**URS Web API**

It is continuation of “Router Behind” API originally presented in 8.0.0 URS.

This API is essentially REST style one and no any SOAP counterparts are provided.

The exposed functionality is divided into specialized sub-modules: interactions, statistics, etc. Normally any method is accessible through both HTTP GET and POST commands.

Content-Type for POST commands expected to be **application/x-www-form-urlencoded**.

URS WEB API supports TLS and HTTP Authentication.

Also some API methods can be configured as disabled. In such case attempt to execute them will result in HTTP 403 error response with body **Method disabled**.

By default all methods except **Sending TRequests** are enabled. Enabling/disabling is performed with URS option **methods** in section **http**.

The value of option is concatenation of short methods names (or word **all**) optionally preceded with +(means enabled) or -(means disabled). No sign means +. Default value is **all-trek** meaning all methods except trek (= Sending TRequests) are enabled.

**1. URS WEB API - Interactions sub-module**

All methods of interaction sub-module have prefix urs/call. Methods of this sub-module are used to start new (web) “interactions” and/or communicate with existing ones (no matter was they started as web one or are regular TServer based interactions).

Most methods from this section require identification of interaction the action applied to (in methods description it is marked as **connid**). As such identification **ConnectionID** is used by default. CallUUID also can be used and in such case must be prefixed with @.

One more way to identify calls is call’s index. Index value assigned to call from strategy with function **CallAddIndex**[map, key]. After that call can be identified by construction **[table:key]** (it supposed to used instead of **connid** in requests).

Apart of that there are “discovery” methods that allow looking for interaction(s) having specific properties (see this sub-module methods **findall** and **findone**).

And finally URS can be configured to run “static” strategies for some tenants. Such “virtual” calls can provide build in background for running strategies outside of context of any specific call.

Such calls can be accessed by special identification in form of tenant name prefixed with underscore char: **\_TenantName**.

Self-awareness cluster of routers allows requesting any router about interactions processed by other routers.

1. **URS WEB API - Start new logical interaction or fork existing one**

This method instantiates new logical interaction (or session). It doesn't wait until session will be over but replies immediately with ConnectionID of new interaction. Interaction will exist on its own and is not internally bound to client that starts it. Client or any other component can communicate with this interaction on assumption it knows how to identify this interaction.

This method also can be used to start new session in context of already running interaction, resulting multiple parallel controls flows over the same interaction. As example - one flow to wait targets, second to play treatments, third to answer on external asynchronous events. It is the same functionality that is provided with router's function Fork (see URS Hints: 30. URS - how to run parallel (“multithreaded”) strategies? ). All parallel sessions share interaction's data and strategy variables (ones with INTERACTION scope).

To start new session :

urs/call/start?strategy=scriptname&tenant=tenantname&udata.key1=value1&...&param1=value1&...

urs/call/start?source=sourcecode&tenant=tenantname&udata.key1=value1&...&param1=value1&...

To fork existing session:

urs/call/connid/start?strategy=scriptname&param1=value1&...

urs/call/connid/start?source=sourcecode&param1=value1&...

Input:

connid: id of existing interaction. If provided will result forking - one more parallel control flow processing the specified interaction will be started.

tenant: optional, name of tenant the new interaction will belongs to (not applicable for forking case).

strategy: optional, name of Script object (Strategy or ECMAScript) or script URL (see Note 6) that will be started.

source: optional, source code (ECMAScript syntax only) of script. This is alternative to strategy way to specify script to run.

udata.somekey: optional, the value will become part of call's attached data (not applicable for case of forking).

param: optional, any set of input parameters that will be available to running session as extensions (can be accessed with function ExtensionDa

ta). Also see Note 1 and Note 3

Positive output:

Connection id of new interaction/session. Can be used later to communicate with this session.

Errors:

HTTP 404 code with body Call not found if call is not found (in case of forking only).

HTTP 404 code with body Object not found if URS failed to find requested script.

HTTP 500 code with body Internal error if URS failed to allocate resources for session.

HTTP 400 code with body Wrong request in all other cases when session was not started.

Note 1 (Not applicable to forking case) Extensions with following fixed names have special meaning.

- mediatype: default media type of new call

- ani: the value becomes call's ANI attribute

- dnis: the value becomes call's DNIS attribute

- ced: the value becomes call's CollectedDigits attribute

- mcr: if set to true or 1 then Interaction server will be assumed as transport layer for this logical interaction, otherwise (and by default) - TServer

- thislocation: the value defines switch name for this call. If present then on on resource allocation phase URS will try to invoke ISCC

GetAccessResource method on behalf of this switch name.

Note 2 If purpose of running script is allocation of available resource then script should have code to report to client allocated resource. For

example send some HTTP message containing information about selected target. There is also built-in functionality that might be used to address

this task (see Note 3).

Note 3 When running logical interactions router gives special meaning of some data stored in interaction's extensions. This special data can be

set through parameters of urs/call/start method as all parameters becomes extensions of interaction or set directly in script with function Extensi

onUpdate[key, value] or even just specified by URS options. This special extensions keys are: replyurl, replybody, replyack, replyenc.

When script execution allocates some resource and come to the point where URS issues routing request - instead of it URS checks for key replyu

rl in call's extensions (if not found then option default\_reply\_url will be checked). If found then URS automatically sends HTTP request to this

URL. The other 3 special keys can be used to set parameters of this HTTP request. if replyurl is not found URS will do nothing on routing phase

but just assumes that routing successfully happen.

- replyurl: identify URL the URS will use to communicate back information about selected target. URL can include as its parts information about

selected target.

- replybody: optional. If missed URS will use HTTP GET method. If provided HTTP POST will be used with this parameter value as POST

message body. Body can include inside information about selected target.

- replyenc: optional. Has sense only if replybody provided. This value will be used as value of Content-Type HTTP header in reply message. By

default application/x-www-form-urlencoded is used.

- replyack: optional. Values are false, true and reply, by default true. If set to true URS will wait confirmation on issued HTTP request. If error will

be reported back then URS will consider that resource allocation failed and behave

exactly as if EventError was reported on RequestRouteCall. If replyack is reply then URS additionally will wait separate replying web request

with information was routing successful or not. Format of replying request expected to

be urs/call/connid/reply/route?result=ok/error&param1=value1&.... If replyack is false then URS will not wait any confirmation and blindly

proceed with script execution (do postrouting).

