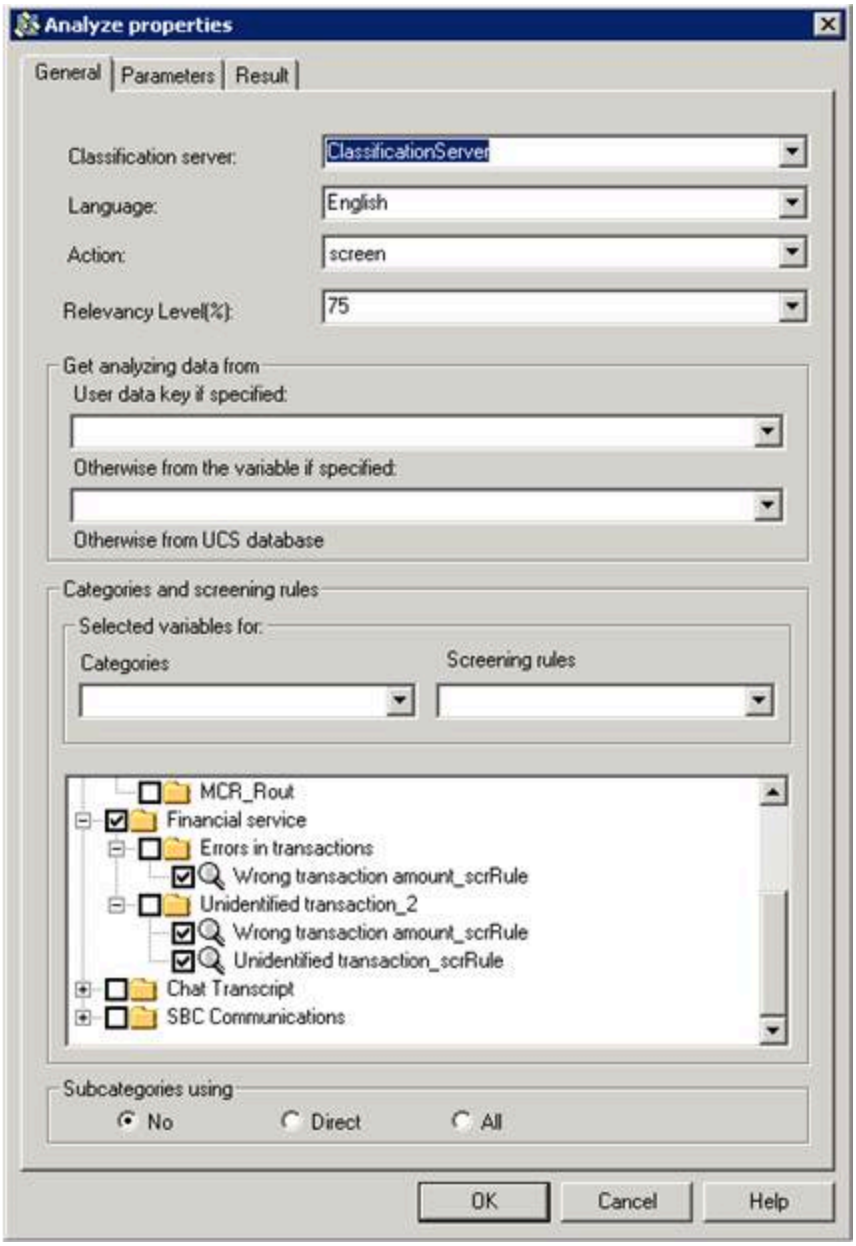
Released in Version	New Feature	Date Released
	fails.	
URS 8.1.400.50	Budget-Based Routing	June 29, 2018
URS 8.1.400.50	New Statistics	June 29, 2018
URS 8.1.400.45	HTTP CONNECT tunneling	March 30, 2018
URS 8.1.400.39 IRD 8.1.400.26	Run function	October 27, 2017
URS 8.1.400.27	Support of HTTP Proxies for an "https://" type of request	June 20, 2016
IRD 8.1.400.22	IRD Localization	April 15, 2016
URS 8.1.400.25	Estimated Wait Time Improvement Using URS Web API	December 18, 2015
URS 8.1.400.24	Look Up Agent Name Media logged into and DN from Agent Login ID and by Employee ID	November 24, 2015
URS 8.1.400.23	Support of HTTP Redirections Escape Sequences Update for HTTP Bridge When URS processes multimedia interactions in a High Availability deployment, a primary URS distributes EventDiverted on a virtual queue if a failover or switchover occurs.	October 7, 2015
IRD 8.1.400.18	Hyphens Allowed in Interaction Data Names	September 1, 2015
URS 8.1.400.20	Virtual Queues Can be Associated with Stat Servers	June 12, 2015
URS 8.1.400.19 URS 8.1.400.22	Using Agent Skills for Ideal Agent Selection Weighted Agent Skills Enhancement	May 8, 2015 September 25, 2015
IRD 8.1.400.15	Analyze Object	March 10, 2015

Analyze Object

Release 8.1.400.15 of Interaction Routing Designer adds an Analyze object for enhanced content analysis. The Analyze object, available on IRD's Multimedia group of objects, combines the functionality of IRD's existing Classify and MultiScreen objects and adds new functionality. The Analyze object:

- Sorts screening rules into categories.
- Visually presents screening rules under their applicable categories. The category name shows all roots so screening rules belonging to multiple categories can be selected in the same request.
- Introduces two new attributes in the ESP Request: `Service` = `Analyze` and `Method` = `ClassifyScreenUniversal`.
- Allows modified screening of ESP requests with the ability to specify a category/folder containing a subset of screening rules to screen against.

The example below shows the `Financial Services` Category, `Unidentified Transaction_2` subcategory, associated with screening rules `Wrong transaction amount_scrRule` and `Unidentified transaction_scrRule`.



Configuring the Analyze Object

1. From the **Multimedia** toolbar, click the **Analyze** object then click inside the Routing Design window to insert the object.
2. Double-click the object to open its properties dialog box. A tree of the default Classification Server categories and screening rules appears below.
3. From the **Classification server** drop-down list on the **General** tab, select a Classification Server from those in the Configuration Server database. If you leave this field empty, Interaction Server uses the

first available Classification Server named in its Connections list.

4. From the **Language** drop-down list, select the language of the incoming interaction. The categories and screening rules shown will be changed according to selected language. Only those roots associated (through their **Annex** property General/Language) with the selected language will be shown.
5. From the **Action** drop-down list, select `screen` or `classify`. Only one **Action** can be configured. either `screen` or `classify`. For action `classify`, screening rules are not used so IRD hides screening rules in this case.
6. If the classify **Action** is selected, opposite **Relevancy Level**, enter a percentage indicating the minimum relevancy (confidence) each category must have (greater or equal) in order for Classification Server to consider an interaction as belonging to that category (threshold). You have the option of changing the default but you cannot enter zero.
   - If Classification Server finds a category with the minimum specified relevancy, the results are processed based on the option you select in the **Result** tab and the interaction goes through the green port.
   - If an existing category cannot be found based on analyzing the interaction, the null result is attached to the interaction and the interaction also goes through the green port (the error is not retrievable through the current Error object). See `Event3rdServerFault` in the Universal Routing 8.1 Reference Manual.
7. Under **Get analyzing data from**, enter a User Data key to search for data to be analyzed or enter a variable. If not specified, the data to be analyzed will be taken from the Universal Contact Server database.
8. Under **Categories and screening rules**, you have the option of selecting a variable to contain the categories and/or screening rules to use. If variables are used, you cannot select screening rules or categories from the tree. You can either select from the tree or specify variables, but you cannot do both.
9. Note the tree of categories/Screening Rules. If the identifiers do not sufficiently describe them, you can look up them up in Configuration Manager (`Business Attributes`) or Genesys Administrator or in eServices Knowledge Manager, where they were originally defined.
10. The **Subcategories using** options (required) determine whether Classification Server should consider parent and child categories (`ParentMode` in Requests as shown in Samples below. There are three options.
    - To use all screening rules for a selected category, select **All**.
    - To use only the direct children of a selected category (excluding parents), select **Direct**.
    - To individually select categories and rules, select **No**.
11. Continue to select categories/rules in this fashion until you have selected all the ones you want to use.
12. Select the **Parameters** tab. If necessary, any additional parameters to the request can be added here. For additional information see eServices Knowledge Manager documentation.
13. Select the **Result** tab. Specify what form the results should take. IRD uses the same **Result** tab for various Multimedia objects. When screening or classifying, the **Do not use output value** option does not apply.
14. Complete the **Result** tab. If you write the results to a variable, the variable can be parsed by URS and attached to the interaction for further use in the strategy. You can also attach all the parameters to the interaction's User Data. This enables you to analyze this User Data to make further routing decisions.
15. Click **OK**.

For classification, The response will be either `Event3rdServerResponse` or `Event3rdServerDefault`.

For detail on these responses as well as on using IRD for screening and classification, refer to the Universal Routing 8.1. Reference Manual.

Samples

[+] Analyze Action Classify

```
REQUEST
'Version' [str] = "1.0"
'AppType' [str] = "90"
'AppName' [str] = "ClassificationServer2_sbbelovdt"
'Service' [str] = "Analyze"
'Method' [str] = "ClassifyScreenUniversal"
'Parameters' [lst] = KVList:
        'Action' [str] = "Classify"
        'Language' [str] = "English"
        'Categories' [str] = "00016a6W8MUA002Q | 00057a9K3JNC02V7 | 0005Ba9QMHV30KCY"
        'ParentMode' [str] = "3"
        'RelevancyLevel' [str] = "10"
        'IxnText' [str] = "You should transplant these irises into an area providing at least
6 hrs of direct sunlight a day."

bstr [bstr] = KVList:
'TenantId' [int] = 101

RESPONSE
'Version' [str] = "1.0"
'Service' [str] = "Analyze"
'Method' [str] = "ClassifyScreenUniversal"
'Parameters' [lst] = KVList:
        'Categories' [lst] = KVList:
                'How do I dig irises' [lst] = KVList:
                        'CtgId' [str] = "00057a9K3JNC038G"
                        'CtgName' [str] = "How do I dig irises"
                        'CtgRootName' [str] = "plants"
                        'CtgPath' [str] = "plants/iris/How do I dig irises"
                        'CtgRelevancy' [str] = "32"
                'Why are they not blooming' [lst] = KVList:
                        'CtgId' [str] = "00057a9K3JNC038N"
                        'CtgName' [str] = "Why are they not blooming"
                        'CtgRootName' [str] = "plants"
                        'CtgPath' [str] = "plants/iris/Why are they not blooming"
                        'CtgRelevancy' [str] = "26"
                'J2EBJ183K80HB6G2' [lst] = KVList:
                        'CtgId' [str] = "00016a6W8MUA005Y"
                        'CtgName' [str] = "J2EBJ183K80HB6G2"
                        'CtgRootName' [str] = "SBC_1_Root"
                        'CtgPath' [str] = "SBC_1_Root/J2EBJ183K80HB6G2"
                        'CtgRelevancy' [str] = "12"
                'Sub_A' [lst] = KVList:
                        'CtgId' [str] = "0005Ba9QMHV30KDE"
                        'CtgName' [str] = "Sub_A"
                        'CtgRootName' [str] = "MySimpleTest"
                        'CtgPath' [str] = "MySimpleTest/Sub_A"
                        'CtgRelevancy' [str] = "100"
                '19CSBDR3K82AXEGS' [lst] = KVList:
                        'CtgId' [str] = "00016a6W8MUA004K"
                        'CtgName' [str] = "19CSBDR3K82AXEGS"
                        'CtgRootName' [str] = "SBC_1_Root"
```

```
                        'CtgPath' [str] = "SBC_1_Root/19CSBDR3K82AXEGS"
                        'CtgRelevancy' [str] = "36"
                'How should I prepare iris for shipping' [lst] = KVList:
                        'CtgId' [str] = "00057a9K3JNC038Y"
                        'CtgName' [str] = "How should I prepare iris for shipping"
                        'CtgRootName' [str] = "plants"
                        'CtgPath' [str] = "plants/iris/How should I prepare iris for shipping"
                        'CtgRelevancy' [str] = "19"
                'Why did my irises change color' [lst] = KVList:
                        'CtgId' [str] = "00057a9K3JNC038T"
                        'CtgName' [str] = "Why did my irises change color"
                        'CtgRootName' [str] = "plants"
                        'CtgPath' [str] = "plants/iris/Why did my irises change color"
                        'CtgRelevancy' [str] = "25"
        'CtgId' [str] = "0005Ba9QMHV30KDE"
        'CtgName' [str] = "Sub_A"
        'CtgRootName' [str] = "MySimpleTest"
        'CtgPath' [str] = "MySimpleTest/Sub_A"
        'CtgRelevancy' [str] = "100"
```

[+] Analyze Action Screen

```
REQUEST
'Version' [str] = "1.0"
'AppType' [str] = "90"
'AppName' [str] = "ClassificationServer2_sbbelovdt"
'Service' [str] = "Analyze"
'Method' [str] = "ClassifyScreenUniversal"
'Parameters' [lst] = KVList:
        'Action' [str] = "Screen"
        'Language' [str] = "English"
        'Categories' [str] = ""
        'Rules' [str] = ""
        'RelevancyLevel' [str] = "15"
        'ParentMode' [str] = "3"
        'IxnText' [str] = "problem, accounts 1111-1111-1111-1111 and 2222-2222-2222-2222"

bstr [bstr] = KVList:
'TenantId' [int] = 101

RESPONSE
'Version' [str] = "1.0"
'Service' [str] = "Analyze"
'Method' [str] = "ClassifyScreenUniversal"
'Parameters' [lst] = KVList:
        'ScreenRuleMatch' [str] = "true"
        'Categories' [lst] = KVList:
                'Neutral' [lst] = KVList:
                        'CtgId' [str] = "00006a69F6861XCW"
                        'CtgName' [str] = "Neutral"
                        'CtgRootName' [str] = "Sentiment"
                        'CtgPath' [str] = "Sentiment/Neutral"
                        'CtgRelevancy' [str] = "75"
                        'Screen' [lst] = KVList:
```

```
                'ScreenForNeutralSentiment' [lst] = KVList:
                        'RuleId' [str] = "00006a69F6861XED"
                        'RuleName' [str] = "ScreenForNeutralSentiment"
                        'RuleOrder' [str] = "12"
                        'RuleRelevancy' [str] = "75"
        'Positive' [lst] = KVList:
                'CtgId' [str] = "00006a69F6861XCP"
                'CtgName' [str] = "Positive"
                'CtgRootName' [str] = "Sentiment"
                'CtgPath' [str] = "Sentiment/Positive"
                'CtgRelevancy' [str] = "85"
                'Screen' [lst] = KVList:
                        'Tech support' [lst] = KVList:
                                'RuleId' [str] = "00003a01DST4006D"
                                'RuleName' [str] = "Tech support"
                                'RuleOrder' [str] = "400"
                                'FoundValues' [lst] = KVList:
                                        'CardNo(1)' [str] = "1111-1111-1111-1111"
                                        'CardNo(2)' [str] = "2222-2222-2222-2222"
                                'RuleRelevancy' [str] = "85"
                        'SB_MultiScanTest' [lst] = KVList:
                                'RuleId' [str] = "0003Fa8RRCHA0030"
                                'RuleName' [str] = "SB_MultiScanTest"
                                'RuleOrder' [str] = "10"
                                'FoundValues' [lst] = KVList:
                                        'CardNo(1)' [str] = "1111-1111-1111-1111"
                                        'CardNo(2)' [str] = "2222-2222-2222-2222"
                                'RuleRelevancy' [str] = "75"
        'Actionable' [lst] = KVList:
                'CtgId' [str] = "00006a69F6861XHN"
                'CtgName' [str] = "Actionable"
                'CtgRootName' [str] = "Action"
                'CtgPath' [str] = "Action/Actionable"
                'CtgRelevancy' [str] = "75"
                'Screen' [lst] = KVList:
                        'Tech support' [lst] = KVList:
                                'RuleId' [str] = "00003a01DST4006D"
                                'RuleName' [str] = "Tech support"
                                'RuleOrder' [str] = "400"
                                'FoundValues' [lst] = KVList:
                                        'CardNo(1)' [str] = "1111-1111-1111-1111"
                                        'CardNo(2)' [str] = "2222-2222-2222-2222"
                                'RuleRelevancy' [str] = "75"
                        'SB_MultiScanTest' [lst] = KVList:
                                'RuleId' [str] = "0003Fa8RRCHA0030"
                                'RuleName' [str] = "SB_MultiScanTest"
                                'RuleOrder' [str] = "10"
                                'FoundValues' [lst] = KVList:
                                        'CardNo(1)' [str] = "1111-1111-1111-1111"
                                        'CardNo(2)' [str] = "2222-2222-2222-2222"
                                'RuleRelevancy' [str] = "45"
        'UnclearIfActionRequired' [lst] = KVList:
                'CtgId' [str] = "00006a69F6861XJ3"
                'CtgName' [str] = "UnclearIfActionRequired"
                'CtgRootName' [str] = "Action"
                'CtgPath' [str] = "Action/UnclearIfActionRequired"
                'CtgRelevancy' [str] = "75"
                'Screen' [lst] = KVList:
                        'UnclearIfActionRequired' [lst] = KVList:
                                'RuleId' [str] = "00006a69F6861XN2"
                                'RuleName' [str] = "UnclearIfActionRequired"
                                'RuleOrder' [str] = "10"
                                'RuleRelevancy' [str] = "75"
```

```
'CtgId' [str] = "00006a69F6861XCP"
'CtgRelevancy' [str] = "85"
'CtgName' [str] = "Positive"
'CtgRootName' [str] = "Sentiment"
'CtgPath' [str] = "Sentiment/Positive"
```

Budget-Based Routing

Starting with release 8.1.400.50, URS release extends Agent Capacity-Based Routing to support the Budget-Based Routing functionality implemented in Stat Server.

> **Important**
>
> Budget-Based Routing works with Stat Server version 8.5.110.03 or higher. To enable Budget-Based Routing, refer to the Stat Server documentation. Budget-Based Routing is not related to Cost-Based Routing and is an extension to Agent Capacity-Based Routing.

Budget-Based Routing is an optional addition to the existing Agent Capacity-based routing. Legacy Agent Capacity routing is based on the certain number of simultaneous interactions of different media types that an agent can handle.

The new functionality is based on the interaction cost and agent budget for a given media type, in addition to the legacy Agent Capacity. It allows to take into account that different interactions (even of the same media type) might require different levels of handling effort from an agent. When the budget model is enabled, URS selects a target according to the following rules:

- It determines an agent's ability to accept an interaction of a particular media type based on agent capacity information (legacy Agent Capacity) provided by Stat Server.
- It verifies if the cost of an interaction is explicitly attached to the interaction with a key as defined in the `interaction-cost-key` configuration option.

> **Important**
>
> If the name of the key is not defined, then the name `InteractionCost` is used.

- It uses agent budget information (available, total and used) provided by Stat Server in the `CurrentTargetState` statistic.
- It verifies that the cost of the interaction is less than the agent's remaining budget.

> **Important**
>
> If the interaction has no cost-related information in its attached data, then URS considers its cost as `0` (zero) and as a result, budget restrictions are not applied to the interaction.

Using Agent Skills for Ideal Agent Selection

Starting with URS 8.1.400.19, Universal Routing can select the most ideal agent to handle an interaction when more than one agent is available. You can also use this functionality to select the most ideal interaction when there is more than one interaction competing for the same agent. To implement this functionality, this release introduces two new functions, `SetIdealAgent`, `TargetListSelected` and new option `set_ideal_agent`.

SetIdealAgent

Parameter: Skill Expression: `STRING` (constant or variable representing a skill expression)

Return value type: `VOID`

This function is available from IRD's **Function** object, **Target Manipulation** category. The `SetIdealAgent` function utilizes a "best fit" factor as one criteria for selecting the most ideal agent to handle an interaction or to select the most ideal interaction when there is more than one interaction competing for the same agent. Once you define the ideal agent's skill set, Universal Routing Server will use this definition if there is any choice when assigning agents to interactions.

Example Skill Expression

The `SetIdealAgent` function accepts as input a skill expression that you define via the **Skill Expression** field in the Function object dialog box. Clicking opposite **Skill Expression** opens the **Skill Expression Properties** dialog box where you create the skill expression. Genesys recommends that functions not be used in ideal agent skill expressions and that the skill expression be in DNF form, such as:

```
Skill1=value1 & Skill2>=value2 & Skill2<=value3 | Skill3=value3
```

URS would interpret the above expression as follows: The ideal agents are those who have Skill1 set to value1 (`Skill  Level` in Configuration Database) and Skill2 in between value2 and value3 or alternatively agents with Skill3 set to a value of 3.

Other examples:

```
Skill_A < 3 | Skill_A < 6 & Skill_C > 3 & Skill_D >= 6
Skill_A > 3 & Skill_B > 3 & Skill_C > 3 & Skill_D < 3
Skill_A >= 2 & Skill_B >= 3 & Skill_C >= 4
```

For information on creating skill expressions, see the *Universal Routing 8.1 Interaction Routing Designer Help*. Go to Creating a New Strategy > Expression Building. For more detailed information, search on "Skill Expression" in the *Universal Routing 8.1 Reference Manual*.

How the Skill Expression is Used

When processing interactions using the ideal agent definition, URS measures every qualified agent for the interaction being processed to determine how exactly the agent's skills match the ideal agent skills. An extra metric is associated with every agent (see `TargetListSelected` below), which indicates how close the agent is to the ideal agent definition. URS then selects the agent whose skills deviate the least from the skills of the ideal agent.

For more information on this deviation, see Multiple interactions Competing for Same Agent.

Calculated Deviation

A new function, `TargetListSelected`, returns the calculated deviation of the selected agent from the ideal agent.

TargetListSelected

Parameter: Key: `None`

Return value type: `STRING`

This function returns information about the selected target. If used after the function `SetIdealAgent`, it will return additional key `mismatch` with the deviation value of the selected agent. This function also returns much of the same data that functions SelectDN/SuspendForDN return about the selected target as described in the *Universal Routing 8.1 Reference Manual*.

Calculating the Deviation From Ideal

URS calculates the deviation of an agent from the ideal agent in following way:

If the comparison operation evaluates to a true score of 0, the agent skill does not deviate from the ideal agent skill. A score greater than 0 indicates how much the agent skill is different from the ideal agent skill.

- Use of the & operator in the skill expression adds to the score.
- Use of the | operator in the skill expression results in a minimal score.

URS pre-calculates agent skill deviations at the moment some queue is created, not at the moment when an agent or interaction is actually selected. That means that skill expressions used to define the ideal agent must contain only skills, numbers, and logical/comparing/mathematical operations. URS tracks skill updates in the Configuration Database and will recalculate deviations if a skill update occurs. Genesys recommends not using functions in skill expressions. URS does not track when a function value changes, which could result in the wrong deviations being used.

