

Technical Articles



Carlos Roggan

February 1, 2013 | 8 minute read

Speed up your Gateway services using Parallelization

7 4 5,740

Follow



Background

After creating a *SAP Gateway* service, you activate it on a Gateway Hub system and you specify the Backend System, from where the service will get its data. This Backend System is specified via a "System Alias". In the transaction **/IWFND/MAINT_SERVICE** (on the Hub system), it is possible to specify more than one "System Alias" for a single service (see screenshot in below section). This ability is called "Multiple Origin Composition" (abbr. "MOC", also known as "Multi Destination").

When calling such a service which is configured to use MOC, it will collect the data from all backend systems and aggregate them in one service call.

The description about MOC can be found in the SAP Documentation:

Configuration Guide -> OData Channel Configuration -> Settings ... on the Hub System -> Multiple Origin Composition

Direct Link:

http://help.sap.com/saphelp_gateway20sp05/helpdata/en/dd/f1ceb93a7d48fab4aa16efebc90e02/frameset.htm

As per default, the calls to the configured Backends are executed one following the other, combining the collected data afterwards.

Since **Netweaver Gateway 2.0 SP06** it is possible to define parallel execution for these Backend calls.

As you might assume: this will speed up the operation significantly!

Note:

Parallelization is used for QUERY operations, since this kind of operation might deal with huge amount of data.

Obviously, parallelization is only available for the MOC scenario.

Within the following step-by-step Guide, we will execute a simple QUERY operation two times:

First in the default mode and a second in parallel mode.

Afterwards, we compare the duration of both calls using the *SAP Netweaver Gateway Performance and Trace Tool*.

The prerequisites for following this guide are

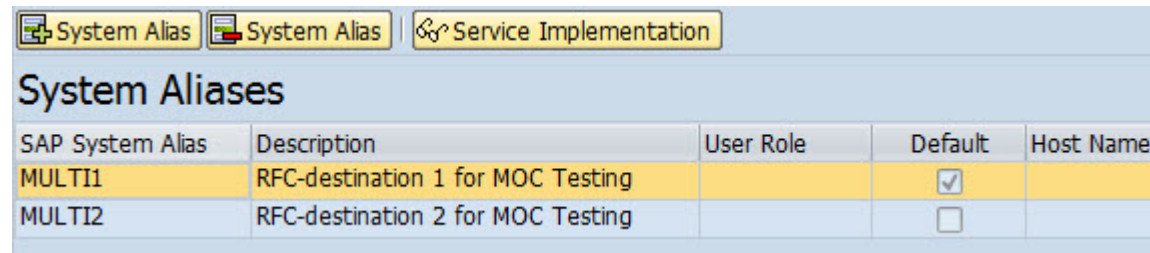
- SAP Netweaver Gateway 2.0 SP06,
- an activated Gateway service with at least an implemented QUERY operation,
- at least 2 Backend Systems (to be configured for MOC)

1) Prerequisite: Multiple Origin Composition

Since parallelization of service calls makes only sense in a scenario, where multiple Backends are used, we have to configure our sample service to use *Multiple Origin Composition*.

Go to transaction **/IWFND/MAINT_SERVICE**.

Choose your service and configure at least 2 System Aliases.



SAP System Alias	Description	User Role	Default	Host Name
MULTI1	RFC-destination 1 for MOC Testing		<input checked="" type="checkbox"/>	
MULTI2	RFC-destination 2 for MOC Testing		<input type="checkbox"/>	

2) Check if parallelization is disabled

First we want to execute the default mode, which is not parallelized.