In addition to option default\_reply\_url the following options accordingly exist: default\_reply\_body, default\_reply\_enc, default\_reply\_ack.

replyurl and replybody supposed to contain inside the information about selected target. For that the following formatting strings inside replyyurl a

nd replybody can be used:

[call.connid] - replaced with connection id of the interaction (this is the same value that method start returns).

[udata] or [udata.\*] or [udataj] or [udata.key] - extended to value of attached data (entire attached data in form of key=value pairs separated

by & or comma or in json form or just value of some specific key).

[ext] or [ext.\*] or [extj] or [ext.key] - the same about interaction extensions.

[target.\*] or [target.key] - extended to selected target (complete specification or some separate property like selected agent's name). The

following keys can be used:

- return, value - default, direct, target. Provide information how target was selected as default target, as result of force routing or normal target

selection procedure.

- dn, value - DN number that was selected.

- resource, values - DN number call was routed to (can differ from dn if external routing is used).

- switch, value - name of switch call was routed to.

- type, value -type of selected target: A, GA, etc.

- id, value - name of selected target, like name of agent group, etc.

- agent, value - selected agent.

- place, value - selected place.

- vq, value -name of virtual queue the selected target belongs to.

- stat\_value, value - statistic value the selected target has (if known).

- \*, all listed above values in comma separated list of pairs in format name=value

[target] - similar to [target.\*] but pairs are separated by & and always urlencoded

[targetj] - similar to [target.\*] but presents all targets parameters in JSON format

[targetx] - similar to [target.\*] but presents all targets parameters in XML format

Note 4 Other special extension data that URS recognizes are: notifyurl, notifybody, notifyenc. They are set up similarly to just

described reply keys and can compliment them in VCB like scenarios when using of allocated resource requires preliminary actions (like dialing).

If specified then URS will send notification to initiate "dialing" in advance before resource itself will be allocated. URS doesn't expect any

confirmation on these notifications. Notifications are desitributed only for interactions in DoNotSelectCall mode (set by executing DoNotSelectCal

l[true]). As at the moment of sending notification resource is not allocated yet notifyurl and notifybody must not contain target related formatting

strings.

Note 5 (Not applicable to forking case) If neither strategy nor source is provided then URS interprets request as request to find available target. In

such case it runs predefined GetTarget script:

var targetList= ExtensionData('TargetList');

var waitTime= ExtensionData('WaitTime');

var statistic= ExtensionData('Statistic');

var criteria= ExtensionData('Criteria');

var virtualQueue= ExtensionData('VirtualQueue');

var priority= ExtensionData('Priority');

var media= ExtensionData('MediaType');

if (media) UseMediaType(media);

target= SelectDN(virtualQueue, priority, statistic, criteria, targetList);

if (Failed()) target= SuspendForDN(waitTime);

RouteCall(target);

if (Failed()) Default();

It is expected that parameters of such start request will contain all needed data (and only mandatory one is TargetList).

Note 6 Parameter strategy also can have as value of URL pointing to ECMAScript (or other supported type) doc, for example strategy=http://loc

alhost:2082/SomeScript.js.

If scriptname is started with http:// then script content will be fetched from WEB server rather then taken from Config Server.

Samples:

Execute script CheckAgentLoadingMax and report result back through provided url:

http://urshost:2082/urs/call/start?tenant=Ten&strategy=CheckAgentLoadingMax&replyurl=http://router:8000/SelectedResource?[target]

029f028715e0d2cbIn result of execution if target was selected the following GET message might be sent Web service on router:8000 -

return=target&type=A&id=UN\_104\_vit\_sw2&agent=UN\_104\_vit\_sw2&place=Place\_104\_vit\_sw2&dn=104&switch=vit\_sw2&resource=104&vq=V

Q1\_vit\_sw2&stat\_value=128

Find available agent from provided TargetList and report result back through provided url:

http://urshost:2082/urs/call/start?tenant=Ten&TargetList=AgentGroup.GA&WaitTime=600&VirtualQueue=VQ1&replyurl=http://router:8000/[target]

029f028715e0d2ccIn result of execution if target was selected the following GET message might be sent Web service on router:8000 - return=t

arget&type=GA&id=AgentGroup&agent=UN\_104\_vit\_sw2&place=Place\_104\_vit\_sw2&dn=104&switch=vit\_sw2&resource=104&vq=VQ1

Attach to already running session some call "spying" process which will trace user data attaches (through

SuspendForEvent[EventAttachedDataChanged, ...]). Call is identified with its ANI:

http://urshost:2082/urs/call/[ani:12345678]/start?strategy=ReportUDataAttaching

029f028715e0d2cd

See also:

09. URS WEB API - Querying interaction.

02. URS WEB API - Execute new logical interaction.

03. URS WEB API - Terminating of interaction processing

URS Hints: 30. URS - how to run parallel (“multithreaded”) strategies?

02. URS WEB API - Execute new logical interaction.

This method is very similar to 01. URS WEB API - Start new logical interaction or fork existing one method - it also creates new logical

interaction (or session) and accepts the same set of input parameters.

The main difference - method will wait until session is over before replying to client. Result of script execution is reported to client as HTTP

response.

Just like with 01. URS WEB API - Start new logical interaction or fork existing one new session can be started in context of existing

interaction (=forked from another session). It results multiple parallel controls flows over the same interaction. Forking in context of this method

also makes it similar to 04. URS WEB API - Strategy injection. method but without restrictions on executed functions.

Executing

urs/call/exec?strategy=scriptname&tenant=tenantname&udata.key1=value1&...&param1=value1&...

urs/call/exec?source=sourcecode&tenant=tenantname&udata.key1=value1&...&param1=value1&...

Forking:

urs/call/connid/exec?strategy=scriptname&param1=value1&...

urs/call/connid/exec?source=sourcecode&param1=value1&...

Input:

See 01. URS WEB API - Start new logical interaction or fork existing one method

Positive output:

JSON object having as properties all call's extensions at the moment of script ending (set by running script implicitly or explicitly).