Example Deviation Calculation

Agent Skills:

AGENTS	SKILL A	SKILL B	SKILL C	SKILL D
Agent 1	1	2	3	4
Agent 2	2	3	4	
Agent 3	3	4	5	6
Agent 4	4	5	7	1
Agent 5	5	6	1	2
Agent 6	5	6	1	2

Deviation for the Skill_expression:

```
Skill_A >=2 & Skill_B <=5 & Skill_C=5
```

AGENTS	SKILL_A >=2	SKILL_B <=5	SKILL_C = 5	SKILL_D	DEVIATION
Agent 1	1	0	2	0	3
Agent 2	0	0	1	0	1
Agent 3	0	0	0	0	0
Agent 4	0	0	2	0	2
Agent 5	0	1	4	0	5
Agent 6	0	1	4	0	5

Agents with Same Deviation

If an ideal agent is set for an interaction and there are multiple available agents, then URS checks every one of them and selects the agent having the smallest deviation. If deviations are the same, then URS uses the value of a statistic to select an agent.

Ideal agent selection is agent-level selection and works as described above only when URS uses agent-level statistics. In this case, strategy targets are based on skill expressions or are the result of function `UseAgentStatistics[true]` being called in the strategy.

Agent/Place Groups

If a strategy uses Group-level statistics (URS uses statistics to select the best Agent Group or Place Group), then URS behaves as follows:

- URS uses the ideal agent skill expression to select an agent inside every participating Agent Group.
- If more than one Agent/Place Groups has available agents, then URS uses Group statistics to decide which group to use. URS selects based on the Group having the best statistics, not the group having the most ideal agents.

Specifically, this means if routing targets are sets of Agent Groups, URS uses `UseAgentStatistics[true]` to select the most ideal agent throughout all listed Agent Groups. Note

that because the default value of this function is `false`, when creating the strategy, you must set `UseAgentStatistics` to `true` and position the function before the Agent Group target.

Multiple Interactions Competing for Same Agent

Note: Apart from step 2, all the steps listed below are the standard steps that URS always executes.

URS can also use skill expressions defined for function `SetIdealAgent` when there are multiple interactions in queue competing for the same agent (desired agent). As an example, assume an agent becomes available and there are two competing interactions for this agent (Call1 from VQ 1 and Call2 from VQ2). In this case:

1. URS checks interaction priority. You can use various functions (`Priority`, `IncrementPriority`, `SelectDN`, `SetVQPriority`, and so on) in a strategy to specify interaction priority for different queues. If none of the priority functions is used, then interaction priority is 0. If one of the priority functions is used, the interaction with the higher priority will be used. If priorities are the same, go to step 2.
2. URS checks "best fit" factor. URS only executes this step if both interactions have ideal agent set.

- *URS will select the interaction with the ideal agent skill set that is closest to the skill set of the desired agent.*
- **Note:** There may be some cases where at least one of the interactions will not have ideal agent set, or cases where the ideal agent definition for both interactions is equally close to the desired agent. For these cases, URS checks timing as described in step 3.

- When URS checks timing, the following values are calculated: `X1 = T1 + D1 and X2= T2+D2.`

The example below uses voice interactions.

- `T1` - The time Call1 has been waiting for this agent (with milisecond precision)
- `T2` - The time Call2 has been waiting this agent (with milisecond precision)
- `D1` - If the strategy for Call1 instructs to use prediction ("what-if" wait time where function `PriorityTuning` is used with Prediction set to true), then it is prediction time; otherwise it is 0.
- `D2` – If the strategy for Call2 instructs to use prediction ("what-if" wait time where function `PriorityTuning` is used with `Prediction` set to `true`) then it is prediction time; otherwise it is 0. Go to step 3b.

3b.If both interactions have a risk of not meeting the Service Objective, then

`X1= X1/Call1ServiceObjective` and `X2= X2/Call2ServiceObjective`.

If at least one of the interactions does not use Service Objective, this step is skipped (values X1 and X2 are not changed). Go to 3c.

3c.If X1 is bigger than X2, Call1 is used and vice versa. If (in the rare case) X1 and X2 are still the same, go to step 4.

4. URS keeps a global counter. Every time an interaction is placed into a waiting queue, the counter is incremented by one (no two different entries of some interaction into some queue can have the same

value). URS will select the interaction having the smaller counter.

Weighted Agent Skills

Starting with URS 8.1.400.22, you can apply a weight to any skill (or combination of skills) to increase the importance of a skill. Expressions can include both skill functions and arithmetic operations:

+ (plus)
- (minus)
/ (divide)
and
* (multiply)

URS considers the value of this expression as a deviation from the ideal agent (the less its value, the more ideal the agent).

There are no restrictions to using these operators, other than that the final skill expression must be a well-formed mathematical expression having the right balance of opening and closing brackets.

Weighted Ideal Agent Samples

For example, the following skill expressions show how mathematical expressions can increase or decrease a deviation and define the importance of a specific skill:

```
5*(Skill1=value1) & Skill2>=value2 & 4+(Skill2<=value3)
(Skill1=value1)-2 & 3*(Skill2>=value2 & Skill2<=value3)
3*(Skill2>=value2) & (Skill2<=value3)/2
```

The skill expression below is an example where an agent with an `employeeid` starting with `ABC` is 10 times better other agents (the deviation will be 0 for such an agent and 10 for all others).

```
10*(name(“ABC*”)=1)
```

The `name` and other skill expression functions are available in IRD's Skill Expression Properties dialog box.

If Strategy Does Not Use SetIdealAgent

Option `automatic_ideal_agent` can be used as an alternative to using `SetIdealAgent`.

automatic_ideal_agent

Location in Configuration Layer by precedence: Routing Point, T-Server, Tenant, URS

Default value: `false`

Valid values: `true`, `false`

Value changes: take effect immediately

If set to `true`, then when URS places the interaction into a queue for first time:

- If this queue targets/agents are defined as a skill expression and
- if function SetIdealAgent was not yet called for this call, then

URS will automatically call the `SetIdealAgent` function with the value of the skill expression used for this queue as a target.

Important Notes About Ideal Agent

- When URS evaluates routing targets, the use of the `SetIdealAgent` function adds an extra layer to the evaluation process. Except for the interaction priority layer, the Ideal Skill layer will take precedence over other layers defined in a routing strategy. As a result, use of the `SetIdealAgent` function in a strategy can possibly affect (for good or bad) existing routing solutions, such as Service Objective routing, Age of Interaction routing, and so on. If your routing strategy combines different target selection criteria, Genesys recommends that you analyze how these different types of target selection criteria will interact and coexist with each other.
- The first time URS executes a strategy, statistics are normally not yet opened. As a result, if the target selection is statistic or skill-based, the first interaction could be distributed to any ready agent.
- Selection of an ideal agent for an interaction is effectively agent-level selection. It works as described here only when URS uses agent-level statistics: either when targets are skill expressions or when function `UseAgentStatistics[true]` is called in strategy.
- The interaction selection criteria associated with the SetIdealAgent function are only supported in a multi-URS environments where the same target might be selected by different instances of URSs if:
  all URS instances have the same value of option `automatic_ideal_agent`
  or
  all strategies running/served by URSes include function `SetIdealAgent` (with not empty parameter).

> **Important**
>
> Applying different routing priority criteria (Service Level, Ideal Agent) to calls waiting for the same targets might sometimes result in out of order routing. In this case, URS records a *composite priority misused* message in the log.

Associating Virtual Queues with Stat Servers

Starting with Release 8.1.401.00, the URS `default_stat_server option` is extended to allow you to specify a separate default Stat Server for every Virtual Queue. The extension covers cases where the URS `Connections` list contains multiple Stat Servers.

> **Important**
>
> For the option description, see the `default_stat_server` section in the New or Updated Option Descriptions topic.

New Statistics

Two new statistics, StatAgentLoadingWgt and StatAgentLoadingWgtMedia, are introduced in release 8.1.400.50.

- When the StatAgentLoadingWgt statistic is used as the target selection criterion, the agent within the Agent or Place Group who has the lowest used agent's budget for all media types is selected as the target. If more than one agent has the same total used agent's budget, the time they have been in their current state is considered. When this statistic is used in a function, such as SData, it returns the used agent's budget for all media types.
- When the StatAgentLoadingWgtMedia statistic is used as the target selection criteria, the agent within the Agent or Place Group who has the lowest used agent's budget for the current media type is selected as the target. If more than one agent has the same used agent's budget, the time they have been in their current state is considered. When this statistic is used in a function, such as SData, it returns the used agent's budget for the current media type.

> **Important**
>
> When the StatAgentLoadingWgt and StatAgentLoadingWgtMedia statistics are defined in the Target Selection object, the **Min/Max** settings are ignored.

Hyphens Allowed in Interaction Data Names

Starting with Interaction Routing Designer Release 8.1.400.18, when configuring strategies that use Interaction Data, IRD allows the use of hyphens in Interaction Data names.

For example, you may want to display customer information for an incoming call on the agent desktop in the form of a screen pop. In this case, when using IRD to configure the Interaction Data for the screen pop, you now have the option of using hyphens in Interaction Data names.

HTTP Bridge Updates

To communicate with Web Services through SOAP/XML and/or REST over HTTP/HTTPS protocols, URS uses a component called HTTP Bridge. HTTP Bridge allows strategy developers to communicate with Web Services applications outside of Genesys via the Web Service IRD strategy-building object. For more information on HTTP Bridge, see the Universal Routing 8.1 Reference Manual.

Support of HTTP Redirections

Starting with Release 8.1.400.23 in October of 2015, URS enhances its support of HTTP redirections. HTTP Bridge now resends an HTTP request to the new address specified in the Location header of the received 3xx response.

When responding to a redirect request, HTTP Bridge now:

- Supports both absolute and relative redirection URLs.
- Checks for redirect loops. No more than 5 chained redirects will be allowed.
- Always uses the GET method for redirection URLs if a return code is 303.

Handling of Escape Sequences

Starting with Release 8.1.400.14 in February 2015, HTTP Bridge no longer interprets XML escape sequences as regular delimiters (< and >) of XML tags. It now passes them into the strategy as is and does not terminate while processing XML data returned by SOAP-based Web Services containing XML escape sequences.

Add the following information to the Web Services Options section and also to Appendix B, IRD Web Services Object:

HTTP Bridge does not get completely XML-formatted text from a Web Service. The text is XML-formatted to some level, but at deeper levels, the XML text is escaped. For example:

```
<?xml version='1.0' encoding='utf-8'?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Body>
<ns:saveCaseResponse xmlns:ns="http://util.ilog.ist.apple.com">
<ns:return>
< ;?xml version="1.0" encoding="UTF-8"?> ;< ;GENESYS_CASE_CREATE_RESPONSE> ;.....> ;
</ns:return>
</ns:saveCaseResponse>
</soapenv:Body>
</soapenv:Envelope>
```

Note that the content inside the `ns:return` tag is not XML-formatted text. It might be interpreted as XML-formatted text after replacing the Escape sequences

`< ; and > ;`

with the appropriate < >.

HTTP Bridge does not perform this extra interpreting. Any XML response, if needed, must be in entirely and correctly-formed XML format.

New or Updated Function Descriptions

The following functions have been either newly added or updated after the *Universal Routing 8.1 Reference Manual* was last published. Unless otherwise noted, these functions are available in IRD's Function object. Other functions are located in IRD's Selection object.

ExcludeAgents

Parameters: Agents: STRING (comma-separated list of agent IDs or variable)
Return value type: STRING

This function instructs URS not to route interactions to any agent on the specified list of agents. Parameter Agents is comma-separated list of agent IDs. Function returns the previous list of excluded agents.

Previous to 7.6, if a target was selected (if it was ready according to Stat Server and reserved) and then excluded from the list of valid targets using the ExcludeAgents function, this target was not actually excluded. Starting with 7.6, the ExcludeAgents function does exclude the agent in the above scenario.

Note: When URS executes the ExcludeAgents function for an interaction, the URS-provided list of excluded agents will be applied to the current or any future Selection objects. The effect of the ExcludeAgents execution can be cancelled only by the execution of another ExcludeAgents function or if URS stops this interaction processing.

Warning! Function ExcludeAgents affects IVR targets.

FindConfigObject

As of 11/24/15, the existing FindConfigObject function is extended to support additional object types as shown in the FindConfigObject Returned Results table below.

Look Up Agent Name, Media logged Into and DN, from Agent Login ID and by Employee ID

Function FindConfigObject with object type CFGPerson and function TargetState can be used to look up an agent name, the media channel logged into, and the agent's DN, based on an agent's Login ID or Employee ID.

Function TargetState[‘EmployeeID.A’] with input parameter agent EmployeeID returns agent readiness information, a list of available medias and DNs.

For example, to use FindConfigObject to find agent (Person) information, the following search criteria can be used:

- DBID of agent: `FindConfigObject[CFGPerson, 'dbid:SomeDBID']`
- DBID of one of agent's logins: `FindConfigObject[CFGPerson, 'logindbid:SomeDBID']`
- EmployeeID of agent: `FindConfigObject[CFGPerson, 'employeeid:SomeEmployeeID']`
- Agents Login: `FindConfigObject[CFGPerson, 'switch:SomeSwitchName|login:SomeLogin']`

FindConfigObject Function Description

Parameters: TYPE: INTEGER
Valid Values: See FindConfigObject Returned Results table
Properties: LIST or variable (provide list of search criteria)
Return value type: LIST

This function returns information about a requested configuration object. You must specify the type of object for which to search (for example, CFGPlace) and the list-presenting search criteria. A valid set of search properties consists of either the name or a combination of the switch and number.
Examples:

- For CFGDN objects, name specifies an alias of the required DN, switch specifies the name of the switch to which the DN must belong, and number specifies this DN number.
- For CFGPlace object, name specifies the name of required place, switch specifies the name of the switch to which a DN (from among the DNs belonging to this Place) belongs, and number specifies the number of this DN.

The search criteria specifies a subset of the object properties, while the function provides the rest of the data. The search criteria must be a unique subset of the properties that identify the configuration object. See the following search criteria and results and also the table below:

Search Criteria: `FindConfigObject[CFGDN, 'name:2201_vit_sw2']` or `FindConfigObject[CFGDN, 'number:2201|switch:vit_sw2']`
Results: `"dbid:1122|name:Place_102_vit_sw2|tenantdbid:103|tenant:Vit|#1.number:102|#1.switch:vit_sw2|#1.type:2|#2.number:112|#2.switch:vit_sw2|#2.type: 1|dns:2".`

When search criteria is not based on dbid, but on names, the tenant is required for an object search. By default, the tenant is the one for the current interaction, but tenant also can be explicitly specified with extra keys: tenant or tenantdbid.

In addition to specifying a single Configuration object, you can also specify a collection of objects. In such cases, the search criteria must contain key all with value true. Additionally, the search criteria might contain extra filters on the Annex (only objects with this value of Annexes are returned). For example, annex.section,option1:value1|annex.section,option2:value2. When specifying a collection of objects, URS returns a reduced set of objects properties, all highlighted in bold, in the table below. Specifying a collection of objects is not supported for tenants, applications, folders, and enumerator values.

FindConfigObject Returned Results

Object Type	Search Key Combinations	Returned Properties
CFGSwitch	dbid,	**dbid**, type, link, **name**, tenant, tenantdbid, **tserverdbid**,

Object Type	Search Key Combinations	Returned Properties
	name	**tserver**, folders, annex, targetdata
CFGFDN	dbid, name switch or switchdbid+number	**dbid, type, number, name, switchdbid, switch**, tenant, tenantdbid, annex, targetdata
CFGPlace	dbid, name switch or switchdbid+number	**dbid, name**, tenant, tenantdbid, annex, targetdata, dns, folder, annex
CFGPerson	dbid, logindbid employeeid switch or switchdbid+login	**dbid, employeeid, firstname, lastname, username, email**, externalid, tenantdbid, tenant, **placedbid, skills**, logins, folders, annex
CFGTenant	dbid, name	dbid, name, annex,
CFGApplication	dbid, name switch or switchdbid	dbid, type, name, workdir, commandline, hostname, hostip, port, switch, servers, annex
CFGSkill	dbid, name	**dbid, name**, tenant, tenantdbid, annex
CFGAgentLogin	dbid, name switch or switchdbid+login	**dbid, login, switchdbid**, switch, tenant, tenantdbid, override, annex
CFGTransaction	dbid, type+name	**dbid, type, name**, tenant, tenantdbid, alias, description, annex
CFGStatDay	dbid, name	**dbid, name**, tenant, tenantdbid, dayofweek, day, starttime, endtime, min, max, target, rate, annex
CFGFolder	dbid	dbid, name, type, class, ownertype, ownerdbid, size, folders, annex
CFGEnumerator	dbid name	**dbid, type, name**, tenant, tenantdbid, description, **displayname**, annex
CFGEnumeratorValue	dbid enumeratordbid+name enumeratorname+name	dbid, enumeratordbid, name, description, displayname, isdefault, annex

GetInteractionAge

Parameters: None
Return value type: FLOAT

This function returns the time difference in seconds (with milliseconds precision) between the current moment in time and the age of the interaction timestamp.

PriorityTuning

Update the Warning on page 606 of the *Universal Routing 8.1 Reference Manual* as follows:

Warning! The interaction selection criteria associated with the `PriorityTuning` function (age of interaction, relative wait time (such as wait time in queue or predictive wait time), service objective risk factor, or any combination of these parameters) are only supported in a multi-URS environments where the same target might be selected by different instances of URSs if:

- all URS instances have the same value of option `use_service_objective` and
- all strategies running/served by URSes include the `PriorityTuning` function with the same parameters values across all strategies.

RequestRouter

See the `RequestRouter` function in Estimated Waiting Time Improvement.

run

Starting with release 8.1.400.39, Universal Routing adds support for the `run` function in skill and threshold expressions.

Parameters:

- for threshold expression:
  subroutine: STRING (Subroutine Name)

  param1: STRING

  param2: STRING
- for skill expression:
  subroutine: STRING (Subroutine Name)

  Agent: will be provided to subroutine by URS

  Virtual Queue: will be provided to subroutine by URS

param1: STRING

param2: STRING

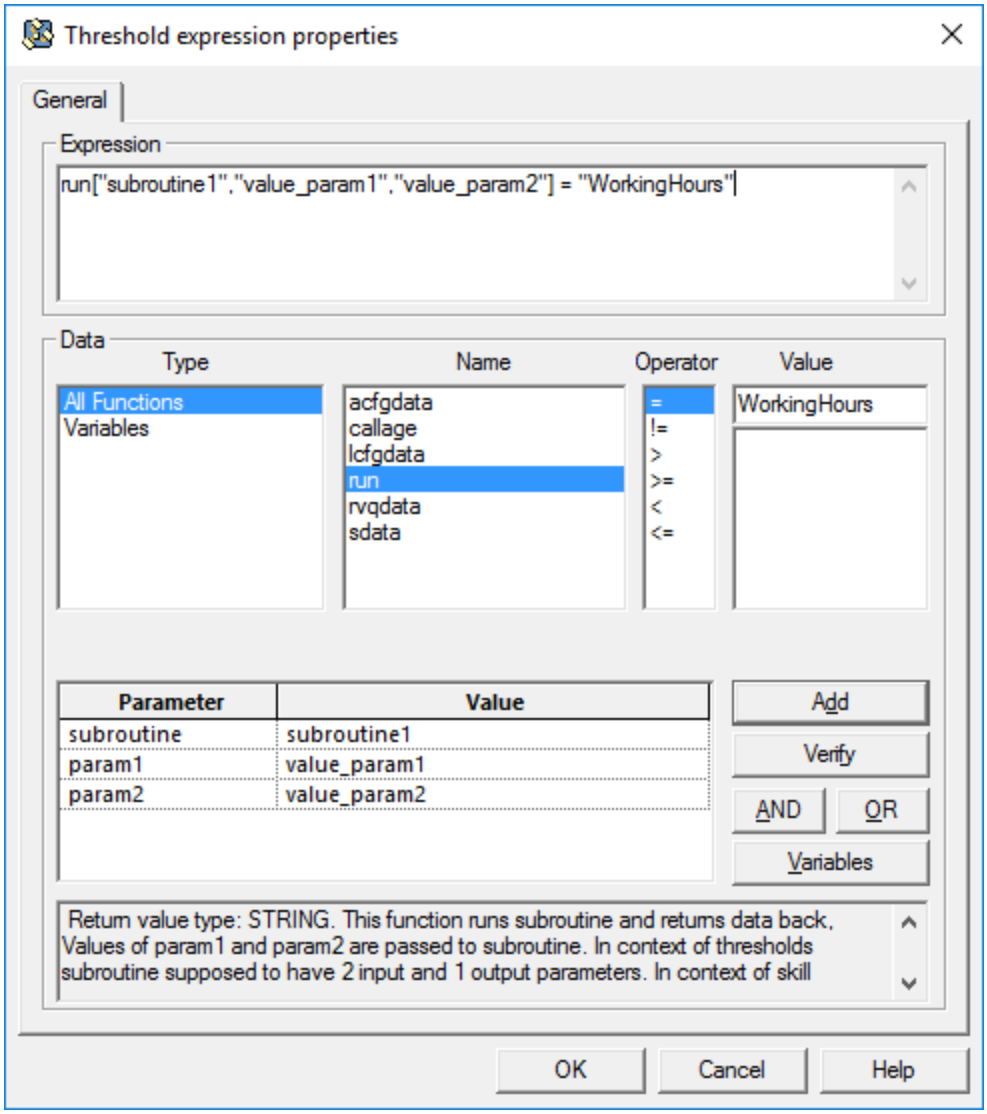
Return value type: STRING

This function executes a defined subroutine with provided parameters. The values of all parameters are passed to the subroutine as input parameters. The subroutine returns 1 output parameter value. The `run` function returns a STRING data type. If the expression is comparing a returned value with some other value, the comparison may not work. If needed, returned data can be explicitly converted to a number via the type-converting functions `num` or `int`.

When accessing the function from a threshold expression, IRD shows only subroutines with 2 input and 1 output parameters.