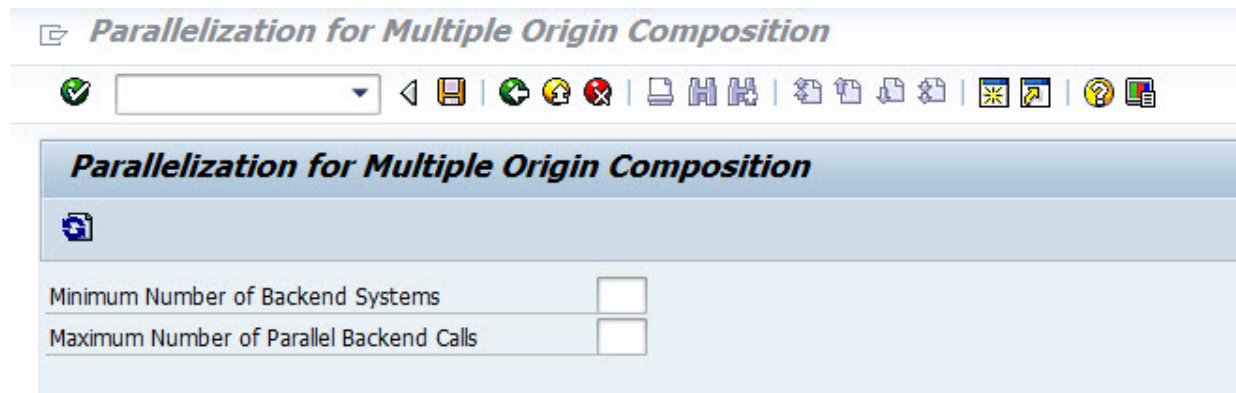
Go to transaction **SPRO**

Expand the IMG-tree as follows:

SAP Netweaver -> Gateway -> OData Channel->Administration->General Settings

Open the activity: *Define Parallelization for Multiple Origin Composition*

The following screen is displayed:



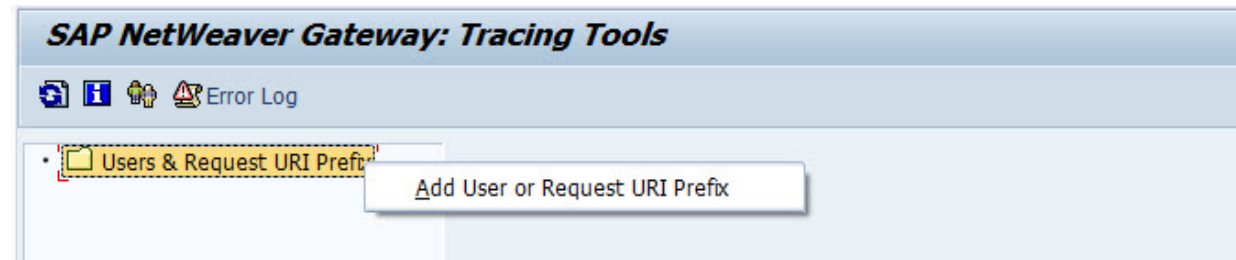
On the screen “Parallelization for Multiple Origin Composition”, the default settings are: the fields are empty, which means that the calls to the Backend are not executed in parallel mode (obviously, for parallel execution, at least two “Backend Systems” would be necessary).

For the moment, we don’t change the default settings and leave the transaction.

3) Enable tracing

In order to be able to afterwards compare the speed of the parallelized call versus the serialized mode, we need to first enable the tracing:

Start transaction /IWFND/TRACES



On the left pane, open context menu on “Users & Request URI Prefix”.

Add your user name.

On the right pane, the “Configuration” tab is displayed.

Within the drop-down field for “Performance Trace”, choose “Active”

Press “Save Configuration”.

From now on, all service requests for the specified user will be traced.

Open the tab “Performance Trace” in order to monitor the traces of the requests which will be sent in the following steps.

Find more information about the *SAP Netweaver Gateway Trace Tool* in the official SAP documentation:

Technical Operations Guide -> Supportability -> Performance Trace

Direct Link:

http://help.sap.com/saphelp_gateway20sp06/helpdata/en/9d/da3a2ceca344cf85568ae927e9858d/frameset.htm

4) Run the service in default mode

Now we run the service, i.e. we execute a QUERY operation.

You can use your favorite browser or the built-in *Gateway Client* for executing the service request.

In order to use the SAP Netweaver Gateway Client, go to transaction `/IWFND/GW_CLIENT`

Enter the URI for invoking a collection of your service.

In our sample, we execute

`/sap/opu/odata/IWFND/RMTSAMPLEFLIGHT;mo/FlightCollection`

Note: don't forget to specify `;mo` right after the service name, in order to indicate that multiple origin composition should be used.

Specify the HTTP Method GET, execute the request pressing the toolbar button "Execute" (or pressing F8)

The result is displayed in the "HTTP Response" pane.

In order to make sure that the response body contains data from all configured Backend Systems (as defined in step 1), we should have a look at the entries of the collection:

There should be an attribute with name `SAP__Origin`.

The value of this attribute is one of the System Aliases specified in step 1).

Search for a different entry, which contains the `SAP__Origin`-Attribute with the second System Alias.