Errors:

HTTP 404 code with body Call not found if call is not found (in case of forking only).

HTTP 404 code with body Object not found if URS failed to find requested script.

HTTP 500 code with body Internal error if URS failed to allocate resources for session.

HTTP 400 code with body Wrong request in all other cases when session was not started.

Note 1 Just like for 01. URS WEB API - Start new logical interaction or fork existing one method parameters strategy and/or source are

optional.

Samples:

Execute script CheckAgentLoadingMax and report result back:

http://urshost:2082/urs/call/exec?tenant=Ten&strategy=CheckAgentLoadingMax

{"Agent1":1,"Agent2":1}

Find available agent from provided TargetList and report result back:

Check that among all agents with required skill at least one logged in:

http://urshost:2082/urs/call/exec?tenant=Ten&source=x=GetSkillInGroupEx('','','Deutsch>5',true);n=CountTargetsByThresholdEx(x,'StatAg

entsTotal',0,1,1);ExtensionUpdate('answer',n>0?'yes':'no');Exit();

{"answer":"yes"}

See also:

01. URS WEB API - Start new logical interaction or fork existing one

URS Hints: 30. URS - how to run parallel (“multithreaded”) strategies?

[03. URS WEB API - Terminating of interaction processing.](https://intranet.genesys.com/pages/viewpage.action?pageId=56825322)

This method terminates running script for provided interaction. Interaction can be logical started with [**01. URS WEB API - Start new logical interaction or fork existing one**](https://intranet.genesys.com/pages/viewpage.action?pageId=56823933) method or just regular TServer's interaction. In last case it doesn't mean terminating of interaction itself but only terminating of script processing the call. This method also can be used to reset running of script for TServer's interaction.

**urs/call/connid/terminate?reset=true/false&abandoned=true/false**

Input:

- connid: id of interaction the request will be applied to.

- reset: optional, by default false, results in sending TQueryCall request to TServer about the interaction (see Note 1).

- abandoned: optional, by default false, results in terminated interaction will be considered by URS as abandoned.

Positive output:

OK

Errors:

HTTP 404 code with body **Call not found** if call is not found

**Note 1** TQueryCall request if confirmed with TServer results in recreating the interaction and starting new session for it from the beginnig.

**Samples**:

<http://urshost:2082/urs/call/006b01fff974500b/terminate>

OK

**See also**:

[**01. URS WEB API - Start new logical interaction or fork existing one**](https://intranet.genesys.com/pages/viewpage.action?pageId=56823933)

[**09. URS WEB API - Querying interaction.**](https://intranet.genesys.com/pages/viewpage.action?pageId=57809155)

[04. URS WEB API - Strategy injection.](https://intranet.genesys.com/pages/viewpage.action?pageId=56825404)

The method allows to communicate with interaction transparently (invisibly) for main script that processes the interaction. If invoked it results in execution of some logic (subroutine) in context of specified call. Main script processing the call is suspended (interrupted) while specified subroutine runs and resumed afterwards. Executed logic can be collecting some data about running interaction and/or some corrective actions - changing priorities, set of targets, attaching data, etc.

**urs/call/connid/invoke**?strategy=scriptname&sync=0/1&param1=value1&...

**urs/call/connid/invoke**?source=sourcecode&sync=0/1

Input:

- connid: id of interaction the request will be applied to.

- strategy: name of Script object (Subroutine or ECMAScript) to run.

- source: optional, source code (ECMAScript syntax only) of script. This is alternative to strategy way to specify script to run (see Note 2).

- sync: optional, by default 0, synchronization with backup router.

- param: any set of parameters that might be passed to subroutine. Subroutine's input parameters will be automatically initiated with parameters values taken from this request parameters on assumption parameters names match to name of subroutine input parameters.

Positive output:

JSON presentation of object having as properties the subroutine's output parameters (also see Note 2)

Errors:

HTTP 404 code with body **Call not found** if call is not found

HTTP 404 code with body **Object not found** if URS failed to find requested script.

HTTP 500 code with body **Internal error** if router fails to execute subroutine or subroutine explicitly returns error.

**Note 1** Set of possible functions that might be executed from inside subroutine is limited - URS raises error on attempt to execute any lengthy operation from inside such subroutine. It means that is not possible to directly from subroutine body invoke operation like routing, applying treatments, querying external data source (database, web services, external services).

**Note 2** If source is used to specify script to run then following rules are applied:

- script passes result of execution through **out** variable

- no param is accessible from script

**Samples**:

<http://urshost:2082/urs/call/006b01fff974500b/invoke?strategy=Add&arg1=10&arg2=20>  
{"Sum":"30"}

**See also**:

[**05. URS WEB API - Remote function invocation.**](https://intranet.genesys.com/pages/viewpage.action?pageId=56825429)

[05. URS WEB API - Remote function invocation.](https://intranet.genesys.com/pages/viewpage.action?pageId=56825429)

This method is simplified version of [**04. URS WEB API - Strategy injection**](https://intranet.genesys.com/display/RP/04.+URS+WEB+API+-+Strategy+injection) when instead of entire script just single function is invoked. This method is executed in the same way and the same restrictions are applied.

**urs/call/connid/func**?name=function&sync=0/1&params=values

**urs/call/connid/func/function**?sync=0/1&params=values

Input:

- connid: id of interaction the method will be applied to.

- name/function: name of function to execute.

- sync: optional, by default 0, synchronization with backup router.

- params: the JSON array of function parameters.

Positive output:

JSON presentation of value returned by function.

Errors:

HTTP 404 code with body **Call not found** if call is not found

HTTP 500 code with body **Internal error** if router fails to execute function or function explicitly returns error.

**Note 1** Check for quotes->double quotes conversion when mapping between invoking function in IRD and through web API. Otherwise else mapping is straightforward.