For example:

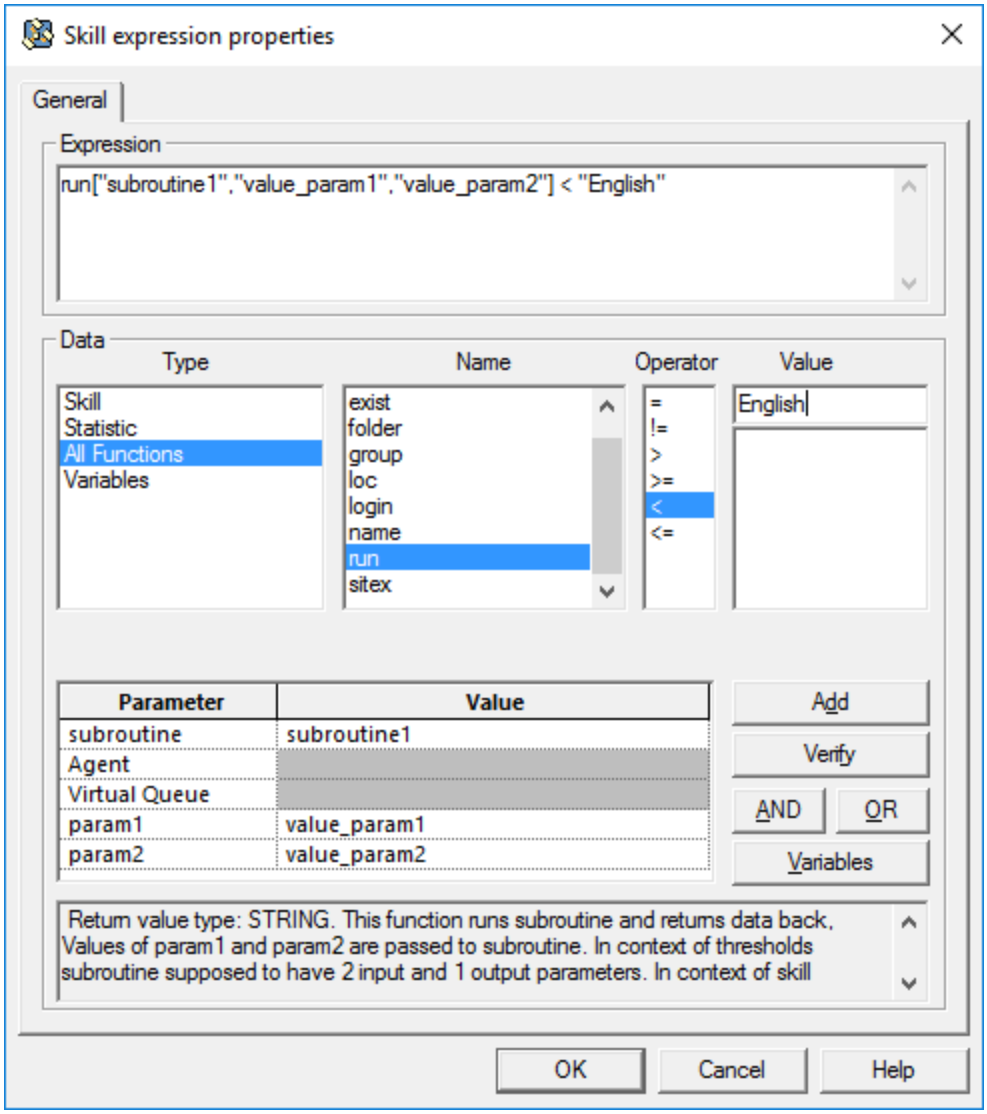
```
num[run[“subroutine1”,”value_param1”, ”value_param2”]] = “5”

run[“subroutine1”,”value_param1”, ”value_param2”] = “WorkingHours”
```



When accessing function from skill expression, IRD shows only subroutines that have 4 input and 1 output parameters.

For example:

```
run[“subroutine1”,”value_param1”, ”value_param2”]<”English”
```



Limitations:

A subroutine called by the run function cannot execute any waiting function, access external servers for some data, or invoke another subroutine. Any attempt to do so raises an error and terminates execution of skill or threshold expressions. All data that the subroutine is allowed to use must reside in URS memory, which means the subroutine can use data only from Configuration Server, Stat Server, interaction data, and data stored in URS memory.

SetIdealAgent

See the `SetIdealAgent` function in Using Agent Skills for Agents/Calls Prioritization.

StrAsciiTok

On page 583 of the *Universal Routing 8.1 Reference Manual*, the description of the string manipulation function StrAsciiTok should read "Every time the function StrAsciiTok is called, it stores in memory the index in String of the ***next*** character of the obtained substring." The manual incorrectly states "...it stores in memory the index in String of the ***last*** character...."

TargetListSelected

See the `TargetListSelected` function in Using Agent Skills for Agents/Calls Prioritization.

TargetState

Update `ready:1` in the TargetState function description, results—LIST Fields as follows:

`ready:1` The agent is considered (by URS) to be ready. If agent capacity is not used, then URS flags the agent as ready if his state is reported by Stat Server as `WaitForNextCall`. If agent capacity is used then the agent has no state, but only a list of medias, which results in URS flagging the agent as ready if this list of medias is not empty. By default, URS takes the ready flag into account when selecting agents, but it can be overwritten in the strategy by calling function `CheckAgentState` with an argument of `false`. This will cause URS to ignore the ready flag when looking for an available agent for the current call.

You can use the `FindConfigObject` and `TargetState` functions look up an agent name, the media channel logged into, and the agent's DN, based on an agent's Login ID or Employee ID.

Function `TargetState['EmplooyeeID.A']` returns agent readiness information, a list of available medias and DNs. This information is in addition to the returned results listed in the function description in the *Universal Routing 8.1 Reference Manual*.

Also see FindConfigObject.

transfer-to-agent

Location in Configuration Layer by precedence: T-Server, URS
Default value: `false`
Valid values: `true`, `false`,`never`
Value changes: take effect immediately

Instructs URS to request T-Server to transfer interactions from an IVR directly to a target agent instead of returning them to the routing point.

- `true` - allow transferring from IVR to agent.
- `false` - do not allow transferring from IVR to agent, except cases when IVR is Routing Point itself.

- never - do not allow transferring from IVR to agent in all cases.

Use this function where standard routing scenarios do not apply or cannot be used to override the usual method of routing a call to an agent. For example, there may be special hardware or reporting needs. Allows you to initiate a direct transfer to an agent using T-Library functionality. For more information on T-Library functions,see the *TLib Reference Guide*.

Selection Object Functions

To access these functions in IRD's Routing Design view, click **Routing** > **Selection** to open the the Selection properties dialog box. Cilck **Add Item**. Under **Type**, select **Skill.** Under **Name**, select **All Functions**.

cfgdata

Parameters:

- `folder: STRING` (Section name on the Annex tab of the `Person` object)
- `option: STRING` (Option name in the section on the Annex tab of the `Person` object)
- `default: FLOAT` (Preset value is returned, if the option is not found)

Return value type: `FLOAT`

This function returns a numeric value. If the agent's option is found, it returns `1`. If the option is not found and the default value is not specified, it returns `0`. If the option is not found and the default value is specified, the default value is returned.

exist

Parameters: `Skill Name: STRING` (Skill name)
Return value type: `FLOAT`

This function checks if an agent has the provided skill and is applied directly to skill names. It returns `1`, if an agent has the skill, or `0` otherwise.

folder

Parameters: `template: STRING` (Folder name in the Configuration Layer under Persons)
Return value type: `FLOAT`

This function checks if an agent is configured in a folder with a name that matches the specified template. It returns `1`, if there is a match, or `0`, otherwise.

group

Parameters: `Agent Group: STRING` (Agent Group name)
Return value type: `FLOAT`

This function checks if an agent belongs to the specified group. It returns 1, if yes, or 0, otherwise.

loc

Parameters: Switch Name: STRING (Switch name)
Return value type: FLOAT

This function verifies the agent location. It returns 1, if an agent has a DN belonging to the specified Switch, or 0, otherwise.

login

Parameters: media: STRING (Media name)
Return value type: FLOAT

This function returns 1, if an agent is logged in to provided media, or 0, otherwise. If media is set to any and an agent is logged in to at least one media, then 1 is returned.

name

Parameters: template: STRING (Agent employee ID template)
Return value type: FLOAT

This function checks if an agent name matches the specified template. It returns 1, if there is a match, or 0, otherwise.

sitex

Return value type: FLOAT

The sitex function verifies an agent's location (site). It returns 1 if the agent was logged in at the provided site and 0 otherwise. If the name of the site is set to this, then instead of checking, the function just returns the site name. The agent's site is usually provided by the agent himself through the site parameter in the Reasons attribute of the agent ready request. **Note:** In SIP cluster configuration, it is automatically populated by SIP Server.

String Manipulation Functions

The following string manipulation functions are available in IRD starting with version 8.1.400.39:

StrUTF8Encode

Parameters: String: STRING or variable (representing the string)
Return value type: STRING

This function performs encoding of the provided multibyte String into UTF8 format. Encoding is done based on current locale.

StrUTF8Decode

Parameters: `String: STRING` or variable (representing the string)
Return value type: `STRING`

This function performs decoding of the provided UTF8 String into multibyte format. Decoding is done based on current locale.

StrURLEncode

Parameters:

- `String: STRING` or variable (representing the string)
- `UTF8: INTEGER` or variable (representing the integer true/false value)

Return value type: `STRING`

This function performs URL Encoding of the input String. Additionally UTF8 conversion might also be applied before URL encoding if the UTF8 parameter is true (on any integer that is not 0). If UTF8 encoding is specified but fails, then URL encoding will be applied to the original string.

Note: URL encoding is the process of replacing unprintable characters within URLs, such as -, _, ., !, ~, *, ', (, and), with their corresponding hexadecimal code prefixed with %. The URL encoded value is safe to use as a value for the Web URL parameters.

StrURLDecode

Parameters:

- `String: STRING` or variable (representing the string)
- `UTF8: INTEGER` or variable (representing the integer true/false value)

Return value type: `STRING`

This function performs URL decoding from the input string. Additionally UTF8 conversion might also be applied after URL decoding if the UTF8 parameter is true (on any integer that is not 0). If UTF8 decoding is specified but fails, then just the result of the URL decoding process will be returned.

Note: URL decoding is the process of replacing the URL encoded characters with their corresponding original characters and replacing the + sign with a space.

Support for JavaScript

Beginning with release 8.1.400.67, URS supports execution of stand-alone JavaScript strategies and subroutines. The supported standard is **ECMA-327**, 3rd edition.

IRD now provides a possibility to write a part of or the entire strategy logic using the JavaScript

language in Macro or Script objects.

- To enable Script objects, navigate to the **Tools/Routing Design Options** dialog and in the **Views** tab select the **Routing Design/Scripts** checkbox.
- For Script objects, use the **Script/ecma** type.
- For Macro objects, select the **Complex macro** checkbox.
- Write the script source code in the **Definition** tab and always use the **Verify** button to check the validity of the script or macro you create.

The objects you create can be used in regular IRD strategies/subroutines. Script objects can also be used on their own.

To pass data from the calling strategy to the Script object:

Accessing of input parameters requires writing explicit code that will do it. Calling the strategy will put input parameters in a stack. In scripts, use the `GetInputParams()` function to retrieve the input parameters. Retrieved data will be removed from the stack, so calling the function more than once in a Script will result in an error. It is recommended to call the `GetInputParams()` function at the very beginning of the script.

To pass data from the Script object to the calling strategy:

Use the `SetOutputParams(object)` function to return output parameters.

> ### Important
> If the script is used as a subroutine, then the last executed command should be a Return(error_code); where error code is **0** if all OK and a non-zero error code if any error occurs. The following is a sample script with 2 input parameters (**i1** and **i2**) and 2 output parameters (**out1** and **out2**):
>
> ```
> var inp = GetInputParams();
> var res= new Object();
> res.out1= inp.i1+inp.i2;
> res.out2= inp.i1-inp.i2;
> SetOutputParams(res);
> Return(0);
> ```

To pass data to a Macro:

Macros are called as subroutines. All macro parameters are considered as input and passed automatically, there are no output params.

To pass data from a Macro to the calling strategy:

Data can be passed back to the calling strategy from Macro through the following:

- stack – combination of `Push(data)` and `data=Pop()`functions.
- wake up calls – combination of `WakeCallUp('', data)` and `data=WakeUpData['']` functions.

- dedicated returning – combination of `ReturnEx(error, data)` and data=`ReturnData[]`functions.

INTERACTION scope variables can be used anywhere (Scripts, Macros, strategies, subroutines) to pass data in any direction.

URS JavaScript Code/Functions Support:

- Standard **ECMA-327**, ECMAScript 3rd Edition Compact Profile.
- All operators must be ended with ";".
- Internal characters' presentation – multibyte (not UTF16).

Functions for JavaScript Strategies

All URS functions can be used. Functions are listed in the **compiler.dat** file and located in the IRD installation directory. The following functions have mostly been created for use in JavaScript strategies (though they also can be used in IRD strategies, if needed):

GetCallObject()

The `GetCallObject()` function returns an object presenting the current call. The properties of this object can be used to access (read and sometimes write) real call data. This call object has the following properties:

- **udata** or **userdata** – allows read/write access to call attached data
  - `GetCallObject().udata.abc = 123`; the same as `Update('{d}abc', 123)`
  - `X = GetCallObject().udata.abc`; the same as `X = UData ('abc')`
  - `X = GetCallObject().udata['*']`; returns all calls user data
  - `GetCallObject().udata.abc = null`; the same as `DeleteAttachedData ('abc')`
  - `GetCallObject().udata.abc = undefined`; the same as `DeleteAttachedData ('abc')`
  - `delete GetCallObject().udata.abc`; the same as `DeleteAttachedData ('abc')`
  - `GetCallObject().udata.abc+ = 5`; take current value of attached data, increment it by 5 and re-attach.
  - `GetCallObject().udata+ = {"abc":123, "def":"aaa"}`; the same as `Update('', '{d}abc:123|def:aaa')`
  - `GetCallObject().udata- = "abc"`; the same as `DeleteAttachedData ('abc')`
- **xdata** or **extensions** – allows read/write access to call extensions data
- **voice** – allows read access to following call data: `thisdn` or `_dest`, `_orig`, `acdqueue`, `type`, `ani`, `dnis` or `contactedaddr`, `ced`, `media` or `category`, `customerid`, `connid`, `g_uid`, `callid`, `trunk`, `media_server`, and `control_server`
- **location** – allows read access to following call data: `media_server` and `control_server`
- **data** – allows read/write access to the call's `INTERACTION` scope variables

The following functions for target selection are preferred for performing target selection from JavaScript strategies:

PutIntoQueue(id, virtual queue, priority, statistic, selection flag, target, . . .)

This function is similar to the SelectDN function with following differences:

- It allows explicit control of call queues with the first parameter. In the `SelectDN` function, `id` is selected by the compiler and they are all different within a strategy. Using the same `id` in different instances of the `PutIntoQueue` function will result in the call's requeueing (call targeting resulted from an older `PutIntoQueue` function will be eliminated and replaced with the new one). If `id` is set to **0** then the new `id`-less call's queue will always be created.
- It only puts the call into queue but does not try to select a target. The function returns the `refid` that can later be used for target selection.

RemoveFromQueue(id, virtual queue)

This function is similar to the ClearTargets function, but allows more explicit control over excluding a call from one or another queue.

- If the `virtual queue` is empty and `id` is not **0**, then the call will be removed only from the call queue in which it was placed by the `PutIntoQueue` function using the same `id`.
- If the `virtual queue` is empty and the `id` is **0**, then the call will be removed from all call queues as well as from all virtual queues (Virtual queue EventDiverted(redirected) will be distributed for all virtual queues call is in).
- If virtual queue is not empty the n call will be removed from the all call's queues associated with provided virtual queue as well as from this virtual queue (the virtual queue `EventDiverted`(redirected) event will be distributed for this virtual queue).

RemoveFromQueueN(refid)

This function is similar to the RemoveFromQueue function, but it removes the call from call queue identified by `refid` returned by the `PutIntoQueue` function. The call is not removed from any virtual queue.

SelectTargetFromQueue(refid, timeout)

This function is similar to the `SuspendForDN` function (and in case of *refid=0*, they are identical). The function places the call into a *waiting for targets from all queues calls are in* state for the duration of the provided timeout. Before doing that function explicitly tries to select target (by statistics) from the queue the call is in, pointed by the provided `refid` (return value of the `PutIntoQueue` function). If `refid` is **0**, then all call queues will be tried. The returned value is identical to the value provided by the SuspendForDN function.

RouteToTarget(target)

This function is a more advanced version of the `RouteCall` function. Similar to the `RouteCall` function it accepts a target returned by the `SuspendForDN` function or the `SelectTargetFromQueue` function and routes the call to the provided target. The returned value is identical to the value returned by the `RouteCall` function.

The main difference is that this function will try all means to route the call; that is, answer the call if

needed, return it back to the original Routing Point, and if the call is on an uninterruptible treatment or in a transition state, it will wait until routing is possible again. It is this function, and not the RouteCall function that IRD uses when custom routing is activated in Routing objects.

JavaScript support for Treatment functions

The function, TreatmentPlayAnnouncement, and other Treatment<TreatmentType> functions which are used to start and wait for treatment ending can not be used in JavaScript directly due to a name conflict with the same name constant.

The recommended way to use this function is by referring to it indirectly through a URS functional module. For example:

```
var funcTreatmentPlayAnouncement= FunctionalModule('[www.genesyslab.com/modules/
urs')['TreatmentPlayAnnouncement'];
funcTreatmentPlayAnouncement('LANGUAGE', 'English(US)', 'PROMPT.1.TEXT', 'http://127.1.1.1/
audios/bienvenida_001.wav');
```

Alternatively the function, StartTreatmentPlayAnnoncement (followed with SuspendForTreatmentEnd) can be used instead in this case. Effectively, the Treatment<TreatmentType> functions can be considered as shortcuts for combinations of two functions StartTreatment<TreatmentType> and SuspendForTreatmentEnd.

Notice how parameters in the above sample are passed to Treatment<TreatmentType> or StartTreatment<TreatmentType> functions, it is different from passing them to busy treatment functions. The parameters must be passed directly and as a sequence of keys and values: key, value, key, value, and so on.

Sample JavaScript Snippet

The following is a sample JS snippet where the call is routed to any available agent during working hours and if outside working hours, the not the strategy plays an appropriate treatment. The treatments to play are obtained from some a Web Service.

```
var funcTreatmentPlayApplication= FunctionalModule('[www.genesyslab.com/modules/
urs')['TreatmentPlayApplication'];

function RouteAnyAgent(greeting, busy)
{
   funcTreatmentPlayApplication('APP_ID',greeting);
   var queue= PutIntoQueue(1, '', 0, "StatAgentLoading", SelectMin, '?:2>1.GA');
   AddBusyTreatment(TreatmentPlayApplication, 'APP_ID:' + busy, 0);
   var target= SelectTargetFromQueue(queue, 60);
   if (!Failed())
     RouteToTarget(target);
}

function RoutetoAnyAgentWorkingHours(greeting, weekend, outoftime, busy)
{
    if (Date.getDay()==0 || Date.getDay()==6)
       funcTreatmentPlayApplication('APP_ID',weekend);
    else if (Date.getHours()<8 || Date.getHours()>18)
       funcTreatmentPlayApplication('APP_ID',outoftime);
    else
       RouteAnyAgent(greeting, busy);
}
```

```
x= ObjectFromJSON(
       GetHttpRequestInfo(1,
            StrFormat("http://myhost:myport/MyMessage?ani=~s&dnis=~s&connid=~s",
            ANI(), DNIS(), ConnID()), "", "", 0, "", ""));

if (!Failed())
      RoutetoAnyAgentWorkingHours(x.greeting, x.weekend , x.outoftime , x.busy);

Default();
```

Secure Connections Support Includes SNI Functionality

Starting with release 8.1.400.71, URS supports Server Name Indication (SNI) extension for TLS handshakes. As a result, HTTPS resources can be contacted from within URS, that is, using the Web Service block in IRD.

If the SNI functionality is activated, URS adds an extra parameter, `tls-target-name`, into the transport parameters of the connecting requests, set to the name of the host the web request is directed to.

The SNI functionality can be activated using one of the following methods:

- Set the `def_sni` option to `true`. For a description of the new option, refer to the New or Updated Option Descriptions topic.
- Specify `sni:true` in the **Hints** field of the IRD Web Service object. The **Hints** field is located under **TLS group** in the **Security** tab of the Web Server object.

> ### Important
> URS first uses the value from the IRD Web Service object. If that is absent, it uses the value set by the `def_sni` URS option.

Optimal Skill Update Mode

Starting with release 8.1.400.75, URS is enhanced to provide a possibility for more optimally updating skill expressions when part of the skill expression uses the `login` function. The optimal skill update mode is activated when the new URS option, `content_update_on_login`, is set to `false`. Setting it to `force`, which is the default value, overrides the enhancement and retains the behavior from the previous URS versions.

`content_update_on_login`
Location: `default` section of URS `Application` object
Default Value: `force`
Valid Values: `force`, `false`
Changes Take Effect: Immediately

Timeout to Wait before Sending Negative Response to Web Client

Beginning with version 8.1.400.78, URS allows users to specify a timeout to wait for before sending a negative response to a web client's request to perform an operation for an interaction that URS does not have. Timeout (in seconds) can be defined in the additional parameter, **maxdelay**, in the web request to URS or in the **call_sync_time** option.

call_sync_time
Location: `http` section of URS `Application` object
Default Value: `0`
Valid Values: `0` to `10`
Changes Take Effect: After restart

The value defined in the **call_sync_time** option is used if the **maxdelay** parameter is not specified.

For example, `urs/call/ConnID/func?name=Timeout¶ms=[120]&maxdelay=2`. Here, URS is requested to apply the `Timeout[120]` function to the interaction. If the requested interaction is not found, then URS will wait up to 2 seconds. If URS gets the requested interaction during the 2 seconds, it will apply the function to it. Else, it answers with an error.

> **Important**
>
> The maximum value allowed for the `maxdelay` parameter (and the `http/call_sync_time` option) is **10** seconds.

Multithreading Capability

Beginning with release 8.1.400.78, URS is enhanced with a multithreading capability to find matching agents who satisfy the conditions of the specified skill expression, in a given configuration . This improves URS performance in larger environments characterized by agent headcounts exceeding 10,000 or even 100,000 across locations.

A new configuration option, `mts`, is introduced to control the multithreading capability.

mts
Location: `default` section of URS `Application` object
Default Value: `0:0`
Valid Values: `0:0`, `1:0`, `1:1`
Changes Take Effect: Immediately

`0:0` - indicates multithreading is switched off.
`1:0` - indicates multithreading is turned on, but will be applied to expressions containing only skills. Skill expressions with statistics and functions are excluded in single threading mode.

`1:1` - multithreading is turned on and skill expressions with statistics and functions are also included for multithreaded processing.

A new console command, `mtskill`, is also provided for exploring the multithreading capability.

Format: `mtskill` <TenantName> <SkillExpression>.

As an output, URS provides 2 corresponding time intervals (in microseconds).

The following **limitations** are to be considered before turning on multithreading:

- Multithreading is justifiable only in very big environments.
- URS must run on very powerful hardware with multiple processors available for URS (running multithreading on single processor machine will slowdown URS).
- URS logging is disabled in multithreaded mode, while URS is updating skill expressions.
- Some skill expression functions, such as `run`, `group`, `folder`, and `tag`, have too big a footprint to be safely used in a multithreaded environment. Skill expressions containing these functions will always be executed in the single threaded mode irrespective of the value of the `mts` option.

Improved EWT Accuracy

EWT accuracy is determined by the difference between the EWT value given when a call enters a queue and the actual wait time that the particular call spent in the queue. When the difference is minimal it translates to a better EWT accuracy. EWT accuracy is impacted if calls of different types are placed into the same VQ. For example, if one VQ is used for both inbound, virtual callback (CB), and CB outbound calls, then EWT accuracy is low because outbound CB calls have some unique properties, such as:

- outbound CB calls are placed in VQ only for a short time of several seconds.
- outbound and virtual CB calls are two call representing the same interaction in a VQ. This creates double counting.
- outbound and virtual CB calls leave the queue at the same time breaking the EWT calculation model, which assumes that one call is distributed at a time and calls are distributed at a constant rate.

EWT accuracy can be increased if outbound CB calls are not taken into account for EWT calculation.

Beginning with release 8.1.400.83, URS provides more accurate EWT calculations for a Virtual Queue when interactions that are not intended to be routed by URS end up in the Virtual Queue. URS now provides the ability to mark and ignore such calls in EWT calculations. You can mark an interaction by setting its run time mode to 524288 and exclude them from EWT calculations.

To mark an interaction you can, use the `SetRunTimeMode` function within the IRD strategy. Or, it is set automatically if `EventRouteRequest` starting a strategy contains `AttributeCallType` with value **3** (outbound) and `AttributeUserData` has the `_CB_N_CALLBACK_ACCEPTED` key set to **1**.

Ignoring such marked calls in EWT calculations is controlled with the new configuration option, `lvq_ignore_duplicates`.

`lvq_ignore_duplicates`
Location: `default` section of URS `Application` object; can also be defined at the VQ level
Default Value: `false`
Valid Values: `true`, `false`
Changes Take Effect: Immediately

Setting the option to `true` ignores marked calls in EWT calculations.

If the option is set to `true` for a VQ and when an interaction marked as 524288 leaves the VQ:

- it will not change the distributing quitting rate from that VQ (its quitting will not go into an array of the last 32 quits). This improves the accuracy of `lvq` requests when `aqt=urs`.
- it will be skipped (not counted) when URS goes through (counts) all calls in the VQ to answer an `lvq` web request. This improves the accuracy of all types of `lvq` requests (`aqt=urs`, `urs2` or `stat`).

If the option is set to `false` for a VQ, then all interactions in that VQ (regulars or duplicates) will be counted for EWT calculations.

Log messages about sending VQ events (such as, *Queued*, *Diverted*) are now extended to indicate if URS counts or ignores the interaction in EWT calculations.

```
12:38:36.134_T_I_00820324eb06cefd \[14:02\] sending event 58 for vq
RBWM_UKFD_Ingress_VQ_EMEA (0 53-7 (0, i=349923 o=349904) 1633001911 170/32) (x)
```
(The x in the above sample message indicates if the interaction is considered by URS as a duplicate. If x is 1, it indicates that the interaction is a duplicate, and if x is 0, it indicates that the interaction is not a duplicate.)

Improved EWT Consistency

Beginning with version 8.1.400.84, URS is enhanced to improve consistency in EWT calculations for web requests that are interactionless and utilize average handling time provided by the default `lvq` URS method.

Improved consistency in EWT calculations is characterised by:

- absence of sharp changes (peaks or drops) in EWT values provided by URS.
- similarity in values of EWT returned by different URS instances in the same environment.

Default behavior

When obtaining EWT values for virtual queues (in response to the `lvq` web request) URS utilizes its own data, that is, calls that it places into virtual queues, targets that these calls are waiting for, and so on. As a result:

- it is possible that there is a sharp change in the provided EWT values if URS switches between sets of data used to calculate EWT.

- values returned by different URS instances might not be the same (if the data used by the different URS instances are also different).

To simulate a global queue perception in a distributed call center, multiple URS instances in the same environment must be able to provide consistent (ideally, the same) EWT value for the same VQ. A caller must get the same EWT value regardless of which data center the call lands in.

URS has the following two data sources for calculating the average handling time (time per call from the VQ) for interactions that are not defined in a web request (`max` is used in place of `connid`):

1. Internal queues created by URS when executing strategies.
2. One of the skill expressions from the internal queues that URS remembers (referred as the VQ Presenter). Internal queues are short lived objects that URS might dispose if not used. It is possible that all internal queues associated with a VQ are deleted. To be able to provide reasonable data even in such cases, URS remembers one of the skill expressions. URS selects the skill expression to remember based on which expression returns the highest number of agents. The identified skill expression represents the VQ when there are no internal queues and is referred to as the **VQ Presenter**.

Usually, URS tries to get data from the internal queues first and if at least one internal queue associated with the VQ exists, then, as listed above, data source 1 is used. If there are no internal queues, data source 2, that is, data provided by the VQ Presenter is used. A side effect to this approach is that URS can spontaneously switch between the two data sources (for interactionless web requests). This might result in the returned EWT values fluctuating frequently.

New enhanced behavior

In the new enhanced behaviour, URS uses the VQ Presenters as the primary source of data for calculating EWT even if internal queues exist.

To facilitate the new behaviour, URS ensures that:

- A VQ Presenter always exists.
  - Any skill expression that the strategy uses can be used as a VQ presenter, even if the skill expression contains statistics.
  - A skill expression identified as a VQ Presenter will not be deleted even if it has not been used for a long time.
- A VQ Presenter can be set or changed only in the following cases:
  - On VQ creation. If a VQ associated with an Agent Group as its origination DN, then this Agent Group will be set as the initial presenter for the VQ.
  - On creating a new internal queue associated with the VQ. At each such instance, sizes of the current VQ presenter and skill expression used by the internal queue are compared and the biggest one is selected as the new VQ presenter.
  - On placing any interaction into the VQ. Re-skilling of agents theoretically might result in reducing the size of the selected presenter and it will no longer be the biggest presenter. As a result, URS constantly rechecks the size of the current VQ presenters whenever a call is added into the VQ. The size of the current presenter and skill expression used by the call to enter the VQ are compared and the biggest one is selected.

Each time a VQ Presenter is changed, an entry is logged. For example, `15:27:23.757_M_I_ [10:85]`

`LVQ NameOfVQ presenter set: <?Agents5_10:>(21499ec7bb0) media=0.`

If different URS instances are executing the same set of strategies, it is likely that all those URS instances will also have the same VQ Presenters.

By deafult, the existing `lvq` method with `aqt=urs2` will continue to work as before. A new configuration option, `lvq_force_presenter`, is introduced to activate the new behaviour.

`lvq_force_presenter`
Location: `default` section of URS `Application` object; can also be defined at the VQ level with section name as `URS application name` or `__ROUTER__`
Default Value: `false`
Valid Values: `true`, `false`
Changes Take Effect: Immediately

Setting the option to `true` activates the new behaviour where a VQ presenter is used as a primary source of information to obtain the average handling time per call.

Reporting

To allow evaluation of the quality of the EWT calculations, URS can be enabled to collect (and report on) data about the estimated and actual waiting times for calls in a virtual queue.

- Every time an interaction enters a virtual queue the current EWTs are obtained and stored inside the interaction.
- Every time an interaction leaves the virtual queue the stored EWTs along with the actual time the interaction was waiting for is stored in the virtual queue. The virtual queue store information only about the latest interaction that quit the queue.

The `lvq` web request is extended to include this information as well as other statistical data that can be useful for tracing processing of calls in one or another virtual queue.

You can use the following requests to query data:

- `urs/call/max/lvq?tenant=TenantName&name=VirtualQueueName&filter=presenter`

- returns the skill expression/agent group used as the current presenter for the specified virtual queue.

- `urs/call/max/lvq?tenant=TenantName&name=VirtualQueueName&filter=trace&ewttrace[=N]`

- `filter=trace` returns tracing data for the specified VQ.
- `ewttrace` or `ewttrace=N` triggers the tracing mode for the specified VQ for the next `N` minutes (by default `N=3`).

For a virtual queue in tracing mode, URS collects extra data about the virtual queue (as permanent collection of such data takes a toll on performance). Additionally, for a VQ in tracing mode, URS records extra information in the log entries even if the `default/verbose` option is set to `false`.

When `filter=trace`, the following data is returned (note that when a value is unknown the field might not be returned):

Field	Description
time	current UTC time
lcalls_in	Local number of calls that have entered the VQ so far.
lcalls_out	Local number of calls that have exited the VQ so far.
lcalls	Local number of calls in the VQ (lcalls_in - lcalls_out).
rlcalls	Local number of real calls in the VQ (lcalls – duplicates).
calls	Global number of calls in the VQ (effectively StatCallsInQueue as returned by StatServer).
mrs	Multi-URS factor (used to convert local data into global (calls/lcalls).
rcalls	An estimate of the global number of real calls (rlcalls * mrs).
aqt_stat	Time per call according to StatServer (=StatExpectedWaitingTime/StatCallsInQueue).
ewt_stat	Waiting time according to StatServer (=aqt_stat * (rcalls+1)).
aqt_urs	Time per call according to URS or global quitting rate (local quiting rate / mrs).
ewt_urs	Waiting time according to URS (=aqt_urs * (rcalls+1)).
aqt_urs2	Time per call according to URS average handling time (calculated as per URS settings).
ewt_urs2	Waiting time according to URS (=aqt_urs2 * (rcalls+1)).
aqt_ursp	Same as aqt_urs2, but aht is calculated based on presenter.
ewt_ursp	Same as aqt_urs2, but aht is caluclated based on presenter.
xid	connid of latest call distributed into the VQ.
xtm	Latest call entry time into the VQ.
xewt_stat	StatServer based estimate of waiting time for the latest call (at the xtm time).
xewt_urs	URS quit rate based estimate of waiting time for latest call (at the xtm time).
xewt_urs2	URS average handling time based estimate of waiting time for latest call (at the xtm time).
xewt_ursp	URS average handling time for presenter based estimate of waiting time for the latest call (at the xtm time).

For a VQ in tracing mode the log message (logged when the call is distributed from the VQ) is as follows:

`12:36:04.200_M_I_03390320b4c930b0 [14:02] LVQ NameOfLVQ (58,1) ewts: xtm, xwt, xewt_stat, xewt_urs, xewt_urs2, xwt_ursp,` (along with some other data).

Note that if the VQ is not traced the above message might still be logged if the log level is set to 4 or 5, but the message will have no data for `xewt_urs2` and `xwt_ursp`.

You can follow one of the two patterns given below for tracing EWT for a VQ with the provided web requests:

1. Periodically (for example, once per minute) you can send URS the following web request, `urs/call/max/lvq?tenant=TenantName&name=VirtualQueueName&filter=trace&ewttrace`. Collect the output data for a period of time (say, a few hours) and visualize the output (for example, as an Excel spreadsheet).
2. Send the following web request to URS, `urs/call/max/lvq?tenant=TenantName&name=VirtualQueueName&filter=trace&ewttrace=180`. Collect URS logs for the next 180 minutes (3 hours), extract the related log messages from the logs and visualize them.

Limitations

The new behaviour is not necessarily better if compared with the default behaviour where URS relies on internal queues. That depends on how a specific solution has been implemented and how virtual queues are used in the solution. It is expected that the new behaviour will work good in cases of cascaded routing.

Also, URS cannot detect by itself, when the usage of one or another virtual queue changes sharply. For instance, solutions/strategies may start to use completely new skill expressions. When the usage of a virtual queue is changed, URS might still continue to use old virtual queues' presenters (if they happen to be bigger). To address such cases and avoid restarting URS to align virtual queue usage, the `lvqs` console command can be used with an extra optional parameter, `reset_presenter`.

For example: `lvqs TenantName VQName reset_presenter`
- where VQName is name of the virtual queue or *.
- All matched virtual queues will have their presenter updated (their presenters will be reset based on the current internal queues URS has for them).

From the Web API, the `lvqs` console command can be executed as, `urs/console?lvqs TenantName VQName reset_presenter`.

Estimated Waiting Time Improvement and URS Web API

Universal Routing improves its capability of calculating the estimated waiting time for a routing target.

- Starting with URS 8.1.400.25, you can use URS Web API methods to calculate a more precise estimated waiting time. With this release, the following two Web API methods support the improved estimated waiting time functionality: lvq and query.
- The existing InVQWaitTime function is enhanced to no longer require the association of agents with virtual queues and can be used in an environment with multiple URSs, queues, and multi-skilled agents.
- Starting with IRD 8.1.400.21, a new function, RequestRouter is available in IRD's Function object, which simplifies calling the Web API methods from routing strategies.

Method urs/call/connid/lvq

The `lvq` method allows more control over how estimated wait time is counted. The `lvq` method can also be applied to groups of virtual queues. If a virtual queue name ends with `.GQ` URS counts estimated waiting time and other parameters for the entire collection of virtual queues.

When the `lvq` method is invoked with the `aqt` parameter, the calculation includes:

- `stat`—identical with value counted by the `InVQWaitTime` function.
- `urs2`—Similar to the `InVQWaitTime` function but URS itself counts `AverageHandlingTime`. It does not matter whether or not a virtual queue is configured with an Agent Group.
- `urs`—Default value, Like `urs2`, but instead of `AverageHandlingTime`, URS uses average quitting time which is counted based on last 32 calls distributed from the queue.

Returned values for the `lvq` method depend on the input parameters supplied. Samples:

```
urs/call/connid/lvq/VirtualQueueName?aqt=stat
urs/call/connid/lvq/VirtualQueueName?aqt=urs
urs/call/connid/lvq/VirtualQueueName?aqt=urs2
```

> **Important**
>
> If `aqt=urs2`, URS calculates an agent's average handling time based on the agent's `CurrentState/CurrentTargetState` statistics. AHT is calculated based on the last 10 calls (for every media) that an agent processes. If `aqt=stat`, URS calculates an

> agent's average handling time using a sliding window of 10 minutes.

To view input parameters and output information: With URS running, open a browser and run the `help` method for `lvg`. The `help` method is described in the Universal Routing 8.1 Reference Manual, Appendix C, "Supported Methods." Example: `http://host:port/urs/help/call/lvq`

The `lvq` method can be called from a routing strategy with function RequestRouter.

Method urs/call/connid/query

The `query` method is described in the Universal Routing 8.1 Reference Manual, Appendix C, "Supported Methods." This method can be helpful when the use of virtual queues is not appropriate or not possible for calculating estimated wait time. For example, a virtual queue could include calls that might wait for totally different targets or have different thresholds. This method uses router's queues instead of virtual queues (all calls in router's queues wait for the same targets and have the same thresholds). Internally this method uses URS function `RvqData` to get aggregation information about all router's queues where a call resides.

When the `query` method is invoked, the calculation includes:

- `min_rvq_ewt`—Minimal estimated waiting time among all router's queues call is counted with using URS-provided `AverageHandlingTime`.
- `min_ewt`—Minimal estimated waiting time among all calls in router's queues is calculated by URS based on average quitting time.

Notes:

- Calculation of `AverageHandlingTime` that URS provides for router's queues does not consider multi-skilled agents or support multiple URSes.
- The `query` method has no input parameters.

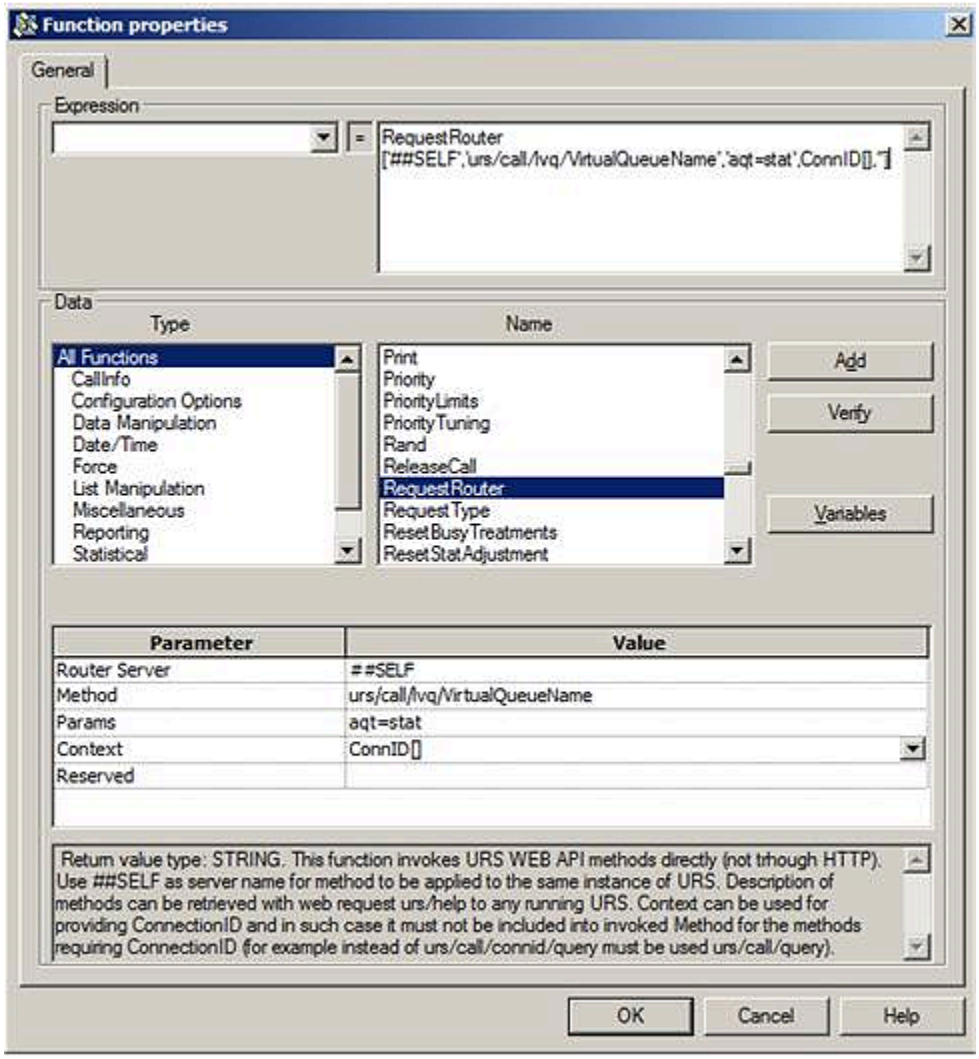
This method can be called from a routing strategy with function RequestRouter.

Function RequestRouter

This function simplifies using URS Web API methods from routing strategies. It allows you to directly invoke URS Web API methods, which can be any supported methods. Example:

```
RequestRouter['##SELF', 'urs/call/lvq/VirtualQueueName', 'aqt=stat',ConnID[], '' ]
where STRING RequestRouter[STRING RouterServer, STRING Method, STRING Params, STRING
Context, STRING Reserved]
```

A sample IRD Function properties dialog box using the `lvq` method is shown below.



Parameters:
Router Server: Uses ##SELF as the server name for the method to be applied to the same instance of a running URS. Default value: '##SELF'.
Method: With IRD release 8.1.400.21, supported methods for use with this function are `urs/call/lvq` or `urs/call/query`.
Params: If the method must have, but does not contain parameters, they can be provided separately here.
Context: If the method must have, but does not include an Interaction ID, it can be provided here. Can also be used for providing a `ConnectionID`, in which case it must not be included into invoked methods requiring `ConnectionID`. For example, instead of `urs/call/connid/query`, URS just uses `urs/call/query`.
Reserved: Not used in IRD release 8.1.400.21.
Return Value Type: `STRING`

InVQWaitTime Enhancement

Starting with URS release 8.1.400.25, the `InVQWaitTime` function is enhanced to no longer require the association of virtual queues with Agent Groups.

- If a virtual queue is associated with some Agent Group, then URS uses Stat Server's `min(max(StatLoadBalance, 0),10000)` statistic for the virtual queue and adjusts it relative to call position in the queue.
- If a virtual queue is not associated with an Agent Group, then URS calculates an `AverageHandlingTime` for all agents from the virtual queue that URS considers as a target and multiplies the calculated result by the call position in the queue. While calculating an `AverageHandlingTime`, URS attempts to consider multi-skilled agents and multiple URSes.

> **Important**
>
> For more information on Expected Wait Time (EWT) and its related statistics, functions, and methods, refer to the White Paper on EWT.

IRD Localization

Starting with Interaction Routing Designer 8.1.400.22, the IRD interface can be adjusted for the user's language by installing a Language Pack on top of the base installation and by setting a language preference. Every time a Language Pack is installed, the `lang` folder in the installation directory is modified to insert the localized resources, such as the text strings that appear on the screen. Each logged in user can select their preferred language in the Windows Region and Language dialog box.

When IRD starts, it attempts to render the screens in the user's preferred language. If the Language Pack is unavailable, the IRD interface will default to the English language.

Available Language Pack

- Japanese Language Pack.

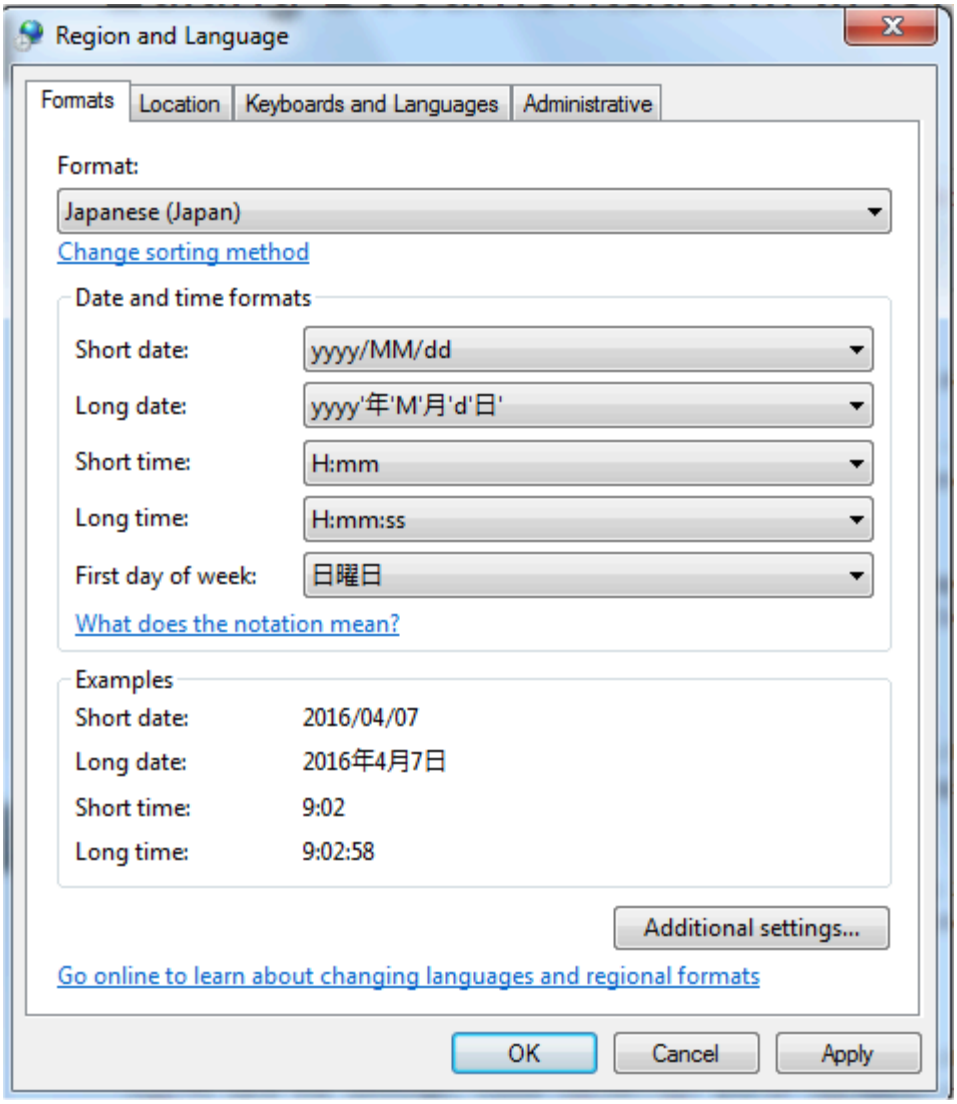
Installing a Language Pack on Windows

1. Install IRD 8.1.400.22.
2. Contact your Genesys representative for the Language Pack.
3. Run setup for the Language Pack you want to install.
4. Follow the steps in the Language Pack installation wizard.