IRD: CountSkillInGroupEx['statserver','Agents\_sw2','Sales > 5',false]

API: **/urs/call/006b01fff974500b/func/**CountSkillInGroupEx**?params=**["statserver","Agents\_sw2","Sales > 5",false]

**Samples**:

http://urshost:2082/urs/call/006b01fff974500b/func/ExtensionData?params=["SomeKey"]

"QWERTY"

**See also**:

[**04. URS WEB API - Strategy injection**](https://intranet.genesys.com/display/RP/04.+URS+WEB+API+-+Strategy+injection)

[06. URS WEB API - Changing interaction attached data](https://intranet.genesys.com/display/RP/06.+URS+WEB+API+-+Changing+interaction+attached+data)

This method changes interaction attached data. It is shortcut method for performing the same task with [**04. URS WEB API - Strategy injection**](https://intranet.genesys.com/display/RP/04.+URS+WEB+API+-+Strategy+injection) or [**05. URS WEB API - Remote function invocation.**](https://intranet.genesys.com/pages/viewpage.action?pageId=56825429).

**urs/call/connid/update**?sync=0/1&key1=value1&key2=value2&...

Input:

   - connid: id of interaction the request will be applied to.

   - sync: optional, by default 0, synchronization with backup router, use for virtual calls only.

   - keys: attached data to be set or changed.

Positive output:

OK

Errors:

HTTP 404 code with body **Call not found** if call is not found

**Samples**:

<http://urshost:2082/urs/call/006b01fff974500b/update?MyData1=123&MyData2=qwerty>  
OK

**See also**:

[**04. URS WEB API - Strategy injection**](https://intranet.genesys.com/display/RP/04.+URS+WEB+API+-+Strategy+injection)

[**05. URS WEB API - Remote function invocation.**](https://intranet.genesys.com/pages/viewpage.action?pageId=56825429)

[07. URS WEB API - Changing interaction extensions](https://intranet.genesys.com/display/RP/07.+URS+WEB+API+-+Changing+interaction+extensions)

This method changes interaction extension data. It is shortcut method for performing the same task with [**04. URS WEB API - Strategy injection**](https://intranet.genesys.com/display/RP/04.+URS+WEB+API+-+Strategy+injection) or [**05. URS WEB API - Remote function invocation.**](https://intranet.genesys.com/pages/viewpage.action?pageId=56825429)

**urs/call/connid/update**?sync=0/1&key1=value1&key2=value2&...

Input:

- connid: id of interaction the request will be applied to.

- sync: optional, by default 0, synchronization with backup router, use for virtual calls only.

- keys: extensions data to be set or changed.

Positive output:

OK

Errors:

HTTP 404 code with body **Call not found** if call is not found

**Samples**:

<http://urshost:2082/urs/call/006b01fff974500b/update?MyData1=123&MyData2=qwerty>  
OK

**See also**:

[**04. URS WEB API - Strategy injection**](https://intranet.genesys.com/display/RP/04.+URS+WEB+API+-+Strategy+injection)

[**05. URS WEB API - Remote function invocation.**](https://intranet.genesys.com/pages/viewpage.action?pageId=56825429)

[08. URS WEB API - Accessing attached data.](https://intranet.genesys.com/pages/viewpage.action?pageId=57809133)

This method query interaction attached data. It is shortcut method for performing the same task with [**04. URS WEB API - Strategy injection**](https://intranet.genesys.com/display/RP/04.+URS+WEB+API+-+Strategy+injection) or [**05. URS WEB API - Remote function invocation.**](https://intranet.genesys.com/pages/viewpage.action?pageId=56825429)

**urs/call/connid/udata**?key1&key2&...

Input:

- connid: id of interaction the query will be applied to.

 - keys: attached data to be returned. If not provided then all user data will be returned.

Positive output:

JSON presentation of object having as properties the asked attached data keys

Errors:

HTTP 404 code with body **Call not found** if call is not found

**Samples**:

<http://urshost:2082/urs/call/006b01fff974500b/udata?MyData1&MyData2>  
{"MyData1":"18001234567","MyData2":14}

**See also**:

[**04. URS WEB API - Strategy injection**](https://intranet.genesys.com/display/RP/04.+URS+WEB+API+-+Strategy+injection)

[**05. URS WEB API - Remote function invocation.**](https://intranet.genesys.com/pages/viewpage.action?pageId=56825429)

[09. URS WEB API - Querying interaction.](https://intranet.genesys.com/pages/viewpage.action?pageId=57809155)

The quick and having no side effects way to find out does URS process the interaction or not.

**urs/call/connid/query**

Input:

 - connid: id of interaction the query will be applied to.

Positive output:

JSON presentation of object having as properties different information about the call

   - state - current routing state of call.

   - time - time in seconds URS run script for this call.

   - priority - current global call's priority.

   - rvq\_ewt - aggregated expected waiting time calculated based on agents average talking time.

   - min\_rvq\_ewt - minimal expected waiting time calculated based on agents average talking time.

   - min\_ewt - minimal expected waiting time calculated based on quitting rates.

   - max\_ewt - maximal expected waiting time calculated based on quitting rates.

Errors:

HTTP 404 code with body **Call not found** if call is not found

**Samples**:

<http://urshost:2082/urs/call/006b01fff974500b/query>  
{"state":4,"time":40,"prioroty":0,"rvq\_ewt":86,"min\_rvq\_ewt":86,"min\_ewt":86,"max\_ewt":86}

**See also**:

[**01. URS WEB API - Start new logical interaction or fork existing one**](https://intranet.genesys.com/pages/viewpage.action?pageId=56823933)

[**03. URS WEB API - Terminating of interaction processing**](https://intranet.genesys.com/display/RP/03.+URS+WEB+API+-+Terminating+of+interaction+processing)

[**URS EWT**](https://intranet.genesys.com/display/RP/URS+EWT)

[10. URS WEB API - Querying calls queuing information in virtual queue(s)](https://intranet.genesys.com/pages/viewpage.action?pageId=57831656)

Method returns information about call positioning in one or more virtual queues or group of virtual queues. It can be considered as more detailed version of [**09. URS WEB API - Querying interaction.**](https://intranet.genesys.com/pages/viewpage.action?pageId=57809155)Also for more detailed information about URS provided EWT see  [**URS EWT**](https://intranet.genesys.com/display/RP/URS+EWT).