Setting the Language Preference

Specify a language preference in the Windows Region and Language dialog box.

1. On the host where IRD is installed, from the Control Panel, open the Region and Language dialog box. The figure below shows the dialog box for Windows 7. The dialog box may appear slightly different for your version of Windows.
2. On the **Formats** tab, in the **Formats** section, select the language and click **OK**. The figure below shows the dialog box after selecting **Japanese**.


Region and Language
Formats
Location
Keyboards and Languages
Administrative
Format:
Japanese (Japan)
Change sorting method
Date and time formats
Short date:
yyyy/MM/dd
Long date:
yyyy'年'M'月'd'日'
Short time:
H:mm
Long time:
H:mm:ss
First day of week:
日曜日
What does the notation mean?
Examples
Short date:
2016/04/07
Long date:
2016年4月7日
Short time:
9:02
Long time:
9:02:58
Additional settings...
Go online to learn about changing languages and regional formats
OK
Cancel
Apply

Graceful shutdown

Staring with Release 8.1.400.37, URS supports graceful shutdown, the process for uninterrupted handling of interactions.

How it works

When a graceful shutdown starts, Management Layer still sees the primary/backup pair relationship between the two URSs. To facilitate continuous processing of calls, internally those URSs turn into a cluster of two independent URSs, where HA functionality is no longer available.

Prerequisites

The prerequisites for a graceful shutdown are:

- A primary/backup pair of URSs must be configured and running. Graceful shutdown does not apply to single (backup-less) URS instances.
- Genesys recommends configuring agent reservation before using graceful shutdown, since the graceful shutdown process temporarily turns a primary/backup pair of URSs into a cluster of two independent URSs. For more information, see the `agent_reservation` option in the *Universal Routing 8.1 Reference Manual*.
- If Orchestration Server-controlled interactions are present, verify that the current backup URS is aware of all such calls. To verify, send a web request to the primary URS: `urs/console?backup`. The response should be `name(host:port,socket);000000000000`. If a non-null connection ID appears after the semicolon, it is not safe to start the shutdown process, as some ORS-controlled interactions could be lost.

Performing a graceful shutdown

You can start graceful shutdown with either of the following:

- A web request to the current primary URS: `urs/console?rollover`, or
- In Genesys Administrator, initiate the graceful shutdown procedure from the URS Application object. Note that when you start graceful shutdown through GA, URS ignores the request if no proper backup exists, effectively resulting in an ungraceful shutdown.

During a graceful shutdown, the current primary URS:

- rejects processing requests of new voice interactions (from T-Servers or Web clients).
- stops processing of any multimedia interactions (returns them to interaction queues) and ORS-

controlled voice interactions.

- imitates switching into backup mode of operation, forcing ORS (if any) to reconnect to the current backup node, which is switching to primary mode.
- finishes processing all existing voice interactions, then exits.

The current backup URS:

- switches into primary mode of operation.
- starts processing new voice calls and all multimedia interactions.

Support of HTTP Proxies

Starting with 8.1.400.27, URS provides support of HTTP Proxies for an "https://" type of request. HTTP Proxies are specified either in the request itself or globally at the URS `Application` level, in the `web` section.

Starting with 8.1.400.45, URS can use the HTTP CONNECT method to establish a secure tunnel between URS and the web server when accessing a secure web server through a proxy server. The option proxy_use_connect controls the connection method.

def_https_proxy_host
Location in Configuration Layer by precedence: `web` section of URS
Default Value: An empty string
Valid Values: Any valid host name
Changes Take Effect: After restart

This option specifies the HTTP Proxy host for an "https://" type of connection.

def_https_proxy_port
Location in Configuration Layer by precedence: `web` section of URS
Default Value: An empty string
Valid Values: TCP port
Changes Take Effect: After restart

This option specifies the HTTP Proxy port for an "https://" type of connection.

URS checks these options only if a request does not contain the HTTP Proxy host and port specified. HTTP Proxy for "https://" must be fully trusted and support secure connection on the Proxy port.

proxy_use_connect
Location in Configuration Layer by precedence: `web` section of URS
Default Value: `true`
Valid Values: `true, false`
Changes Take Effect: After restart

This option specifies the connection method to a secure web server through a HTTP proxy server.

- A value of *true* uses the HTTP CONNECT method. URS communicates with web servers through HTTP proxy and performs TLS negotiations directly with the web server.
- A value of *false* uses the legacy method (not recommended). URS communicates with web servers through HTTP proxy and performs TLS negotiations with the proxy server.

Updates to Existing 8.1.4 Documentation

Below are documentation updates for the *Universal Routing 8.1 Reference Manual*:

Chapter Number	Chapter Name	Update Summary
2	IRD Objects	Classification Object Update Classification Server 8.5 and Strategy Execution Identify Contact Object Update Web Service Object Update Workbin Owner Types and Compatible Target Types Using Variables - Find Interaction Object Also see the Security section below.
3	URS/IRD Functions	Additional Data Returned by SelectDN New or Updated Function Descriptions
4	URS Options	New or Updated Configuration Option Descriptions
Appendix C	URS Methods	Estimated Wait Time Improvement Using URS Web API Additional Information on HTTP Report Method
Various	Other Updates	Maximum Characters for Combined Length of Target and Stat Server Name Distribution of Multimedia Interactions During Shutdown or Backup Mode Removal of 9999 License Limit Maximum Length Limitation for Text Field on Web Service Object Corresponding Genesys Administrator (GA) Screenshots for Old Configuration Manager (CME) Screenshots
Security		Secure communication with Workforce Management Server

Chapter Number	Chapter Name	Update Summary
		using the standard Transport Layer Security (TLS) protocol. See the Transport Layer Security section in the *Genesys Security Deployment Guide* for details. IRD supports TLS 1.2. The Security TLS Protocol control in the Web Service object contains a new value, TLSv12. IRD supports communication with Message Server through a secure port. To configure a secure connection to Message Server, refer to the *Genesys Security Deployment Guide*. TLS Support: When connecting to a server through a security channel (TLS), URS relies on Genesys Security Layer. Server-side and Mutual authentication are supported as specified in the Secure Connections (TLS) section of the Genesys Security Deployment Guide.

New or Updated IRD Object Descriptions

Add/update the following IRD object descriptions in the *Universal Routing 8.1 Reference Manual*:

Classification Object Update

Chapter 2: Interaction Routing Designer Objects: In the section on the Classification object, Table 34: User Data Example, is updated with additional information in the form of notes. The table should now appear as follows:

Parameter	Value
CtgId	00001a05F5U900QW Note: This value is the Universal Contact Server identifier for the selected category (category having the maximum relevancy).
CtgRelevancy	95 Note: This is relevancy of the selected category.
CtgName	Cooking Note: This is name of the selected category.
CtgId_00001a05F5U900QW	95 Note: This identifier, as well as those below, are other categories (together with their relevancies) returned by Classification Server as a result of classification.
CtgId_00001a05F5U900QX	85
CtgId_00001a05F5U900QY	75
CtgId_00001a05F5U900QZ	65

Classification Server 8.5 and Strategy Execution

When working in a multimedia deployment, if a screening rule is not found:

- Classification Server 8.1 reports an `ExternalServiceResponse` message indicating that there is no match for a screening rule and strategy execution continue through the object's green port.
- Classification Server 8.5 reports an `ExternalServiceFault` message with `FaultCode 904` indicating that a screening rule is not found and strategy execution continue through the object's red port.

Identify Contact Object Update

Chapter 2: Interaction Routing Designer Objects: In the Identify Contact object, General Tab, the Update User Data definition should now read as follows:

Universal Contact Server returns contact attribute values only when a unique contact is found/created. Select Update User Data in the Identify Contact object to have the contact attribute values returned by the Universal Contact Server be part of the User Data of the response. When the Update User Data property is selected AND if a unique contact is identified or created, Universal Contact Server returns the contact attribute values in the User Data part of its response. If a unique contact is identified or created and the Update User Data property is not set, the contact attribute values will be returned in the parameter part of the ESP response.

Web Service Object Update

Starting with IRD Release 8.1.400.15, security parameters of the Web Service object are extended to include Proxy Server parameters—`Host` and `Port`—to enable execution of Web requests through HTTP Proxy Server. New parameters are supported by URS version 8.1.400.16 or later.

Security parameters of the Web Service object are extended to include the following fields:

- `Client authentication`—To explicitly enable/disable authentication of a client (URS) when a corresponding Web request is involved. The allowed values for this field are *true* or *false* / *1* or *0*.
- `Protocol`—To explicitly specify the security protocol to be used for the specific Web request. It applies only to UNIX and overrides the URS `def_sec_protocol` option setting.

New parameters are supported by URS version 8.1.400.16 or later.

Usage of Prefixes to Define Value Types in KVList

Any URS function creating or updating a KVList allows to define the type of values by adding a prefix to the key name.

Allowed prefixes:

- `{s}` for string
- `{d}` for integer

To define type in a nested list, for example, `key1.key2.key3`, the prefix should be provided before the innermost key: `key1.key2.{d}key3`.

Workbin Owner Types and Compatible Target Types

The list of Targets selected in the **Workbin** object depend strictly on the selected **Type of Workbin Owner**. Changing the Workbin Owner Type may result in clearing the list of Targets specified in the corresponding tab.

The following table lists the types of workbin owners and their compatible target types:

Workbin Owner Type	Compatible Target Types
Agent	Agent
	Agent Group
	Skill
	Variable
Agent Group	Agent Group
	Variable
Place	Place
	Place Group
	Variable
Place Group	Place Group
	Variable

Using Variables in the Find Interaction Object

Given below is some additional information on using variables in **Condition** tab of the **Find Interaction** object, explained on page 228 of the Universal Routing 8.1 Reference Manual (Chapter 2: Interaction Routing Designer Objects, Multimedia Objects section):

You can use character sets {|...|} or {?...?} to denote an expression that URS must evaluate. For example,

`var\_contact = {|contactId|} and MediaType = "email".`

The result is put into quotes when using `{|...|}` and is displayed as is, that is not put into quotes, when using `{?...?}`.

For example,

- `{|contactId|}` is displayed as '**0002Pa5U45EP0015**'.
- `{?contactId?}` is displayed **0002Pa5U45EP0015**.

Additional Data Returned by SelectDN

Update the SelectDN list of returned keys in the *Universal Routing 8.1 Reference Manual* as follows:

If the function SelectDN succeeds in selecting an available target, then, in addition to the pair `return:ok`, the list returned by the function will contain the following keys:

- `target_name`—the name of the selected target from the list.
- `target_location`—the location of the selected target from the list.
- `target_type`—the type of the selected target from the list.
- `vq`—the virtual queue name specified as a parameter of the function SelectDN.
- `dn`—an available DN of the selected target; this value is reported by Stat Server. If the target is an agent and the option `use_agentid` is set to true, then the value of this key will not be a DN but the Employee ID of the agent.
- `switch`—the switch where the DN provided for the selected target is located; this value is also reported by Stat Server.
- `agent`—the Employee ID of the agent selected; this value, reported by Stat Server, is present only if the target is of type Agent (`.A`) or Group of Agents (`.GA`).
- `place`—the name of the place selected; this value, reported by Stat Server, is present only if the target is of type Agent (`.A`), Group of Agents (`.GA`), Agent Place (`.AP`), Group of Places (`.GP`) or Campaign Group.
- `rdnan`—available DN of the selected target; this value is reported by Stat Server. As opposed to `dn` key. this value is always real agent's DN.
- `stat_value`—if target was selected based on some statistic then this key will be included and provide value of this statistic.
- `priority`—if priority of call in queue from which target was selected is not `0`. then this value will be included and provide this priority value.
- `*mismatch`—if for target selection best fit factor was used, then this value will be included and provide `mismatch` of the selected target from the ideal one.
- Also if cost type routing was used, then the list can contain a set of keys (`cost.type`, `cost.cost1`. `cost.cost2`, `cost.orig`, `cost.dest`, `cost.contract`, `cost.table`. `cost.day`, `cost.time`, `cost.interval`, `cost.va`) reflecting different aspects of cost information for this specific call and selected target.

New or Updated Function Descriptions

The following functions have been either newly added or updated after the *Universal Routing 8.1 Reference Manual* was last published. Unless otherwise noted, these functions are available in IRD's Function object. Other functions are located in IRD's Selection object.

ExcludeAgents

Parameters: Agents: STRING (comma-separated list of agent IDs or variable)
Return value type: STRING

This function instructs URS not to route interactions to any agent on the specified list of agents. Parameter Agents is comma-separated list of agent IDs. Function returns the previous list of excluded agents.

Previous to 7.6, if a target was selected (if it was ready according to Stat Server and reserved) and then excluded from the list of valid targets using the ExcludeAgents function, this target was not actually excluded. Starting with 7.6, the ExcludeAgents function does exclude the agent in the above scenario.

Note: When URS executes the ExcludeAgents function for an interaction, the URS-provided list of excluded agents will be applied to the current or any future Selection objects. The effect of the ExcludeAgents execution can be cancelled only by the execution of another ExcludeAgents function or if URS stops this interaction processing.

Warning! Function ExcludeAgents affects IVR targets.

FindConfigObject

As of 11/24/15, the existing FindConfigObject function is extended to support additional object types as shown in the FindConfigObject Returned Results table below.

Look Up Agent Name, Media logged Into and DN, from Agent Login ID and by Employee ID

Function FindConfigObject with object type CFGPerson and function TargetState can be used to look up an agent name, the media channel logged into, and the agent's DN, based on an agent's Login ID or Employee ID.

Function TargetState['EmployeeID.A'] with input parameter agent EmployeeID returns agent readiness information, a list of available medias and DNs.

For example, to use FindConfigObject to find agent (Person) information, the following search criteria can be used:

- DBID of agent: `FindConfigObject[CFGPerson, 'dbid:SomeDBID']`
- DBID of one of agent's logins: `FindConfigObject[CFGPerson, 'logindbid:SomeDBID']`
- EmployeeID of agent: `FindConfigObject[CFGPerson, 'employeeid:SomeEmployeeID']`
- Agents Login: `FindConfigObject[CFGPerson, 'switch:SomeSwitchName|login:SomeLogin']`

FindConfigObject Function Description

Parameters: TYPE: INTEGER
Valid Values: See FindConfigObject Returned Results table
Properties: LIST or variable (provide list of search criteria)
Return value type: LIST

This function returns information about a requested configuration object. You must specify the type of object for which to search (for example, CFGPlace) and the list-presenting search criteria. A valid set of search properties consists of either the name or a combination of the switch and number.
Examples:

- For CFGDN objects, name specifies an alias of the required DN, switch specifies the name of the switch to which the DN must belong, and number specifies this DN number.
- For CFGPlace object, name specifies the name of required place, switch specifies the name of the switch to which a DN (from among the DNs belonging to this Place) belongs, and number specifies the number of this DN.

The search criteria specifies a subset of the object properties, while the function provides the rest of the data. The search criteria must be a unique subset of the properties that identify the configuration object. See the following search criteria and results and also the table below:

Search Criteria: `FindConfigObject[CFGDN, 'name:2201_vit_sw2']` or `FindConfigObject[CFGDN, 'number:2201|switch:vit_sw2']`
Results: `"dbid:1122|name:Place_102_vit_sw2|tenantdbid:103|tenant:Vit|#1.number:102|#1.switch:vit_sw2|#1.type:2|#2.number:112|#2.switch:vit_sw2|#2.type: 1|dns:2".`

When search criteria is not based on dbid, but on names, the tenant is required for an object search. By default, the tenant is the one for the current interaction, but tenant also can be explicitly specified with extra keys: tenant or tenantdbid.

In addition to specifying a single Configuration object, you can also specify a collection of objects. In such cases, the search criteria must contain key all with value true. Additionally, the search criteria might contain extra filters on the Annex (only objects with this value of Annexes are returned). For example, annex.section,option1:value1|annex.section,option2:value2. When specifying a collection of objects, URS returns a reduced set of objects properties, all highlighted in bold, in the table below. Specifying a collection of objects is not supported for tenants, applications, folders, and enumerator values.

FindConfigObject Returned Results

Object Type	Search Key Combinations	Returned Properties
CFGSwitch	dbid,	**dbid**, type, link, **name**, tenant, tenantdbid, **tserverdbid**,

Object Type	Search Key Combinations	Returned Properties
	name	**tserver**, folders, annex, targetdata
CFGFDN	dbid, name switch or switchdbid+number	**dbid, type, number, name, switchdbid, switch**, tenant, tenantdbid, annex, targetdata
CFGPlace	dbid, name switch or switchdbid+number	**dbid, name**, tenant, tenantdbid, annex, targetdata, dns, folder, annex
CFGPerson	dbid, logindbid employeeid switch or switchdbid+login	**dbid, employeeid, firstname, lastname, username, email**, externalid, tenantdbid, tenant, **placedbid, skills**, logins, folders, annex
CFGTenant	dbid, name	dbid, name, annex,
CFGApplication	dbid, name switch or switchdbid	dbid, type, name, workdir, commandline, hostname, hostip, port, switch, servers, annex
CFGSkill	dbid, name	**dbid, name**, tenant, tenantdbid, annex
CFGAgentLogin	dbid, name switch or switchdbid+login	**dbid, login, switchdbid**, switch, tenant, tenantdbid, override, annex
CFGTransaction	dbid, type+name	**dbid, type, name**, tenant, tenantdbid, alias, description, annex
CFGStatDay	dbid, name	**dbid, name**, tenant, tenantdbid, dayofweek, day, starttime, endtime, min, max, target, rate, annex
CFGFolder	dbid	dbid, name, type, class, ownertype, ownerdbid, size, folders, annex
CFGEnumerator	dbid name	**dbid, type, name**, tenant, tenantdbid, description, **displayname**, annex
CFGEnumeratorValue	dbid enumeratordbid+name enumeratorname+name	dbid, enumeratordbid, name, description, displayname, isdefault, annex

GetInteractionAge

Parameters: None
Return value type: FLOAT

This function returns the time difference in seconds (with milliseconds precision) between the current moment in time and the age of the interaction timestamp.

PriorityTuning

Update the Warning on page 606 of the *Universal Routing 8.1 Reference Manual* as follows:

Warning! The interaction selection criteria associated with the `PriorityTuning` function (age of interaction, relative wait time (such as wait time in queue or predictive wait time), service objective risk factor, or any combination of these parameters) are only supported in a multi-URS environments where the same target might be selected by different instances of URSs if:

- all URS instances have the same value of option `use_service_objective` and
- all strategies running/served by URSes include the `PriorityTuning` function with the same parameters values across all strategies.

RequestRouter

See the `RequestRouter` function in Estimated Waiting Time Improvement.

run

Starting with release 8.1.400.39, Universal Routing adds support for the `run` function in skill and threshold expressions.

Parameters:

- for threshold expression:
  subroutine: STRING (Subroutine Name)

  param1: STRING

  param2: STRING
- for skill expression:
  subroutine: STRING (Subroutine Name)

  Agent: will be provided to subroutine by URS

  Virtual Queue: will be provided to subroutine by URS

param1: STRING

param2: STRING

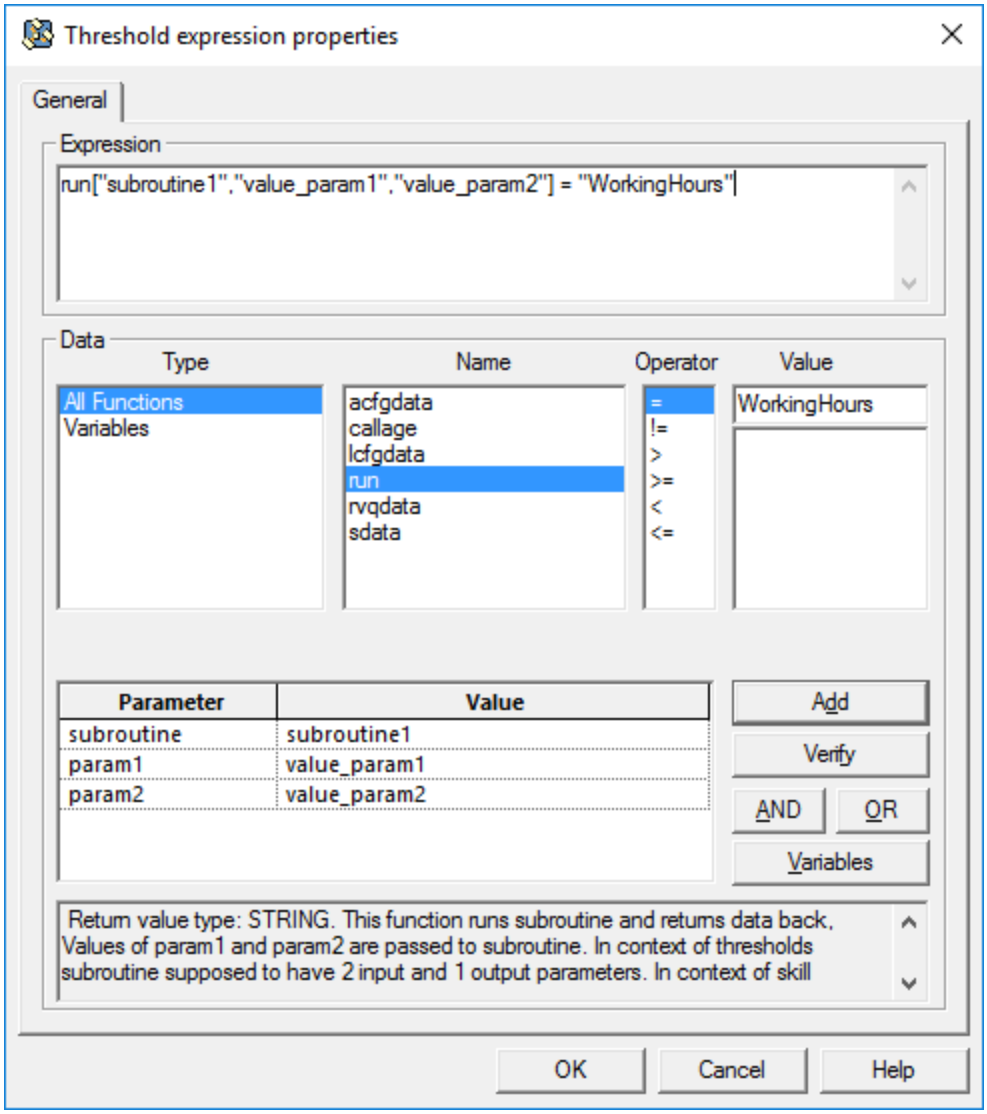
Return value type: STRING

This function executes a defined subroutine with provided parameters. The values of all parameters are passed to the subroutine as input parameters. The subroutine returns 1 output parameter value. The `run` function returns a STRING data type. If the expression is comparing a returned value with some other value, the comparison may not work. If needed, returned data can be explicitly converted to a number via the type-converting functions `num` or `int`.

When accessing the function from a threshold expression, IRD shows only subroutines with 2 input and 1 output parameters.

For example:

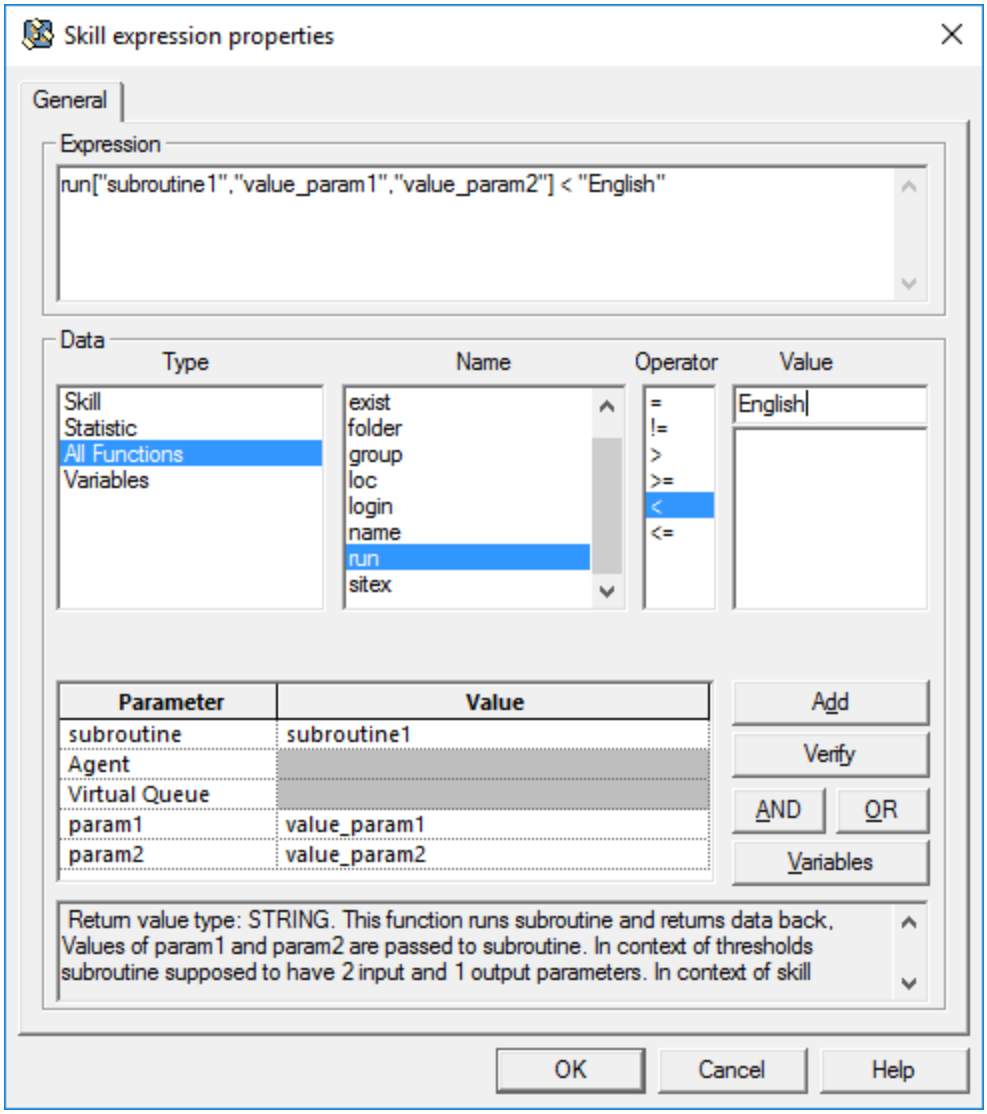
```
num[run[“subroutine1”,”value_param1”, ”value_param2”]] = “5”

run[“subroutine1”,”value_param1”, ”value_param2”] = “WorkingHours”
```



When accessing function from skill expression, IRD shows only subroutines that have 4 input and 1 output parameters.

For example:

```
run[“subroutine1”,”value_param1”, ”value_param2”]<”English”
```



Limitations:

A subroutine called by the run function cannot execute any waiting function, access external servers for some data, or invoke another subroutine. Any attempt to do so raises an error and terminates execution of skill or threshold expressions. All data that the subroutine is allowed to use must reside in URS memory, which means the subroutine can use data only from Configuration Server, Stat Server, interaction data, and data stored in URS memory.

SetIdealAgent

See the `SetIdealAgent` function in Using Agent Skills for Agents/Calls Prioritization.

StrAsciiTok

On page 583 of the *Universal Routing 8.1 Reference Manual*, the description of the string manipulation function StrAsciiTok should read "Every time the function StrAsciiTok is called, it stores in memory the index in String of the ***next*** character of the obtained substring." The manual incorrectly states "...it stores in memory the index in String of the ***last*** character...."

TargetListSelected

See the `TargetListSelected` function in Using Agent Skills for Agents/Calls Prioritization.

TargetState

Update `ready:1` in the TargetState function description, results—LIST Fields as follows:

`ready:1` The agent is considered (by URS) to be ready. If agent capacity is not used, then URS flags the agent as ready if his state is reported by Stat Server as `WaitForNextCall`. If agent capacity is used then the agent has no state, but only a list of medias, which results in URS flagging the agent as ready if this list of medias is not empty. By default, URS takes the ready flag into account when selecting agents, but it can be overwritten in the strategy by calling function `CheckAgentState` with an argument of `false`. This will cause URS to ignore the ready flag when looking for an available agent for the current call.

You can use the `FindConfigObject` and `TargetState` functions look up an agent name, the media channel logged into, and the agent's DN, based on an agent's Login ID or Employee ID.

Function `TargetState['EmplooyeeID.A']` returns agent readiness information, a list of available medias and DNs. This information is in addition to the returned results listed in the function description in the *Universal Routing 8.1 Reference Manual*.

Also see FindConfigObject.

transfer-to-agent

Location in Configuration Layer by precedence: T-Server, URS
Default value: `false`
Valid values: `true`, `false`,`never`
Value changes: take effect immediately

Instructs URS to request T-Server to transfer interactions from an IVR directly to a target agent instead of returning them to the routing point.

- `true` - allow transferring from IVR to agent.
- `false` - do not allow transferring from IVR to agent, except cases when IVR is Routing Point itself.

- never - do not allow transferring from IVR to agent in all cases.

Use this function where standard routing scenarios do not apply or cannot be used to override the usual method of routing a call to an agent. For example, there may be special hardware or reporting needs. Allows you to initiate a direct transfer to an agent using T-Library functionality. For more information on T-Library functions,see the *TLib Reference Guide*.

Selection Object Functions

To access these functions in IRD's Routing Design view, click **Routing** > **Selection** to open the the Selection properties dialog box. Cilck **Add Item**. Under **Type**, select **Skill.** Under **Name**, select **All Functions**.

cfgdata

Parameters:

- `folder: STRING` (Section name on the Annex tab of the `Person` object)
- `option: STRING` (Option name in the section on the Annex tab of the `Person` object)
- `default: FLOAT` (Preset value is returned, if the option is not found)

Return value type: `FLOAT`

This function returns a numeric value. If the agent's option is found, it returns `1`. If the option is not found and the default value is not specified, it returns `0`. If the option is not found and the default value is specified, the default value is returned.

exist

Parameters: `Skill Name: STRING` (Skill name)
Return value type: `FLOAT`

This function checks if an agent has the provided skill and is applied directly to skill names. It returns `1`, if an agent has the skill, or `0` otherwise.

folder

Parameters: `template: STRING` (Folder name in the Configuration Layer under Persons)
Return value type: `FLOAT`

This function checks if an agent is configured in a folder with a name that matches the specified template. It returns `1`, if there is a match, or `0`, otherwise.

group

Parameters: `Agent Group: STRING` (Agent Group name)
Return value type: `FLOAT`

This function checks if an agent belongs to the specified group. It returns 1, if yes, or 0, otherwise.

loc

Parameters: `Switch Name: STRING` (Switch name)
Return value type: `FLOAT`

This function verifies the agent location. It returns 1, if an agent has a DN belonging to the specified Switch, or 0, otherwise.

login

Parameters: `media: STRING` (Media name)
Return value type: `FLOAT`

This function returns 1, if an agent is logged in to provided media, or 0, otherwise. If media is set to any and an agent is logged in to at least one media, then 1 is returned.

name

Parameters: `template: STRING` (Agent employee ID template)
Return value type: `FLOAT`

This function checks if an agent name matches the specified template. It returns 1, if there is a match, or 0, otherwise.

sitex

Return value type: `FLOAT`

The `sitex` function verifies an agent's location (site). It returns 1 if the agent was logged in at the provided site and 0 otherwise. If the name of the site is set to `this`, then instead of checking, the function just returns the site name. The agent's site is usually provided by the agent himself through the `site` parameter in the `Reasons` attribute of the agent ready request. **Note:** In SIP cluster configuration, it is automatically populated by SIP Server.

String Manipulation Functions

The following string manipulation functions are available in IRD starting with version 8.1.400.39:

StrUTF8Encode

Parameters: `String: STRING` or variable (representing the string)
Return value type: `STRING`

This function performs encoding of the provided multibyte String into UTF8 format. Encoding is done based on current locale.

StrUTF8Decode

Parameters: `String: STRING` or variable (representing the string)
Return value type: `STRING`

This function performs decoding of the provided UTF8 String into multibyte format. Decoding is done based on current locale.

StrURLEncode

Parameters:

- `String: STRING` or variable (representing the string)
- `UTF8: INTEGER` or variable (representing the integer true/false value)

Return value type: `STRING`

This function performs URL Encoding of the input String. Additionally UTF8 conversion might also be applied before URL encoding if the UTF8 parameter is true (on any integer that is not 0). If UTF8 encoding is specified but fails, then URL encoding will be applied to the original string.

Note: URL encoding is the process of replacing unprintable characters within URLs, such as -, _, ., !, ~, *, ', (, and), with their corresponding hexadecimal code prefixed with %. The URL encoded value is safe to use as a value for the Web URL parameters.

StrURLDecode

Parameters:

- `String: STRING` or variable (representing the string)
- `UTF8: INTEGER` or variable (representing the integer true/false value)

Return value type: `STRING`

This function performs URL decoding from the input string. Additionally UTF8 conversion might also be applied after URL decoding if the UTF8 parameter is true (on any integer that is not 0). If UTF8 decoding is specified but fails, then just the result of the URL decoding process will be returned.

Note: URL decoding is the process of replacing the URL encoded characters with their corresponding original characters and replacing the + sign with a space.

Support for JavaScript

Beginning with release 8.1.400.67, URS supports execution of stand-alone JavaScript strategies and subroutines. The supported standard is **ECMA-327**, 3rd edition.

IRD now provides a possibility to write a part of or the entire strategy logic using the JavaScript

language in Macro or Script objects.

- To enable Script objects, navigate to the **Tools/Routing Design Options** dialog and in the **Views** tab select the **Routing Design/Scripts** checkbox.
- For Script objects, use the **Script/ecma** type.
- For Macro objects, select the **Complex macro** checkbox.
- Write the script source code in the **Definition** tab and always use the **Verify** button to check the validity of the script or macro you create.

The objects you create can be used in regular IRD strategies/subroutines. Script objects can also be used on their own.

To pass data from the calling strategy to the Script object:

Accessing of input parameters requires writing explicit code that will do it. Calling the strategy will put input parameters in a stack. In scripts, use the `GetInputParams()` function to retrieve the input parameters. Retrieved data will be removed from the stack, so calling the function more than once in a Script will result in an error. It is recommended to call the `GetInputParams()` function at the very beginning of the script.

To pass data from the Script object to the calling strategy:

Use the `SetOutputParams(object)` function to return output parameters.

> ### Important
> If the script is used as a subroutine, then the last executed command should be a Return(error_code); where error code is **0** if all OK and a non-zero error code if any error occurs. The following is a sample script with 2 input parameters (**i1** and **i2**) and 2 output parameters (**out1** and **out2**):
>
> ```
> var inp = GetInputParams();
> var res= new Object();
> res.out1= inp.i1+inp.i2;
> res.out2= inp.i1-inp.i2;
> SetOutputParams(res);
> Return(0);
> ```

To pass data to a Macro:

Macros are called as subroutines. All macro parameters are considered as input and passed automatically, there are no output params.

To pass data from a Macro to the calling strategy:

Data can be passed back to the calling strategy from Macro through the following:

- stack – combination of `Push(data)` and `data=Pop()`functions.
- wake up calls – combination of `WakeCallUp('', data)` and `data=WakeUpData['']` functions.

- dedicated returning – combination of `ReturnEx(error, data)` and data=`ReturnData[]`functions.

INTERACTION scope variables can be used anywhere (Scripts, Macros, strategies, subroutines) to pass data in any direction.

URS JavaScript Code/Functions Support:

- Standard **ECMA-327**, ECMAScript 3rd Edition Compact Profile.
- All operators must be ended with ";".
- Internal characters' presentation – multibyte (not UTF16).

Functions for JavaScript Strategies

All URS functions can be used. Functions are listed in the **compiler.dat** file and located in the IRD installation directory. The following functions have mostly been created for use in JavaScript strategies (though they also can be used in IRD strategies, if needed):

GetCallObject()

The `GetCallObject()` function returns an object presenting the current call. The properties of this object can be used to access (read and sometimes write) real call data. This call object has the following properties:

- **udata** or **userdata** – allows read/write access to call attached data
  - `GetCallObject().udata.abc = 123`; the same as `Update('{d}abc', 123)`
  - `X = GetCallObject().udata.abc`; the same as `X = UData ('abc')`
  - `X = GetCallObject().udata['*']`; returns all calls user data
  - `GetCallObject().udata.abc = null`; the same as `DeleteAttachedData ('abc')`
  - `GetCallObject().udata.abc = undefined`; the same as `DeleteAttachedData ('abc')`
  - `delete GetCallObject().udata.abc`; the same as `DeleteAttachedData ('abc')`
  - `GetCallObject().udata.abc+ = 5`; take current value of attached data, increment it by 5 and re-attach.
  - `GetCallObject().udata+ = {"abc":123, "def":"aaa"}`; the same as `Update('', '{d}abc:123|def:aaa')`
  - `GetCallObject().udata- = "abc"`; the same as `DeleteAttachedData ('abc')`
- **xdata** or **extensions** – allows read/write access to call extensions data
- **voice** – allows read access to following call data: `thisdn` or `_dest`, `_orig`, `acdqueue`, `type`, `ani`, `dnis` or `contactedaddr`, `ced`, `media` or `category`, `customerid`, `connid`, `g_uid`, `callid`, `trunk`, `media_server`, and `control_server`
- **location** – allows read access to following call data: `media_server` and `control_server`
- **data** – allows read/write access to the call's `INTERACTION` scope variables

The following functions for target selection are preferred for performing target selection from JavaScript strategies:

PutIntoQueue(id, virtual queue, priority, statistic, selection flag, target, . . .)

This function is similar to the SelectDN function with following differences:

- It allows explicit control of call queues with the first parameter. In the `SelectDN` function, `id` is selected by the compiler and they are all different within a strategy. Using the same `id` in different instances of the `PutIntoQueue` function will result in the call's requeueing (call targeting resulted from an older `PutIntoQueue` function will be eliminated and replaced with the new one). If `id` is set to **0** then the new `id`-less call's queue will always be created.
- It only puts the call into queue but does not try to select a target. The function returns the `refid` that can later be used for target selection.

RemoveFromQueue(id, virtual queue)

This function is similar to the ClearTargets function, but allows more explicit control over excluding a call from one or another queue.

- If the `virtual queue` is empty and `id` is not **0**, then the call will be removed only from the call queue in which it was placed by the `PutIntoQueue` function using the same `id`.
- If the `virtual queue` is empty and the `id` is **0**, then the call will be removed from all call queues as well as from all virtual queues (Virtual queue EventDiverted(redirected) will be distributed for all virtual queues call is in).
- If virtual queue is not empty the n call will be removed from the all call's queues associated with provided virtual queue as well as from this virtual queue (the virtual queue `EventDiverted`(redirected) event will be distributed for this virtual queue).

RemoveFromQueueN(refid)

This function is similar to the RemoveFromQueue function, but it removes the call from call queue identified by `refid` returned by the `PutIntoQueue` function. The call is not removed from any virtual queue.

SelectTargetFromQueue(refid, timeout)

This function is similar to the `SuspendForDN` function (and in case of *refid=0*, they are identical). The function places the call into a *waiting for targets from all queues calls are in* state for the duration of the provided timeout. Before doing that function explicitly tries to select target (by statistics) from the queue the call is in, pointed by the provided `refid` (return value of the `PutIntoQueue` function). If `refid` is **0**, then all call queues will be tried. The returned value is identical to the value provided by the SuspendForDN function.

RouteToTarget(target)

This function is a more advanced version of the `RouteCall` function. Similar to the `RouteCall` function it accepts a target returned by the `SuspendForDN` function or the `SelectTargetFromQueue` function and routes the call to the provided target. The returned value is identical to the value returned by the `RouteCall` function.

The main difference is that this function will try all means to route the call; that is, answer the call if

needed, return it back to the original Routing Point, and if the call is on an uninterruptible treatment or in a transition state, it will wait until routing is possible again. It is this function, and not the `RouteCall` function that IRD uses when custom routing is activated in Routing objects.

JavaScript support for Treatment functions

The function, `TreatmentPlayAnnouncement`, and other `Treatment<TreatmentType>` functions which are used to start and wait for treatment ending can not be used in JavaScript directly due to a name conflict with the same name constant.

The recommended way to use this function is by referring to it indirectly through a URS functional module. For example:

```
var funcTreatmentPlayAnouncement= FunctionalModule('[www.genesyslab.com/modules/
urs')['TreatmentPlayAnnouncement'];
funcTreatmentPlayAnouncement('LANGUAGE', 'English(US)', 'PROMPT.1.TEXT', 'http://127.1.1.1/
audios/bienvenida_001.wav');
```

Alternatively the function, `StartTreatmentPlayAnnoncement` (followed with `SuspendForTreatmentEnd`) can be used instead in this case. Effectively, the `Treatment<TreatmentType>` functions can be considered as shortcuts for combinations of two functions `StartTreatment<TreatmentType>` and `SuspendForTreatmentEnd`.

Notice how parameters in the above sample are passed to `Treatment<TreatmentType>` or `StartTreatment<TreatmentType>` functions, it is different from passing them to busy treatment functions. The parameters must be passed directly and as a sequence of keys and values: key, value, key, value, and so on.

Sample JavaScript Snippet

The following is a sample JS snippet where the call is routed to any available agent during working hours and if outside working hours, the not the strategy plays an appropriate treatment. The treatments to play are obtained from some a Web Service.

```
var funcTreatmentPlayApplication= FunctionalModule('[www.genesyslab.com/modules/
urs')['TreatmentPlayApplication'];

function RouteAnyAgent(greeting, busy)
{
   funcTreatmentPlayApplication('APP_ID',greeting);
   var queue= PutIntoQueue(1, '', 0, "StatAgentLoading", SelectMin, '?:2>1.GA');
   AddBusyTreatment(TreatmentPlayApplication, 'APP_ID:' + busy, 0);
   var target= SelectTargetFromQueue(queue, 60);
   if (!Failed())
     RouteToTarget(target);
}

function RoutetoAnyAgentWorkingHours(greeting, weekend, outoftime, busy)
{
    if (Date.getDay()==0 || Date.getDay()==6)
       funcTreatmentPlayApplication('APP_ID',weekend);
    else if (Date.getHours()<8 || Date.getHours()>18)
       funcTreatmentPlayApplication('APP_ID',outoftime);
    else
       RouteAnyAgent(greeting, busy);
}
```

```
x= ObjectFromJSON(
       GetHttpRequestInfo(1,
            StrFormat("http://myhost:myport/MyMessage?ani=~s&dnis=~s&connid=~s",
            ANI(), DNIS(), ConnID()), "", "", 0, "", ""));

if (!Failed())
      RoutetoAnyAgentWorkingHours(x.greeting, x.weekend , x.outoftime , x.busy);

Default();
```

Secure Connections Support Includes SNI Functionality

Starting with release 8.1.400.71, URS supports Server Name Indication (SNI) extension for TLS handshakes. As a result, HTTPS resources can be contacted from within URS, that is, using the Web Service block in IRD.

If the SNI functionality is activated, URS adds an extra parameter, `tls-target-name`, into the transport parameters of the connecting requests, set to the name of the host the web request is directed to.

The SNI functionality can be activated using one of the following methods:

- Set the `def_sni` option to `true`. For a description of the new option, refer to the New or Updated Option Descriptions topic.
- Specify `sni:true` in the **Hints** field of the IRD Web Service object. The **Hints** field is located under **TLS group** in the **Security** tab of the Web Server object.

> ### Important
> URS first uses the value from the IRD Web Service object. If that is absent, it uses the value set by the `def_sni` URS option.

Optimal Skill Update Mode

Starting with release 8.1.400.75, URS is enhanced to provide a possibility for more optimally updating skill expressions when part of the skill expression uses the `login` function. The optimal skill update mode is activated when the new URS option, `content_update_on_login`, is set to `false`. Setting it to `force`, which is the default value, overrides the enhancement and retains the behavior from the previous URS versions.