**urs/call/connid/lvq**?name=vqname&filter=string&ewt=min/max&aqt=urs/urs2/stat&scale=true/false

**urs/call/connid/lvq/vqname**?filter=string&ewt=min/max&aqt=urs/urs2/stat&scale=true/false

Input:

- connid: id of interaction the query will be applied to. Special id max can be used for hypothetical "very last" interaction in virtual queue (See Note 5).

- name/vqname: name of Virtual Queue (see Note 6) the request is related with. May include '\*' wildchar at the beginning and/or at the end. If not specified then '\*' will be used.

- filter: comma separated list of properties (see below properties of returned JSON object). If used then only specified fields will be returned.

- ewt: can have values min or max. If provided only information about single virtual queue will be provided - the one having min or max ewt.

- aqt: can have values urs, urs2 or stat. If set to stat then router will try to get aqt from statserver instead of counting internally.

- scale: can have values true or false. If true then returned ewt will be scaled with probability of success of VCB calls and with routing delays if any.

- tenant: required in cases when connid=max (See Note 5).

Positive output:

If wildchars not used - JSON presentation of object describing position of call in the virtual queue. Otherwise - collection of such objects (one per every matched virtual queue the call is in). Object describing call's position in virtual queue has properties: - time - UTC time stamp when call enters the virtual queue.

- wt - time in seconds call is in the virtual queue (effectively CurrentTime-time).

- calls - current number of all calls in the virtual queue.

- wcalls - current number of waiting calls in the virtual queue (see Note 1).

- pos - position of the call in the virtual queue.

- wpos - waiting position of the call in the virtual queue. (see Note 1)

- priority - calls priority in the virtual queue. Absent if scaling is used.

- aqt - average quitting rate of calls from the virtual queue. May be skipped if unknown (see Note 2).

- ewt - expected waiting time for the call (aqt \* wpos or aqt \* scaled\_wpos if scaling is used). May be skipped if unknown.

- hit - percentage of calls distributed into the virtual queue.

- guid - UUID of entering the call into the virtual queue. This property must be asked explicitly and separately from others.

Errors:

HTTP 404 code with body **Call not found** if call is not found

**Note 1**

Even after resource allocation corresponding call remains in queue untill allocation will be confirmed (for example with EventRouteUsed). Such calls while remaining in queue are not considered as "waiting" calls.

**Note 2**If input aqt set to stat and Virtual Queue has preconfigured target(s) then returned aqt value is StatExpectedWaitingTime/StatCallsInQueue.

If input aqt set to stat and Virtual Queue doesn't have preconfigured target(s) then URS get it itself based on average handling time of all involved agents.

If input aqt set to urs2 then returned aqt URS get itself based on average handling time of all involved agents.

If input aqt set to urs or skipped then returned aqt is average time between subsequent events EventDiverted distributed by URS for this Virtual Queue.

**Note 3**

Scaling will result in attributes calls, wcalls, pos, wpos to have paired scaled attributes: scaled\_calls, scaled\_wcalls, scaled\_pos, scaled\_wpos.

**Note 4**

If call is in routing delay state then time will be accompanied with dnr\_time having as value UTC time stamp of routing delay expiring. Also routing delay can result in failure to get call's ewt. In such cases instead of ewt the dnr data with value of routing delay time (effectively dnr\_time-CurrentTime) will be returned.

**Note 5**

Querying call queuing information can be done even before call is really placed into queue - if max is used as value of connid. In such case router will work as if it has extra interaction which is queued right now with lowest possible priority in specified virtual queue(s). Extra input parameter tenant must be provided in such case. Output parameter guid will not be available.

**Note 6**

If name ends with .GQ then it will be interpreted as name of group of virtual queues. Call positioning in entire virtual queue group will be collected. If applied to group of virtual queues method will ignore wildchars and input keys ewt, aqt (urs2 forced) as well as result will not contain time, wt, hit and guid.

**Note 7** Virtual queues are logical entities presenting one or more router's internal queues (internal queue presents some target selection object used in some strategy). Querying information for internal queues is also possible, see [**11. URS WEB API - Querying calls queuing information in internal queue(s)**](https://intranet.genesys.com/pages/viewpage.action?pageId=57831718).

**Samples**:

*Get call queueing information for all virtual queues the call is in:*<http://urshost:2082/urs/call/006b01fff974500b/lvq>

{"VQ2":{"time":1468621149,"wt":258,"calls":2,"wcalls":2,"pos":2,"wpos":2,"priority":0,"aqt":7,"ewt":14,"hit":75}}

*Get call queueing information for virtual queue with minimal call's ewt (among all virtual call's queues):*<http://urshost:2082/urs/call/006b01fff974500b/lvq?ewt=min>  
[{"VQ2":{"time":1468621141,"wt":479,"calls":2,"wcalls":2,"pos":1,"wpos":1,"priority":0,"aqt":7,"ewt":7,"hit":75](http://urshost:2082/urs/call/006b01fff974500b/lvq?ewt=min)}}

*Get queueing information for specific virtual queue and out of context of any call:*<http://urshost:2082/urs/call/max/lvq/MyVQ?tenant=SomeTenant>  
{"time":1468621222,"wt":0,"calls":2,"wcalls":2,"pos":3,"wpos":3,"aqt":7,"ewt":21,"hit":75}

**See also**:

[**11. URS WEB API - Querying calls queuing information in internal queue(s)**](https://intranet.genesys.com/pages/viewpage.action?pageId=57831718)[**URS EWT**](https://intranet.genesys.com/display/RP/URS+EWT)

[Querying router's queues](http://localhost:2082/urs/help/stat/targetqueueinfo)

[11. URS WEB API - Querying calls queuing information in internal queue(s)](https://intranet.genesys.com/pages/viewpage.action?pageId=57831718)

This method provides information about call position inside router internal queue(s). Internal queue normally presents some routing object (or target selecting function) used in some strategy. It contains all agents targeted with this routing object and all calls waiting inside it. Every internal queue usually is included into some virtual queue (the one they refer to). For querying virtual queues information see [**10. URS WEB API - Querying calls queuing information in virtual queue(s)**](https://intranet.genesys.com/pages/viewpage.action?pageId=57831656).

Also for more detailed information about URS provided EWT see  [**URS EWT**](https://intranet.genesys.com/display/RP/URS+EWT).