`content_update_on_login`
Location: `default` section of URS `Application` object
Default Value: `force`
Valid Values: `force`, `false`
Changes Take Effect: Immediately

Timeout to Wait before Sending Negative Response to Web Client

Beginning with version 8.1.400.78, URS allows users to specify a timeout to wait for before sending a negative response to a web client's request to perform an operation for an interaction that URS does not have. Timeout (in seconds) can be defined in the additional parameter, **maxdelay**, in the web request to URS or in the **call_sync_time** option.

call_sync_time
Location: `http` section of URS `Application` object
Default Value: `0`
Valid Values: `0` to `10`
Changes Take Effect: After restart

The value defined in the **call_sync_time** option is used if the **maxdelay** parameter is not specified.

For example, `urs/call/ConnID/func?name=Timeout¶ms=[120]&maxdelay=2`. Here, URS is requested to apply the `Timeout[120]` function to the interaction. If the requested interaction is not found, then URS will wait up to 2 seconds. If URS gets the requested interaction during the 2 seconds, it will apply the function to it. Else, it answers with an error.

> **Important**
>
> The maximum value allowed for the `maxdelay` parameter (and the `http/call_sync_time` option) is **10** seconds.

Multithreading Capability

Beginning with release 8.1.400.78, URS is enhanced with a multithreading capability to find matching agents who satisfy the conditions of the specified skill expression, in a given configuration . This improves URS performance in larger environments characterized by agent headcounts exceeding 10,000 or even 100,000 across locations.

A new configuration option, `mts`, is introduced to control the multithreading capability.

mts
Location: `default` section of URS `Application` object
Default Value: `0:0`
Valid Values: `0:0`, `1:0`, `1:1`
Changes Take Effect: Immediately

`0:0` - indicates multithreading is switched off.
`1:0` - indicates multithreading is turned on, but will be applied to expressions containing only skills. Skill expressions with statistics and functions are excluded in single threading mode.

1:1 - multithreading is turned on and skill expressions with statistics and functions are also included for multithreaded processing.

A new console command, `mtskill`, is also provided for exploring the multithreading capability.

Format: `mtskill` <TenantName> <SkillExpression>.

As an output, URS provides 2 corresponding time intervals (in microseconds).

The following **limitations** are to be considered before turning on multithreading:

- Multithreading is justifiable only in very big environments.
- URS must run on very powerful hardware with multiple processors available for URS (running multithreading on single processor machine will slowdown URS).
- URS logging is disabled in multithreaded mode, while URS is updating skill expressions.
- Some skill expression functions, such as `run`, `group`, `folder`, and `tag`, have too big a footprint to be safely used in a multithreaded environment. Skill expressions containing these functions will always be executed in the single threaded mode irrespective of the value of the `mts` option.

Improved EWT Accuracy

EWT accuracy is determined by the difference between the EWT value given when a call enters a queue and the actual wait time that the particular call spent in the queue. When the difference is minimal it translates to a better EWT accuracy. EWT accuracy is impacted if calls of different types are placed into the same VQ. For example, if one VQ is used for both inbound, virtual callback (CB), and CB outbound calls, then EWT accuracy is low because outbound CB calls have some unique properties, such as:

- outbound CB calls are placed in VQ only for a short time of several seconds.
- outbound and virtual CB calls are two call representing the same interaction in a VQ. This creates double counting.
- outbound and virtual CB calls leave the queue at the same time breaking the EWT calculation model, which assumes that one call is distributed at a time and calls are distributed at a constant rate.

EWT accuracy can be increased if outbound CB calls are not taken into account for EWT calculation.

Beginning with release 8.1.400.83, URS provides more accurate EWT calculations for a Virtual Queue when interactions that are not intended to be routed by URS end up in the Virtual Queue. URS now provides the ability to mark and ignore such calls in EWT calculations. You can mark an interaction by setting its run time mode to 524288 and exclude them from EWT calculations.

To mark an interaction you can, use the `SetRunTimeMode` function within the IRD strategy. Or, it is set automatically if `EventRouteRequest` starting a strategy contains `AttributeCallType` with value **3** (outbound) and `AttributeUserData` has the `_CB_N_CALLBACK_ACCEPTED` key set to **1**.

Ignoring such marked calls in EWT calculations is controlled with the new configuration option, `lvq_ignore_duplicates`.

`lvq_ignore_duplicates`
Location: `default` section of URS `Application` object; can also be defined at the VQ level
Default Value: `false`
Valid Values: `true`, `false`
Changes Take Effect: Immediately

Setting the option to `true` ignores marked calls in EWT calculations.

If the option is set to `true` for a VQ and when an interaction marked as 524288 leaves the VQ:

- it will not change the distributing quitting rate from that VQ (its quitting will not go into an array of the last 32 quits). This improves the accuracy of `lvq` requests when `aqt=urs`.
- it will be skipped (not counted) when URS goes through (counts) all calls in the VQ to answer an `lvq` web request. This improves the accuracy of all types of `lvq` requests (`aqt=urs`, `urs2` or `stat`).

If the option is set to `false` for a VQ, then all interactions in that VQ (regulars or duplicates) will be counted for EWT calculations.

Log messages about sending VQ events (such as, *Queued*, *Diverted*) are now extended to indicate if URS counts or ignores the interaction in EWT calculations.

```
12:38:36.134_T_I_00820324eb06cefd \[14:02\] sending event 58 for vq
RBWM_UKFD_Ingress_VQ_EMEA (0 53-7 (0, i=349923 o=349904) 1633001911 170/32) (x)
```
(The x in the above sample message indicates if the interaction is considered by URS as a duplicate. If x is 1, it indicates that the interaction is a duplicate, and if x is 0, it indicates that the interaction is not a duplicate.)

Improved EWT Consistency

Beginning with version 8.1.400.84, URS is enhanced to improve consistency in EWT calculations for web requests that are interactionless and utilize average handling time provided by the default `lvq` URS method.

Improved consistency in EWT calculations is characterised by:

- absence of sharp changes (peaks or drops) in EWT values provided by URS.
- similarity in values of EWT returned by different URS instances in the same environment.

Default behavior

When obtaining EWT values for virtual queues (in response to the `lvq` web request) URS utilizes its own data, that is, calls that it places into virtual queues, targets that these calls are waiting for, and so on. As a result:

- it is possible that there is a sharp change in the provided EWT values if URS switches between sets of data used to calculate EWT.

- values returned by different URS instances might not be the same (if the data used by the different URS instances are also different).

To simulate a global queue perception in a distributed call center, multiple URS instances in the same environment must be able to provide consistent (ideally, the same) EWT value for the same VQ. A caller must get the same EWT value regardless of which data center the call lands in.

URS has the following two data sources for calculating the average handling time (time per call from the VQ) for interactions that are not defined in a web request (`max` is used in place of `connid`):

1. Internal queues created by URS when executing strategies.
2. One of the skill expressions from the internal queues that URS remembers (referred as the VQ Presenter). Internal queues are short lived objects that URS might dispose if not used. It is possible that all internal queues associated with a VQ are deleted. To be able to provide reasonable data even in such cases, URS remembers one of the skill expressions. URS selects the skill expression to remember based on which expression returns the highest number of agents. The identified skill expression represents the VQ when there are no internal queues and is referred to as the **VQ Presenter**.

Usually, URS tries to get data from the internal queues first and if at least one internal queue associated with the VQ exists, then, as listed above, data source 1 is used. If there are no internal queues, data source 2, that is, data provided by the VQ Presenter is used. A side effect to this approach is that URS can spontaneously switch between the two data sources (for interactionless web requests). This might result in the returned EWT values fluctuating frequently.

New enhanced behavior

In the new enhanced behaviour, URS uses the VQ Presenters as the primary source of data for calculating EWT even if internal queues exist.

To facilitate the new behaviour, URS ensures that:

- A VQ Presenter always exists.
  - Any skill expression that the strategy uses can be used as a VQ presenter, even if the skill expression contains statistics.
  - A skill expression identified as a VQ Presenter will not be deleted even if it has not been used for a long time.
- A VQ Presenter can be set or changed only in the following cases:
  - On VQ creation. If a VQ associated with an Agent Group as its origination DN, then this Agent Group will be set as the initial presenter for the VQ.
  - On creating a new internal queue associated with the VQ. At each such instance, sizes of the current VQ presenter and skill expression used by the internal queue are compared and the biggest one is selected as the new VQ presenter.
  - On placing any interaction into the VQ. Re-skilling of agents theoretically might result in reducing the size of the selected presenter and it will no longer be the biggest presenter. As a result, URS constantly rechecks the size of the current VQ presenters whenever a call is added into the VQ. The size of the current presenter and skill expression used by the call to enter the VQ are compared and the biggest one is selected.

Each time a VQ Presenter is changed, an entry is logged. For example, `15:27:23.757_M_I_ [10:85]`

`LVQ NameOfVQ presenter set: <?Agents5_10:>(21499ec7bb0) media=0.`

If different URS instances are executing the same set of strategies, it is likely that all those URS instances will also have the same VQ Presenters.

By deafult, the existing `lvq` method with `aqt=urs2` will continue to work as before. A new configuration option, `lvq_force_presenter`, is introduced to activate the new behaviour.

`lvq_force_presenter`
Location: `default` section of URS `Application` object; can also be defined at the VQ level with section name as `URS application name` or `__ROUTER__`
Default Value: `false`
Valid Values: `true`, `false`
Changes Take Effect: Immediately

Setting the option to `true` activates the new behaviour where a VQ presenter is used as a primary source of information to obtain the average handling time per call.

Reporting

To allow evaluation of the quality of the EWT calculations, URS can be enabled to collect (and report on) data about the estimated and actual waiting times for calls in a virtual queue.

- Every time an interaction enters a virtual queue the current EWTs are obtained and stored inside the interaction.
- Every time an interaction leaves the virtual queue the stored EWTs along with the actual time the interaction was waiting for is stored in the virtual queue. The virtual queue store information only about the latest interaction that quit the queue.

The `lvq` web request is extended to include this information as well as other statistical data that can be useful for tracing processing of calls in one or another virtual queue.

You can use the following requests to query data:

- `urs/call/max/lvq?tenant=TenantName&name=VirtualQueueName&filter=presenter`

- returns the skill expression/agent group used as the current presenter for the specified virtual queue.

- `urs/call/max/lvq?tenant=TenantName&name=VirtualQueueName&filter=trace&ewttrace[=N]`

- `filter=trace` returns tracing data for the specified VQ.
- `ewttrace` or `ewttrace=N` triggers the tracing mode for the specified VQ for the next `N` minutes (by default `N=3`).

For a virtual queue in tracing mode, URS collects extra data about the virtual queue (as permanent collection of such data takes a toll on performance). Additionally, for a VQ in tracing mode, URS records extra information in the log entries even if the `default/verbose` option is set to `false`.

When `filter=trace`, the following data is returned (note that when a value is unknown the field might not be returned):

Field	Description
time	current UTC time
lcalls_in	Local number of calls that have entered the VQ so far.
lcalls_out	Local number of calls that have exited the VQ so far.
lcalls	Local number of calls in the VQ (lcalls_in - lcalls_out).
rlcalls	Local number of real calls in the VQ (lcalls – duplicates).
calls	Global number of calls in the VQ (effectively StatCallsInQueue as returned by StatServer).
mrs	Multi-URS factor (used to convert local data into global (calls/lcalls).
rcalls	An estimate of the global number of real calls (rlcalls * mrs).
aqt_stat	Time per call according to StatServer (=StatExpectedWaitingTime/StatCallsInQueue).
ewt_stat	Waiting time according to StatServer (=aqt_stat * (rcalls+1)).
aqt_urs	Time per call according to URS or global quitting rate (local quiting rate / mrs).
ewt_urs	Waiting time according to URS (=aqt_urs * (rcalls+1)).
aqt_urs2	Time per call according to URS average handling time (calculated as per URS settings).
ewt_urs2	Waiting time according to URS (=aqt_urs2 * (rcalls+1)).
aqt_ursp	Same as aqt_urs2, but aht is calculated based on presenter.
ewt_ursp	Same as aqt_urs2, but aht is caluclated based on presenter.
xid	connid of latest call distributed into the VQ.
xtm	Latest call entry time into the VQ.
xewt_stat	StatServer based estimate of waiting time for the latest call (at the xtm time).
xewt_urs	URS quit rate based estimate of waiting time for latest call (at the xtm time).
xewt_urs2	URS average handling time based estimate of waiting time for latest call (at the xtm time).
xewt_ursp	URS average handling time for presenter based estimate of waiting time for the latest call (at the xtm time).

For a VQ in tracing mode the log message (logged when the call is distributed from the VQ) is as follows:

`12:36:04.200_M_I_03390320b4c930b0 [14:02] LVQ NameOfLVQ (58,1) ewts: xtm, xwt, xewt_stat, xewt_urs, xewt_urs2, xwt_ursp,` (along with some other data).

Note that if the VQ is not traced the above message might still be logged if the log level is set to 4 or 5, but the message will have no data for `xewt_urs2` and `xwt_ursp`.

You can follow one of the two patterns given below for tracing EWT for a VQ with the provided web requests:

1. Periodically (for example, once per minute) you can send URS the following web request, `urs/call/max/lvq?tenant=TenantName&name=VirtualQueueName&filter=trace&ewttrace`. Collect the output data for a period of time (say, a few hours) and visualize the output (for example, as an Excel spreadsheet).
2. Send the following web request to URS, `urs/call/max/lvq?tenant=TenantName&name=VirtualQueueName&filter=trace&ewttrace=180`. Collect URS logs for the next 180 minutes (3 hours), extract the related log messages from the logs and visualize them.

Limitations

The new behaviour is not necessarily better if compared with the default behaviour where URS relies on internal queues. That depends on how a specific solution has been implemented and how virtual queues are used in the solution. It is expected that the new behaviour will work good in cases of cascaded routing.

Also, URS cannot detect by itself, when the usage of one or another virtual queue changes sharply. For instance, solutions/strategies may start to use completely new skill expressions. When the usage of a virtual queue is changed, URS might still continue to use old virtual queues' presenters (if they happen to be bigger). To address such cases and avoid restarting URS to align virtual queue usage, the `lvqs` console command can be used with an extra optional parameter, `reset_presenter`.

For example: `lvqs TenantName VQName reset_presenter`
- where VQName is name of the virtual queue or *.
- All matched virtual queues will have their presenter updated (their presenters will be reset based on the current internal queues URS has for them).

From the Web API, the `lvqs` console command can be executed as, `urs/console?lvqs TenantName VQName reset_presenter`.

URS Functions and Configuration Server

URS functions that read/write data from Configuration Server (FIndConfigObject, SetObjectProperty, GetObjectProperty, and so on) have to manipulate with Configuration Server object Types and Subtypes in the way that Configuration Server understands them. Every Configuration object has a Type and some of them (Transactions, for example) might have a Subtype. For Configuration Server, both Types and Subtypes are just plain numbers (Enumerators) and are identified by numbers. For example, 9 means CFGApplication, 2 means DN, 16 means Transaction, 21 means Transaction subtype list, and so on.

Which number represents which Type or Subtype is basically Configuration Server-related information (provided in Configuration Server-related documents). Reference to those Configuration Server-related data can be accessed as follows:

- Types of Configuration Objects
- Types of Transactions
- Other Configuration Server Types

For some Configuration Server Enumerators, URS strategies allow dedicated names. For example, CFGApplication means just number 9. Where names are allowed, you can use either names or numbers themselves; where names are not allowed, numbers must be used to present object Types and Subtypes.

When configuring URS strategies, names are not provided for every Type/Subtype as Configuration Server might introduce new Types/Subtypes and URS cannot follow all of them. So names are used to present only well-established object Types with URS having special processing of every Type. For other Types (like Transaction Subtypes), naked numbers need to be used – URS does not interpret them; it just passes them to Configuration Server and any number supported by this Configuration Server can be used.

Regarding Transactions, they are uniquely identified either by dbid or by pair transaction type + transaction name. For example, to find Transactions, the FindConfigObject function must be supplied either with a dbid or with pair type and name. Transaction Types are numbers and Transaction of Type List is 21.

```
FindConfigObject[CFGTransaction, ‘type:21|name:mylist’]
```

Or the same (as IRD will replace word CFGTransaction to number 21)

```
FindConfigObject[16, ‘type:21|name:mylist’]
```

SetObjectProperty

The function SetObjectProperty enables you to set more then one property within a single Section. Every odd parameter after the parameter Section is interpreted as the property name and every even parameter as the property value. If the last property name has no matched parameter with value, it

results in the deletion of this property from object properties.

As in this example, the standard syntax is:

```
SetObjectProperty[CFGTransaction, 21, '<Transaction List Object Name>', '<Section>',
'<Key_1>', '<Value_1>', '<Key_2>', '<Value_2>', '<Key_3>', '<Value_3>']
```

New or Updated Option Descriptions

The option descriptions in this topic are new or replace descriptions in the *Universal Routing 8.1 Reference Manual*.

> **Important**
>
> URS options are placed in the following option folders:
>
> - For the URS Application — in the `default` section of the **Options** tab or the **Annex** tab of the URS **Properties** dialog box. If options are specified in both places, those specified in the **Options** tab take precedence.
> - For a T-Server Application to which URS connects — in the **Options** tab or the **Annex** tab of the **T-Server Application Properties** dialog box, in a section with the same name as the name of the **URS Application**, or in a section named either `__ROUTER__` or `default`.
> - For a Stat Server Application to which URS connects — in the **Options** tab of the Stat Server Application **Properties** dialog box, in a section with the same name as the name of the URS Application, or in a section named `__ROUTER__` or `default`.
> - For a Message Server Application to which URS connects — in the **Options** tab of the Message Server Application **Properties** dialog box, in a section with the same name as the name of the URS Application, or in a section named `__ROUTER__` or `default`.
> - For all other object types—in the **Annex** tab of the object **Properties** dialog box, in a section with the same name as the name of the URS Application, or in a section named `__ROUTER__`.

addp_timeout

ADDP is used to detect a loss of connection between the HTTPBridge and URS. The `addp_timeout` option is used to specify how often the HTTPBridge will send ADDP requests to URS and wait for responses. If no response is received, HTTPBridge will terminate within the defined timeout.

Location: `web` section of the URS `Application` object
Default Value: `30` (seconds)
Valid Values: Any positive integer
Changes Take Effect: After restart

If set to `0` or any invalid value, the ADDP functionality is disabled.

automatic_ideal_agent

You can use this option as an alternative to using `SetIdealAgent`. If set to `true`, when URS places the interaction into a queue for first time, and:

- if this queue targets/agents are defined as a skill expression and,
- if function SetIdealAgent was not yet called for this call, then,

URS will automatically call the `SetIdealAgent` function with the value of the skill expression used for this queue as a target. For more information, see Using Agent Skills for Agents/Calls Prioritization.

Location in Configuration Layer by precedence: Routing Point, T-Server, Tenant, URS
Default Value: `false`
Valid Values: `true`, `false`
Changes Take Effect: Immediately

default_stat_server

The default_stat_server option is extended to allow you to specify a separate default Stat Server for every Virtual Queue. The extension covers cases where URS's Connections list contains multiple Stat Servers. An updated option description is presented below.

Location in Configuration Layer by precedence: Virtual Queue, Routing Point, T-Server, Tenant, Universal Routing Server Application
Default Value: None. If a default is not specified, URS uses the first available Stat Server on its Connections list.
Valid Value: The name of any available Stat Server
Changes Take Effect: Immediately

This option designates which Stat Server to use as the default location when a target in a strategy omits the location (that is, the target has a format of `ID` or `ID.type` rather than `ID@StatServerName.type`).

def_http_proxy_host

Location in Configuration Layer by precedence: `web` section of the URS `Application` object
Default Value: An empty string
Valid Values: Any valid host name
Changes Take Effect: After restart

URS provides support of HTTP Proxies for an "http://" type of request. HTTP Proxies are specified globally at the URS Application level, in the web section, by the `def_http_proxy_host` and `def_http_proxy_port` configuration options. The `def_http_proxy_host` option specifies the HTTP Proxy host.

def_http_proxy_port

Location in Configuration Layer by precedence: `web` section of the URS `Application` object
Default Value: An empty string

Valid Values: TCP port
Changes Take Effect: After restart

This option specifies the HTTP Proxy port for an "http://" type of connection.

def_sec_protocol

Location in Configuration Layer by precedence: web section of the URS Application object
Default Value: As defined by Genesys Security Layer
Valid Values: SSLv23, SSLv3, TLSv1, TLSv11, TLSv12, TLSv13
Changes Take Effect: After restart

This option specifies which handshake protocol HTTP Bridge uses for outgoing HTTPS connections. This option can be used only on UNIX operating systems with Genesys Security Pack on UNIX 8.1.x, starting with 8.1.300.05. This option has no effect on Windows. Protocols are specified by the option values as follows:

- SSLv23—SSL v2.0
- SSLv3—SSL v3.0
- TLSv1—TLS v1.0
- TLSv11—TLSv1.1
- TLSv12—TLS v1.2
- TLSv13—TLS v1.3

Important

- Starting with 8.1.400.96, the def_sec_protocol configuration option supports a new value, TLSv13.
- Starting with 8.1.400.33, the def_sec_protocol configuration option supports a new value, TLSv12. This option was originally introduced in URS 8.1.400.13 on 2/13/15.

def_sni

Location in Configuration Layer by precedence: web section of URS Application object
Default Value: false
Valid Values: true, false
Changes Take Effect: After restart

This new option, introduced in URS 8.1.400.71, enables the Server Name Indication (SNI) extension for TLS handshakes. When this option is set to true, URS adds an extra parameter, tls-target-name, into the transport parameters of the connecting requests, set to the name of the host the web request is directed to.

lds

For the `lds` option, you can also specify `map` as a value, in addition to the valid values of `ar` (access resources), `ciq` (calls in queue), and `blk` (agents blocking), as described on page number 645 in the *Universal Routing 8.1 Reference Manual*.

Use `map` if the Message Server communicates between multiple URS instances the information about agents' tags and global maps that the URS instances might use. If a URS instance tags an agent or adds a key-value pair into a map, then, the information about it will be propagated to the other URS instances in the same self-awareness cluster.

http_log_size

Location in Configuration Layer by precedence: `web` section of the URS `Application` object
Default Value: `false`
Valid Value: Size of log file in kilobytes
Changes Take Effect: After restart

This option specifies the HTTP Bridge maximum log file segment size in kilobytes. Once the specified file size is reached, a new segment/file is created and the new log output goes to this new file. You must configure the `http_log_file` option to use this option. Also, see the option `log_remove_old_files`.

log_file

Location in Configuration Layer by precedence: http section of the URS `Application` object
Default Value: no default value
Valid Value: log file name
Changes Take Effect: After restart

This option specifies the name of the log file for HTTP Interface error and trace messages.

log_remove_old_files

Location in Configuration Layer by precedence: `web` section of the URS `Application` object
Default value: `false`
Valid Values: either `false` (meaning old log files are not deleted and all log files will be kept), or an integer specifying the number of log files that will be kept.
Changes Take Effect: After restart

This option specifies whether the previous segments/ files are to be deleted when the new segment/ file is created. You must also configure the `http_log_file` option to use this option.

log_remove_old_files

Location in Configuration Layer by precedence: `http` section of the URS `Application` object
Default value: `false`
Valid Values: either `false` (meaning old log files are not deleted and all log files will be kept), or an integer specifying the number of log files that will be kept.