**urs/call/connid/rvqdata**?id=label&filter=string

**urs/call/connid/rvqdata/label**?filter=string

Input:

- connid: id of interaction the query will be applied to.

- id/label: id of internal queue the call is in: either label or numeric id. If absent then data about all internal queues call is in will be provided.

- filter: comma separated list of properties of call entrance in internal queue (see below properties of returned JSON object). If used then only specified fields will be returned.

Positive output:

JSON presentation of object or array of objects having as properties different information about entering call into specific internal queue as well as some parameters of this internal queue

- aht - average handling time in the queue. Counted by URS based on average talking time of every involved agent.

- pos - position of the call in the internal queue.

- qlen - number of calls in the internal queue.

- ewt - aht multiplied by pos

- qewt - aht multiplied by qlen+1

- quit - average quitting rate of calls from the internal queue.

- size - number of agents serving the internal queue

- insize - number of logged in agents serving the internal queue

- priority - priority of call in the internal queue

- wt - time in seconds call is already waiting in this internal queue

- time - UTC time stamp when call was placed into this internal queue (with millisecond precision)

- id - numerical id of the call in the internal queue

If the internal queue is not found then empty object is returned.

Errors:

HTTP 404 code with body **Call not found** if call is not found

**Samples**:

<http://urshost:2082/urs/call/006b01fff974500b/rvqdata?id=abc>  
{"aht":26,"pos":1,"qlen":2,"ewt":26,"qewt":78,"quit":8.19,"size":6,"insize":1,"priority":0,"time":1468623066.92,"wt":47,"id":3}

**See also**:

[**10. URS WEB API - Querying calls queuing information in virtual queue(s)**](https://intranet.genesys.com/pages/viewpage.action?pageId=57831656)

[**URS EWT**](https://intranet.genesys.com/display/RP/URS+EWT)

[12. URS WEB API - Sending request to interaction](https://intranet.genesys.com/display/RP/12.+URS+WEB+API+-+Sending+request+to+interaction)

Method provides way to communicate with already running script for some call. If client knows interaction id of running interaction then this method can be used to send message to interaction and get back response. Script for this interaction must contain code expecting external requests and answers on them (functions **GetExternalEvent**, **AnswerExternalEvent**). Request **answering** is the only difference from sending event to interaction, see [**13. URS WEB API - Sending event to interaction**](https://intranet.genesys.com/display/RP/13.+URS+WEB+API+-+Sending+event+to+interaction).

Also see URS Hints [**29. URS - how to process web requests from strategy?**](https://intranet.genesys.com/pages/viewpage.action?pageId=55836419).

**urs/call/connid/request/name**?**param**=value&...

Input:

- connid: id of interaction the request will be sent to.

- name: name of request. It will be accessible from script.

- param: arbitrary set of keys with values. All parameters will be accessible from script.

Positive output:

Whatever script will choose to return

Errors:

HTTP 404 code with body **Call not found** if call is not found

HTTP 429 code with body **Overflow** if script doesn't answer promptly

HTTP 400 code with body **Wrong request** on any other errors

**Note 1** Script also has option to return error code back instead of positive response

**Note 2** Code for processing external requests can complicate script. Therefore recommended ways to communicate with running scripts is either [**04. URS WEB API - Strategy injection**.](https://intranet.genesys.com/pages/viewpage.action?pageId=56825404) method or adding extra control flow for processing external requests with [**02. URS WEB API - Execute new logical interaction.**](https://intranet.genesys.com/pages/viewpage.action?pageId=56824085) method.

**Samples**:

<http://urshost:2082/urs/call/006b01fff974500b/request/tts?URL_ID=http://aaaaaaa>  
id=285

**See also**:

[**13. URS WEB API - Sending event to interaction**](https://intranet.genesys.com/display/RP/13.+URS+WEB+API+-+Sending+event+to+interaction)

[**02. URS WEB API - Execute new logical interaction.**](https://intranet.genesys.com/pages/viewpage.action?pageId=56824085)

[**04. URS WEB API - Strategy injection**.](https://intranet.genesys.com/pages/viewpage.action?pageId=56825404)

[**URS Hints: 29. URS - how to process web requests from strategy?**](https://intranet.genesys.com/pages/viewpage.action?pageId=55836419)

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:trackback="http://madskills.com/public/xml/rss/module/trackback/"> <rdf:Description rdf:about="https://intranet.genesys.com/display/RP/12.+URS+WEB+API+-+Sending+request+to+interaction" dc:identifier="https://intranet.genesys.com/display/RP/12.+URS+WEB+API+-+Sending+request+to+interaction" dc:title="12. URS WEB API - Sending request to interaction" trackback:ping="https://intranet.genesys.com/rpc/trackback/57832557"/> </rdf:RDF>

[13. URS WEB API - Sending event to interaction](https://intranet.genesys.com/display/RP/13.+URS+WEB+API+-+Sending+event+to+interaction)

Method provides way to communicate with already running script for some call. If client knows interaction id of running interaction then this method can be used to send event to interaction. Script for this interaction must contain code expecting external requests (functions **GetExternalEvent, etc**), see URS Hints [**29. URS - how to process web requests from strategy?**](https://intranet.genesys.com/pages/viewpage.action?pageId=55836419). The method will return immediately without waiting any script reaction on the event (that is the only difference from sending request to interaction method, see [**12. URS WEB API - Sending request to interaction**](https://intranet.genesys.com/display/RP/12.+URS+WEB+API+-+Sending+request+to+interaction)).

**urs/call/connid/event/name**?param=value&...

Input:

 - connid: id of interaction the event will be sent to.

 - name: name of request. It will be accessible from script.

 - param: arbitrary set of keys with values. All parameters will be accesible from script.

Positive output:

OK

Errors:

   HTTP 404 code with body **Call not found** if call is not found

   HTTP 429 code with body **Overflow** if number of not yet processed events to the interaction is too big

   HTTP 400 code with body **Wrong request** on any other errors

**Note** Code for processing external requests can complicate script. Therefore recommended ways to communicate with running scripts is either [**04. URS WEB API - Strategy injection**.](https://intranet.genesys.com/pages/viewpage.action?pageId=56825404) method or adding extra control flow for processing external requests with [**01. URS WEB API - Start new logical interaction or fork existing one.**](https://intranet.genesys.com/pages/viewpage.action?pageId=56823933) method.

**Samples**:

<http://urshost:2082/urs/call/006b01fff974500b/event/tts?URL_ID=http://aaaaaaa>  
OK

[14. URS WEB API - Changing or reading interaction variables](https://intranet.genesys.com/display/RP/14.+URS+WEB+API+-+Changing+or+reading+interaction+variables)

This method allows changing or just reading values of strategies variables which have INTERACTION scope. It can be used to inject **data** into instance of running script in the same way method [**04. URS WEB API - Strategy injection.**](https://intranet.genesys.com/pages/viewpage.action?pageId=56825404) can be used to inject some **logic**.

**urs/call/connid/vars**?sync=0/1&var1=value1&var2=value2&...

Input:

- connid: id of interaction the method will be applied to.

- sync: optional, by default 0, synchronization with backup router.

- vars: names of variables to be set ot read. By default type of variable is string. Using prefix {d} or {f} it can be changed to integer or float. Assignment part is optional and if skipped will result in just reading of the variable.

Positive output:

JSON presentation of object having as properties listed strategies variables

Errors:

HTTP 404 code with body **Call not found** if call is not found

**Samples**:

http://urshost:2082/urs/call/006b01fff974500b/vars?{d}MyVar1=123&MyVar2=qwerty  
{"MyVar1":123,"MyVar2":"qwerty"}

http://urshost:2082/urs/call/006b01fff974500b/vars?{d}MyVar1&MyVar2  
{"MyVar1":123,"MyVar2":"qwerty"}