Changes Take Effect: After restart

This option specifies whether the previous segments/ files are to be deleted when the new segment/ file is created. You must also configure the `log_file` option to use this option.

log_size

Location in Configuration Layer by precedence: `http` section of the URS `Application` object
Default Value: `false` (unlimited size of file)
Valid Value: size of log file in kilobytes
Changes Take Effect: After restart

This option specifies the HTTP Interface maximum log file segment size in kilobytes. Once the specified file size is reached, a new segment/file is created and the new log output goes to this new file.

verbose

This option determines the level of log output.

Location in Configuration Layer by precedence: `http` section of the URS `Application` object.
Default Value: `0`.
Valid Values: `0` to `3`.
Changes Take Effect: After restart.

Level `0` produces no log messages. Levels from `1` to `3` produce log information with corresponding higher levels of detail.

lvqwaittime_stat

From version 8.1.400.53, a new option, `lvqwaittime_stat`, is introduced to control which waiting time statistic URS will use to calculate values in response to `lvq` web requests.

Location: Configuration layer by precedence - `Virtual Queue`, `URS`
Default Value: `StatExpectedWaitingTime`
Valid Value: Any valid statistic name
Changes Take Effect: After restart

When specifying a value for this option, ensure that the statistic is properly configured in IRD or StatServer. If the name is invalid, an error is reported.

lvq_quit_rate_history

Location in Configuration Layer by precedence: `default` section of URS `Application` object; can also be defined at the VQ level with section name as `URS application name` or `_ROUTER_`
Default Value: `32`
Valid Values: Any value from `1` to `64`
Changes Take Effect: After restart

One of the methods URS uses to calculate EWT for a Virtual Queue relies on the details of the last 32 calls (that is, the most recent) distributed from the Virtual Queue (VQ). This option introduced in 8.1.400.88, allows users to control how many distributed calls from the VQ to consider for EWT calculations. When URS creates a VQ it uses the value of this option to determine the size of the distributed calls history for the VQ. Once the VQ is created the size of the distributed calls history for the VQ is determined and can be changed only after a URS restart. If URS creates the VQ after the option was changed at the VQ level, a restart is not needed - this VQ will use the option's latest value.

max_loading

Location in Configuration Layer by precedence: DB Server/Custom Server/StatServer, Database Access Point
Default Value: `0` (zero)
Valid Value: any positive integer
Changes Take Effect: Immediately

Specifies the maximum number of unanswered requests that URS can send to a server; for example, the maximum number of unanswered opening statistic requests that URS can send to StatServer. Specifically prevents database access bottlenecks when there is high call volume and high customer request abandonment or when there is high call volume and low DBMS performance. The latter condition can be caused by an untuned or unoptimized database.

The default value of `0` (zero) indicates that URS is not tracking or limiting the number of requests to the server.

> ### Important
>
> Though the max_loading option is retired and no longer used, URS continues to provide the possibility to control its loading on web servers with the max_loading option, similar to how it is done for regular Genesys servers (for example, DB Server). However, the following differences exist:
>
> - for web servers, the option must be set within the URS application itself, in the `web` section (instead of setting it on dedicated servers).
> - all options in `web` section (including `max_loading`) take effect on http bridge restart.

utf8ors

Location in Configuration Layer by precedence: `default` section of URS `Application` object
Default Value: false
Valid Values: true, false
Changes Take Effect: After restart

Introduced in version 8.1.400.95, this option is used to specify if URS must convert the content of responses on ORS requests from the default locale URS works with, into the utf8 format. Previously such conversion was considered as part of HTTP communications and was performed only if clients (including ORS) communicate with URS through the HTTP connection. Conversion was not performed

if clients connect to URS directly through the Genesys connection layer.

The `utf8ors` option extends this functionality for cases when ORS communicates with URS through a direct ORS to URS connection. When converting data to the utf8 format, URS assumes that data is encoded according to the current locale. Conversion will fail if the format of data URS manipulates does not match the current locale.

pickup_calls

Location in Configuration Layer by precedence: Annex properties of `DN` controlled by URS, T-Server, URS
Default Value: false
Valid Values: false, true, reverse
Changes Take Effect: Immediately

Starting with 8.1.400.36, the `pickup_calls` configuration option is now supported at the T-Server and URS Application levels, in addition to the Routing Point DN level. This option enables smart registration for routing points and depends on T-Server's ability to provide information on all interactions pending on a routing point even before routing points are registered by URS at startup. For additional information on this option, refer to the *Universal Routing 8.1 Reference Manual*.

pickup_strategy

Location in Configuration Layer by precedence: RP, T-Server, or URS Application levels
Default Value: Option is absent by default (URS will execute the same strategy that it uses for regular calls at the RP, for picked up calls also.)
Valid Value: Any valid strategy name
Changes Take Effect: Immediately

Starting with release 8.1.400.92, this option is introduced to address the use case where the strategy for picked up calls at a Routing Point (RP) must be different from the regular strategy loaded on the same RP. This option can be set up at the RP, TServer, or URS levels and provides the name of the strategy to be picked up.

Picked up calls are those that existed on some RP before URS was started and can be processed by URS if the `pickup_calls` option is set to true. By default, URS executes the same strategy for the picked up calls that it executes for calls that started with a regular EventRouteRequest event. The `pickup_strategy` option allows a different strategy to be executed for picked up calls.

There are a few predefined and hardcoded strategies in URS, which have been created for special deployments only and must be avoided in regular environments. Specifically, the strategy named `restart` must be used only in Azure-based deployments. That is, the pickup_strategy option can be used by itself in any environment (if pointing to any appropriate strategy), but setting it to point to the `restart` strategy must be done only in Azure environments.

proxy_use_connect

Location in Configuration Layer by precedence: web section of URS
Default Value: `true`
Valid Values: `true, false`
Changes Take Effect: After restart

This option specifies the connection method to a secure web server through a HTTP proxy server.

- A value of *true* uses the HTTP CONNECT method. URS communicates with web servers through HTTP proxy and performs TLS negotiations directly with the web server.
- A value of *false* uses the legacy method (not recommended). URS communicates with web servers through HTTP proxy and performs TLS negotiations with the proxy server.

report_targets

Beginning with version 8.1.400.63, a new value, `waited`, has been added to the `report_targets` option. If the `report_targets` option is set to *true* or *waited*, URS attaches the `RTargetsWaited` key into `AttributeUserData` of the T-Server's events. The value of the new key is a comma separated list of targets the interaction is waiting for. This data can be used by the default routing strategy if the processing of an interaction fails.

Location in Configuration Layer by precedence: URS
Default Value: `true`
Valid Values: `true`, `waited`, `false`
Changes Take Effect: Immediately

For a complete description of the option, refer to page number 660 in *Universal Routing 8.1 Reference Manual*.

self_port

Location in Configuration Layer by precedence: `default` section of URS
Default Value: `default`
Valid Values: none or any valid TCP/IP Port ID
Changes Take Effect: After restart

Release 8.1.400.32 introduces a new option, `self_port`, which enables URS to establish a SELF connection when the default listening port is secured, since a SELF connection is not supported via a secured port.

- A value of none instructs URS to not connect to itself so any related functionality, such as the RequestRouter function, will be unavailable.
- A value of `default` instructs URS to use a default port to connect to itself.
- Use the value, `hip`, for an http interface connection to URS when the default port is secured.
- If you not specify option `self_port`, then URS will use value `default` for this option.

skill_in_group_sync

Location in Configuration Layer by precedence: URS
Default Value: 10
Valid Values: Any non-zero positive integer
Changes Take Effect: After restart

This option specifies the maximum number of attempts for URS to execute `GetSkillInGroupEx` and `CountSkillInGroupEx` functions when their parameter sync is set to `true`.

start_primary

Location in Configuration Layer by precedence: URS `Application` object
Default Value: `true`
Valid Values: `true`, `false`
Changes Take Effect: After restart

This option specifies whether URS starts in primary mode or in backup mode in cases when a primary-backup URS pair is configured. If this option is set to `true`, then URS will start in primary mode. If this option is set to `false`, URS starts in backup mode. After the initial URS startup, Management Layer may switch the running mode of the URS `Application` depending on the startup order.

unknown_aht

Location in Configuration Layer by precedence: URS `Application` object
Default Value: `9.999` (seconds)
Valid Values: Any positive number (seconds)
Changes Take Effect: After restart

This option provides the average handling time for virtual queues that URS will use in cases when there is not enough information to calculate the actual EWT.

virtual_queue_attach

Location in Configuration Layer by precedence: URS `Application` object
Default Value: `true`
Valid Values: `true`, `false`
Changes Take Effect: Immediately

When set to true, URS propagates `AttachedDataChanged` events through Virtual Queue DNs, and does not when set to `false`. The purpose of this option is to reduce network traffic by limiting the amount of changed events in the attached data of calls. If deployed in a SIP Cluster environment (URS option `environment` contains value `tcluster`, then the default value of the option becomes `false`.

wait_agent_activity

Location in Configuration Layer by precedence: `default` section of URS `Application` object
Default Value: `true`
Valid Values: `true`, `false`
Changes Take Effect: Immediately

In URS environments with OCS, URS utilizes the agent assignment information provided by OCS and can delay routing decisions until the agents assignment information is provided. For cases when such delaying is considered undesirable, this option introduced in URS 8.1.400.88 can be used control the behavior. If the option is set to `false`, when agent assignment information is not provided URS will assume agent is not assigned to any outbound campaign. That is, when URS is working in an outbound environment, URS opens the agent assignment statistic for every tried agent and will route to an agent if the interaction activity matches with the activity assigned to the agent. The agent will not be routable until URS manages to open this statistic (StatServer sends a response when the

statistic is opened). Setting this option to `false` will result in URS considering the agent as assigned to default (inbound) activity and routing inbound interactions to the agent during time needed to open the statistic.

wait_time_prediction

Location in Configuration Layer by precedence: URS `Application` object
Default Value: `internal`
Valid Values: `internal`, `virtual`
Changes Take Effect: Immediately

This option controls which queue URS will use when it needs the average quit time of calls from some internal routing queue. You can specify the value for the option as internal (internal queue is used) or virtual (virtual queue associated with the internal routing queue is used). The default value is internal. The option is specified in the URS application object and changes take effect immediately.)

HTTP Bridge Updates

To communicate with Web Services through SOAP/XML and/or REST over HTTP/HTTPS protocols, URS uses a component called HTTP Bridge. HTTP Bridge allows strategy developers to communicate with Web Services applications outside of Genesys via the Web Service IRD strategy-building object. For more information on HTTP Bridge, see the Universal Routing 8.1 Reference Manual.

Support of HTTP Redirections

Starting with Release 8.1.400.23 in October of 2015, URS enhances its support of HTTP redirections. HTTP Bridge now resends an HTTP request to the new address specified in the Location header of the received 3xx response.

When responding to a redirect request, HTTP Bridge now:

- Supports both absolute and relative redirection URLs.
- Checks for redirect loops. No more than 5 chained redirects will be allowed.
- Always uses the GET method for redirection URLs if a return code is 303.

Handling of Escape Sequences

Starting with Release 8.1.400.14 in February 2015, HTTP Bridge no longer interprets XML escape sequences as regular delimiters (< and >) of XML tags. It now passes them into the strategy as is and does not terminate while processing XML data returned by SOAP-based Web Services containing XML escape sequences.

Add the following information to the Web Services Options section and also to Appendix B, IRD Web Services Object:

HTTP Bridge does not get completely XML-formatted text from a Web Service. The text is XML-formatted to some level, but at deeper levels, the XML text is escaped. For example:

```
<?xml version='1.0' encoding='utf-8'?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Body>
<ns:saveCaseResponse xmlns:ns="http://util.ilog.ist.apple.com">
<ns:return>
< ;?xml version="1.0" encoding="UTF-8"?> ;< ;GENESYS_CASE_CREATE_RESPONSE> ;.....> ;
</ns:return>
</ns:saveCaseResponse>
</soapenv:Body>
</soapenv:Envelope>
```

Note that the content inside the `ns:return` tag is not XML-formatted text. It might be interpreted as XML-formatted text after replacing the Escape sequences

```
< ; and > ;
```

with the appropriate < >.

HTTP Bridge does not perform this extra interpreting. Any XML response, if needed, must be in entirely and correctly-formed XML format.

Other Universal Routing 8.1.x Updates

This page contains other updates related to URS and IRD that are important to note.

URS Character Limitation

URS allows a maximum 1,023 characters for the combined length of a target name and a Stat Server name where the target name is the name of a corresponding Configuration Database object plus a `Skill` expression (if a `Skill` expression is used). Also see the Known Issues and Recommendations section of the Universal Routing Server 8.1.x release note.

Distribution of Multimedia Interactions During Shutdown or Backup Mode

Starting with 8.1.400.23, when URS shuts down or switches to backup mode, it distributes virtual queue events for multimedia interactions regardless of whether a backup URS exists. For voice calls, the URS behavior is the same as previous – virtual queue events are distributed only if there is no backup URS.

Removal of 9999 License Limit

Starting with 8.1.400.33, the maximum number of licenses to check out from License Server is extended from 9999 to 1,000,000 seats. Upon startup, URS checks out all available number of `router_seats` defined in License Server. When fewer licenses than available are needed, start URS with a new startup command line parameter: `-licnum <number of licenses>`.

Additional Information on HTTP Report Method

In addition to the information provided on the HTTP **Report** method on page 816 of the *Universal Routing 8.1. Reference Manual* (*Supported Methods* section in *Appendix C*), the **Report** method also supports the following input parameters:

- **ar** - information about agent reservation effectiveness.
- **seats** - provides information about URS licenses (router_seats) usage.

- **tserver** - name of T-Server for which information is required.

In the sample `tserver` method provided on the same page, note that timing related data in the response provides information about the average time (in milliseconds) that a particular category of calls spent being in one or another strategy execution state. When any function is executed and is waiting for an external event to trigger its continuation, the call is placed in a corresponding waiting state. During this stage, the following states are provided in reporting:

- **t** - time spent on mandatory treatments
- **x** - time spent waiting for data from external servers
- **s** - time spent waiting for data from statserver
- **w** - time spent waiting for ready targets, returned by the Wait function, which is used by target selection objects
- **r** - time spent waiting for the route used event
- **f** - time spent on all other waiting functions
- **n** - the time the category of calls were not in any waiting state, that is, time spent in active calls or spent doing nothing and not running any strategy.

New cpu Parameter

A new parameter, **cpu**, is introduced in URS release 8.1.400.63, to provide URS CPU consumption, which can be used for monitoring the health status of URS applications.

There are 2 categories of URS activities - main and background. Main activity is related to executing strategies, responding to requests, etc. Background activity is mostly related to updating content of skill groups. URS collects and stores its CPU usage information for the latest 60 seconds of its work.

Output Parameters:

- **base** - is average CPU consumption on main URS activity. This is the primary parameter to be used to indicate URS loading.
- **max** - is detected peak of total CPU consumption including both main and background activities.
- **base_max** - is detected peak of main CPU consumption.

An example is given below.

Command:
```
urs/stat/report?cpu
```

Output:
```
<cpu>
<base>60</base>
<max>70</max>
<base_max>70</base_max>
</cpu>
```

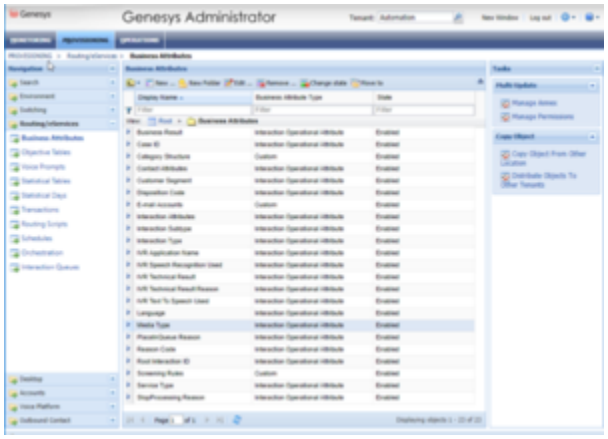
Maximum Length Limitation for Text Field on Web Service Object

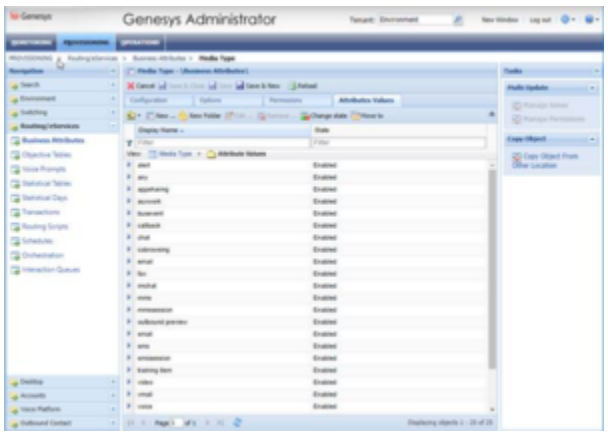
The maximum length of the input string in the **Text** field on the **General** tab of the **Web Service** object is limited to 1,010 bytes. (*Appendix B*, page 776 of the *Universal Routing 8.1 Reference Manual*)

Corresponding Genesys Administrator (GA) Screenshots for Old Configuration Manager (CME) Screenshots

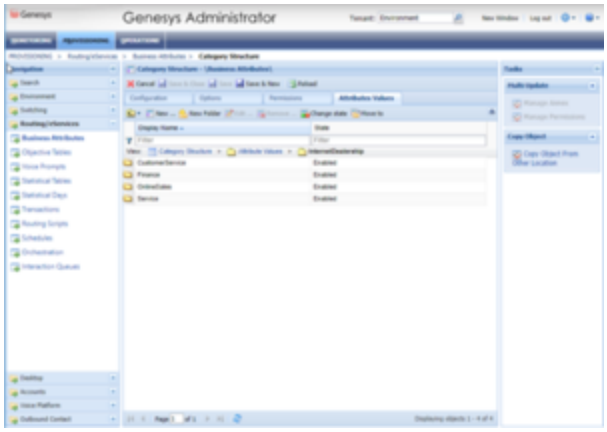
There are references to CME on various pages of the Universal Routing 8.1 Reference Manual. As GA is now more widely used than CME, the equivalent or corresponding GA screenshots for the old CME screenshots from the manual are provided below as a point of reference:

Page Number 102: Media Type Business Attributes and Attribute Values

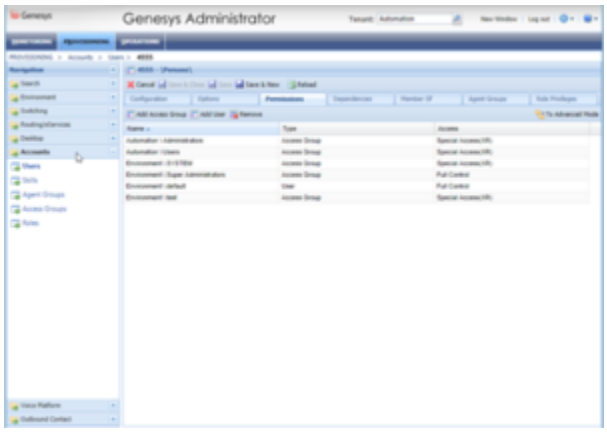




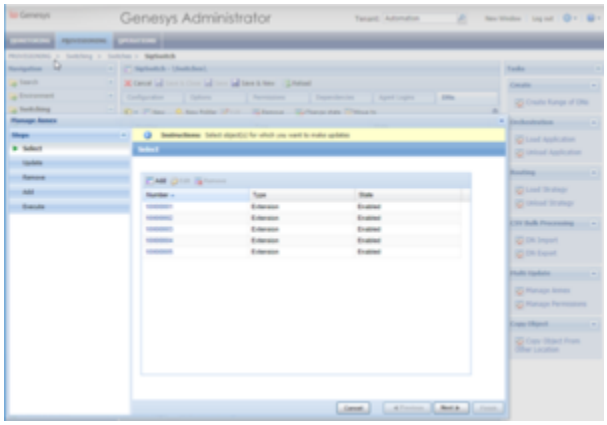
Page Numbers 178, 179, 183, and 201: Category Folders in Knowledge Manager



Page Number 419: Setting Permissions for a Person



Page Number 539: Updating Multiple Objects



Changes to the Limitations of Skill Expressions

The following limitations specified for skill expressions under the *IRD Limitations* section on pages 31 and 32 of the *Universal Routing 8.1 Reference Manual*, are no longer applicable:

- Routing objects cannot exceed 100 elements (skill names, numbers, comparisons, and logical operands).
- A skill expression should have no more than 25 constructions, such as `English > 1`.

However, there is a limit on the maximum number of characters allowed in a skill expression. A skill expression can have a maximum of **2933** characters.

URS REST API Security Considerations and Basic Hardening Steps

In addition to the information provided in the Security section, on page number 59 of the Universal Routing 8.1 Deployment Guide, the following recommendation is to be considered for the REST API.

> **Important**
> The REST API is an internal API and should be appropriately protected because it does not support common security headers in HTTP and does not have built in protections for features normally implemented in firewalls (such as DoS). The REST API is not intended to be exposed to untrusted parties.

It is possible that through the REST API provided by URS, sensitive data stored in strategies processing interactions might be accessed, and URS forced to perform resource-consuming activities (DoS attack).

Major security limitations of the RESTful API implementation are:

- No ability to provision HTTP responses with security headers of any kind.
- No firewall features of any kind (rate throttling, etc.).

Given the above, securing access to the URS web API is important.

Hardening Steps for URS REST API

You can perform the following steps to harden the URS REST API:

1. Provision TLS/SSL transport-level security for communications via HTTP and SOAP ports. This is configured in the Server Info tab of the router application as described in the Genesys Security Deployment Guide.
2. Configure the firewall to allow connections to URS ports only from 100% trusted zones with no exceptions. This is very important because, access to the URS HTTP port means access to all features of the URS REST API.

Evaluation of Skill Expressions

An important note on evaluation of skill expressions by URS:

During evaluation of skill expressions, URS tries to interpret any name/string from the expression (excluding function names) as the name of the skill.

- If the configuration does not have a skill with such a name, then the strings/names are interpreted literally.
- This means that if a skill expression has to contain a literal name (for instance, the name of a folder, name of a media, or simply a name), it should be verified that the configuration does not have a skill with the same name, as otherwise the value of the skill (or 0 if agent has no such skill) will be used instead of the literal name.