|  |
| --- |
| [15. URS WEB API - Placing call in queue](https://intranet.genesys.com/display/RP/15.+URS+WEB+API+-+Placing+call+in+queue?src=mail&src.mail.timestamp=1481590407344&src.mail.notification=com.atlassian.confluence.plugins.confluence-content-notifications-plugin%3Apage-created-notification&src.mail.recipient=ff80808140ae155a0140ae15bf6b2db7&src.mail.action=view) |
| This method makes interaction wait (and potentially be routed to) extra target(s). It accepts parameters similar to ones of "GetTarget" method (see [**01. URS WEB API - Start new logical interaction or fork existing one.**](https://intranet.genesys.com/pages/viewpage.action?pageId=56823933)). It's shortcut method for performing the same task with [**04. URS WEB API - Strategy injection.**](https://intranet.genesys.com/pages/viewpage.action?pageId=56825404) or [**05. URS WEB API - Remote function invocation.**](https://intranet.genesys.com/pages/viewpage.action?pageId=56825429)  **urs/call/connid/addqueue**?TargetList=listoftargets&Statistic=statname&Criteria=min/max&VirtualQueue=vqname&Priority=value  Input:  - connid: id of interaction the method will be applied to.  - TargetList: comma separated list of targets.  - Statistic: name of statistic used to select target.  - Criteria: max or min.  - VirtualQueue: name of virtual queue.  - Priority: priority value.  Positive output:  number, id of entrance of call in the queue.  Errors:  HTTP 404 code with body **Call not found** if call is not found  HTTP 500 code with body **Internal error** if router fails to execute function or function explicitly returns error.  **Samples**:  somebody wants to redirect call to himself  <http://urshost:2082/urs/call/006b01fff974500b/addqueue?TargetList=MySelf.A>  3794  [16. URS WEB API - Sending TEvent to interaction](https://intranet.genesys.com/display/RP/16.+URS+WEB+API+-+Sending+TEvent+to+interaction)  This method makes URS to receive provided TEvent for given interaction. TEvent is specified similarly to [**04. URS WEB API - Sending TRequests**](https://intranet.genesys.com/display/RP/04.+URS+WEB+API+-+Sending+TRequests) method. Effectively this is method to send some TEvent not really using TServer but directly injecting it into URS handling the interaction (No other applications except this URS instance will receive this TEvent).  **urs/call/connid/tevent**?event=event&tserver=tservername&switch=switchname&param1=value1&...  Input:  - connid: id of interaction the event will be sent to.  - event: TEvent to URS. It can be provided as name or as number.  - tserver: optional, name of TServer on behalf of which TEvent will be sent.  - switch: optional, can be used to detect TServer on behalf of which TEvent will be sent.  - params: optional specify attributes of TEvent. They can be provided as names (TLibrary names without Attribute prefix) or as numbers.  Positive output:  OK  Errors:     HTTP 404 code with body **Call not found** if call is not found     HTTP 400 code with body **Object not found** if URS failed to find specified tserver     HTTP 400 code with body **Wrong request** on any other errors  **Note 1** Following attributes will be automatically added to generated TEvent if not provided explicitly: AttributeConnID, AttributeThisDN, AttributeCallID, AttributeCallType and AttributeCustomerID.  **Samples**:  <http://urshost:2082/urs/call/006b01fff974500b/tevent?event=EventTreatmentEnd>  [17. URS WEB API - Finding/Acting on interaction with specific properties](https://intranet.genesys.com/pages/viewpage.action?pageId=57832629)   * Created by [Vitaliy Teryoshin](https://intranet.genesys.com/display/~Vitaliy.Teryoshin@genesys.com), last modified on [Dec 14, 2016](https://intranet.genesys.com/pages/diffpagesbyversion.action?pageId=57832629&selectedPageVersions=2&selectedPageVersions=3)   Go to start of metadata  Like [**04. URS WEB API - Strategy injection.**](https://intranet.genesys.com/pages/viewpage.action?pageId=56825404)this method applies subroutine with parameters to the calls. The purpose of running subroutine is different however - URS applies it  to every **not yet routed** call (so it is sort of injections applied to all calls in memories of all URSes from selfawareness cluster). Subroutine checks interaction properties (though it can  also perform some correction actions). If subroutine exits without raising any error then its output will be returned as result of entire find method and URS will not try any others calls.  If subroutine will exit with errors router will proceed with next call and so on. If for none of them subroutine exit positively then HTTP 404 error will be returned. This is one of few web methods  from Interaction sub-module which doesn't require **connid**.  **urs/call/findone**?strategy=scriptname&tenant=tenantname&target=targetid&maxdelay=num&param1=value1&...  **urs/call/findone**?source=sourcecode&tenant=tenantname&target=targetid&maxdelay=num  Input:   - strategy: name of Script object (Subroutine or ECMAScript) to run.   - source: optional, source code (ECMAScript syntax only) of script. This is alternative to strategy way to specify script to run (see Note 2).   - tenant: name of tenant calls and strategy belongs to.   - target: optional, in format [name@statserver.type](mailto:name@statserver.type), if provided then only calls waiting this target (in priority order) will be tried.   - maxdelay: optional, default 4, provide maximum time in seconds to collect data from other URS nodes.   - param: any set of parameters that might be passed to subroutine. Subroutine's input parameters will be automatically initiated with parameters values taken from this request parameters on assumption parameters names match to name of subroutine input parameters.  Positive output:  JSON presentation of object having as properties the subroutine output parameters (also see Note 2)  Errors:  HTTP 404 code with body **Call not found** if no call exist for which was possible to successfully run subroutine and have positive response  HTTP 404 code with body **Object not found** if specified subroutine or target is not found  **Note 1** Extra parameter with name **call\_pos** will be added to parameters before trying every next call. Its value starts from 1 and is incremnented by 1 for every next call.  **Note 2** If source is used to specify script to run then following rules are applied: - script passes result of execution through **out** variable, any value different from **undefined** or **null** interpreted as success and used as returned value - no param is accessible from script - **call\_pos** value can be taken from stack with Pop() function.  **Samples**:  Script SID checks if call is located on specified RP and if yes then returns its attached data with key SID. Then to find attached data of some interaction (if any) on RP 2203:  <http://urshost:2082/urs/call/findone?strategy=SID&dn=2203> {"SID":"029LK70N8SBO9F2B04000VTAESXX000F"}or (the same)  <http://urshost:2082/urs/call/findone?source=>**if(Dest()=='2203') out={"SID":UData("SID")};** {"SID":"029LK70N8SBO9F2B04000VTAESXX000F"}  **See also**:  [**04. URS WEB API - Strategy injection.**](https://intranet.genesys.com/pages/viewpage.action?pageId=56825404)  [**18. URS WEB API - Finding/Acting on all interactions with specific properties**](https://intranet.genesys.com/pages/viewpage.action?pageId=57832637)    [18. URS WEB API - Finding/Acting on all interactions with specific properties](https://intranet.genesys.com/pages/viewpage.action?pageId=57832637)  Skip to end of metadata   * Created by [Vitaliy Teryoshin](https://intranet.genesys.com/display/~Vitaliy.Teryoshin@genesys.com), last modified on [Dec 14, 2016](https://intranet.genesys.com/pages/diffpagesbyversion.action?pageId=57832637&selectedPageVersions=5&selectedPageVersions=6)   Go to start of metadata  This method is similar to [**17. URS WEB API - Finding/Acting on interaction with specific properties**](https://intranet.genesys.com/pages/viewpage.action?pageId=57832629). It applies subroutine to every **not yet routed** call (so it is sort of injections applied to all calls in memories of all URSes from selfawareness cluster). Subroutine checks interaction properties (though it can also perform some correction actions). If subroutine exits without raising any error then its output will be included into this method result. The method returns collection (array) of results of successful runs of subroutine. This is one of few web methods from Interaction sub-module which doesn't require **connid**.  **urs/call/findall**?strategy=scriptname&tenant=tenantname&target=targetid&maxdelay=num&max=num&param1=value1&...  **urs/call/findall**?source=sourcecode&tenant=tenantname&target=targetid&maxdelay=num&max=num  Input:  - strategy: name of Script object (Subroutine or ECMAScript) to run.  - source: optional, source code (ECMAScript syntax only) of script. This is alternative to strategy way to specify script to run (see Note 2).  - tenant: name of tenant calls and strategy belongs to.  - target: optional, in format [name@statserver.type](mailto:name@statserver.type), if provided then only calls waiting this target will be tried.  - maxdelay: optional, default 0, provide maximum time in seconds to collect data from other URS nodes.  - max: optional, default 0, provide maximum (per URS node) size of returned array, 0 means no limit.  - param: any set of parameters that might be passed to subroutine. Subroutine's input parameters will be automatically initiated with parameters values taken from these request parameters on assumption parameters names match to name of subroutine input parameters.  Positive output:  JSON presentation of array of objects having as properties the subroutine output parameters (also see Note 2).  Errors:  HTTP 404 code with body **Object not found** if specified subroutine or target is not found  **Note 1** Extra parameter with name **call\_pos** will be added to parameters before trying every next call. Its value starts from 1 and is incremnented by 1 for every next call.  **Note 2** If source is used to specify script to run then following rules are applied: - script passes result of execution through **out** variable, any value different from **undefined** or **null** interpreted as success and used as returned value - no param is accessible from script - **call\_pos** value can be taken from stack with Pop() function.  **Samples**:  Script SID checks if call is located on specified RP and if yes then returns its attached data with key SID. Then to get array of "SID" attached data for all interactions on RP 2203:  <http://urshost:2082/urs/call/findall?strategy=SID&dn=2203> [{"SID":"029LK70N8SBO9F2B04000VTAESXX000F"},{"SID":"029LK70N8SBO9F2B04000VTAESXX001A"}]  To get array of ConnIDs of all calls having attached data CustomerSegment equal to Gold:  <http://urshost:2082/urs/call/findall?source=>**if(UData('CustomerSegment')=='Gold') out=ConnID();** ["02e302876e16a009","02e302876e16a01f","02e302876e16a024"]  To kill all interactions waiting in Virtual Queue MyVQ more than one day (and return array of connid of all deleted interactions:  [http://urshost:2082/urs/call/findall?target=MyVQ.Q&source=](http://urshost:2082/urs/call/findall?source=)**if(TimeRunning()>86400000){Timeout(1);out=ConnID();}** ["02e302876e16a009","02e302876e16a01f","02e302876e16a024"] |