In our example:


```
<d:SAP__Origin>MULTI1</d:SAP__Origin>
```

5) Enable Parallelization

Now we run the service again, but this time in parallel mode.

In order to enable the parallelization go to the parallelization screen, as described in step 2, and enter the number of configured Backend Systems in the first field.

In our example, we have configured the service RMTSAMPLEFLIGHT with 2 System Aliases (see step 1), therefore we enter 2, as depicted below:

Parallelization for Multiple Origin Composition	
	
Minimum Number of Backend Systems	<input type="text" value="2"/>
Maximum Number of Parallel Backend Calls	<input type="text"/>

Leave the second field empty and don't forget to press *Save*.

6) Run the service in parallelized mode

Go back to your browser or *Gateway Client* and execute the same service request again.

7) Analyze the performance trace

At this point in time, we have run our service two times and we have collected the performance data for both requests. Now it's time to have a look at the performance data.

Note: The following section is going to be a bit strenuous, you may skip it and jump directly to the summary at the end...;-)

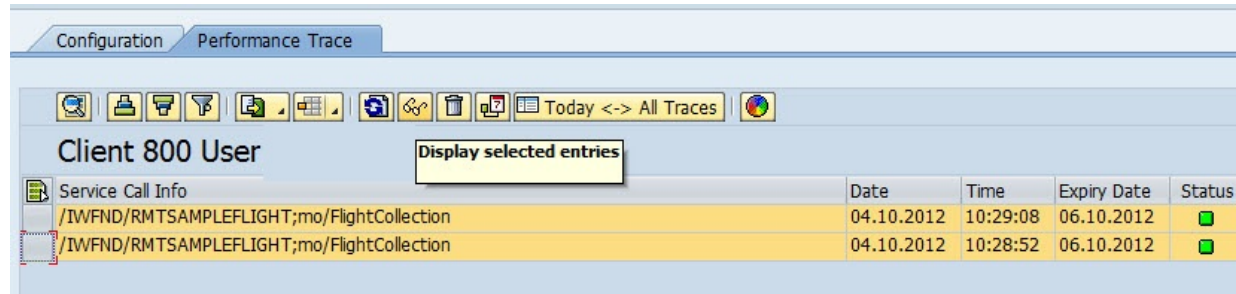
Start transaction `/IWFND/TRACES`.

Open the tab *Performance Trace*.

There should be (at least) 2 entries, for both calls which have been executed above.

Select both lines, then press the button "Display selected entries".

See below:



Service Call Info	Date	Time	Expiry Date	Status
/IWFND/RMTSAMPLEFLIGHT;mo/FlightCollection	04.10.2012	10:29:08	06.10.2012	■
/IWFND/RMTSAMPLEFLIGHT;mo/FlightCollection	04.10.2012	10:28:52	06.10.2012	■

The following screen allows comparing the performance trace of both calls:

Line No	Subcalls	Level	Location	Agent	Class	Method	Duration (ms)	Net Time (ms)
1	3	1	Gateway	OData Lib 1.0 Integ	/IWFND/CL_SODATA_ROOT_HANDLER	DISPATCH	995	93
2					>Response Size	1041187 Bytes		
3	2	2	Gateway	Metadata access	/IWFND/CL_MED_MDL_PROVIDER	GET_SERVICE_GROUP	303	101
4	1	3	Gateway	Metadata access	/IWFND/CL_MGW_MED_MDL_LOAD	Call Remote BEP - GITCLNT800_T	93	64
5		4	BEP	Metadata access	REMOTE_FUNCTION_MODULE	/IWBEP/FM_MGW_MED_LOAD	29	29
6	1	3	Gateway	Metadata access	/IWFND/CL_MGW_MED_MDL_LOAD	Call Remote BEP - GITCLNT800_T	109	17
7		4	BEP	Metadata access	REMOTE_FUNCTION_MODULE	/IWBEP/FM_MGW_MED_TXT_LOAD	92	92
8	4	2	Gateway	OData Lib 1.0 Integ	/IWFND/CL_SODATA_PROCESSOR	READ	590	152
9		3	Gateway	Data Provider: Gene	/IWFND/CL_MGW_PROV_DELEGATOR	GET_DATA_PROVIDER	7	7
10	1	3	Gateway	OData Channel Rem	/IWFND/CL_MGW_RUNT_RCLNT_PRXY	Call Remote BEP - GITCLNT800_T	34	16
11	1	4	BEP	OData Channel Rem	REMOTE_FUNCTION_MODULE	/IWBEP/FM_MGW_READ_ENTITYSET	18	17
12		5	BEP	Data Provider	/IWBEP/CL_MGW_RT_SFLIGHT	GET_ENTITYSET	1	1
13	1	3	Gateway	OData Channel Rem	/IWFND/CL_MGW_RUNT_RCLNT_PRXY	Call Remote BEP - GWQCLNT800_T	183	96
14	2	4	BEP	OData Channel Rem	REMOTE_FUNCTION_MODULE	/IWBEP/FM_MGW_READ_ENTITYSET	87	11
15		5	BEP	Data Provider	/IWBEP/CL_MGW_RT_SFLIGH	GET_ENTITYSET	14	14
16		5	BEP	Data Conversion	/IWBEP/CL_MGW_DATA_HELPER	CONVERT_ENTITYSET_OUTB	62	62
17		3	Gateway	OData Lib 1.0 Integ	/IWFND/CL_SODATA_MAPPER	GET_ENTITY_PROV_BY_FEED_DATA	214	214
18		2	Gateway	OData Lib 1.0 Integ	/IWFND/CL_SODATA_PROC_DISPTCHR	Lib Serialization - write_to	9	9
19	3	1	Gateway	OData Lib 1.0 Integ	/IWFND/CL_SODATA_ROOT_HANDLER	DISPATCH	698	36
20					>Response Size	1041187 Bytes		
21	2	2	Gateway	Metadata access	/IWFND/CL_MED_MDL_PROVIDER	GET_SERVICE_GROUP	217	27
22	1	3	Gateway	Metadata access	/IWFND/CL_MGW_MED_MDL_LOAD	Call Remote BEP - GITCLNT800_T	53	35
23		4	BEP	Metadata access	REMOTE_FUNCTION_MODULE	/IWBEP/FM_MGW_MED_LOAD	18	18
24	1	3	Gateway	Metadata access	/IWFND/CL_MGW_MED_MDL_LOAD	Call Remote BEP - GITCLNT800_T	137	19
25		4	BEP	Metadata access	REMOTE_FUNCTION_MODULE	/IWBEP/FM_MGW_MED_TXT_LOAD	118	118
26	3	2	Gateway	OData Lib 1.0 Integ	/IWFND/CL_SODATA_PROCESSOR	READ	436	124
27		3	Gateway	Data Provider: Gene	/IWFND/CL_MGW_PROV_DELEGATOR	GET_DATA_PROVIDER	2	2
28	2	3	Gateway	DATA_PROVIDER_N	/IWFND/CL_MGW_MDC_DATA	Parallelize Read EntitySet	134	9
29	1	4	Gateway	DATA_PROVIDER_N	/IWFND/CL_MGW_MDC_DISPATCHER	Call Remote BEP - GITCLNT800_T	66	30
30	1	5	BEP	DATA_PROVIDER_N	REMOTE_FUNCTION_MODULE	/IWBEP/FM_MGW_READ_ENTITYSET	36	35
31		6	BEP	Data Provider	/IWBEP/CL_MGW_RT_SFLIGHT	GET_ENTITYSET	1	1
32	1	4	Gateway	DATA_PROVIDER_N	/IWFND/CL_MGW_MDC_DISPATCHER	Call Remote BEP - GWQCLNT800_T	125	35
33	2	5	BEP	DATA_PROVIDER_N	REMOTE_FUNCTION_MODULE	/IWBEP/FM_MGW_READ_ENTITYSET	90	12
34		6	BEP	Data Provider	/IWBEP/CL_MGW_RT_SFLIGH	GET_ENTITYSET	14	14
35		6	BEP	Data Conversion	/IWBEP/CL_MGW_DATA_HELPER	CONVERT_ENTITYSET_OUTB	64	64
36		3	Gateway	OData Lib 1.0 Integ	/IWFND/CL_SODATA_MAPPER	GET_ENTITY_PROV_BY_FEED_DATA	176	176
37		2	Gateway	OData Lib 1.0 Integ	/IWFND/CL_SODATA_PROC_DISPTCHR	Lib Serialization - write_to	9	9

Let's have a closer look at the details of the screenshot:

Line 1 and 19 are highlighted as they represent our two service requests.

The following line respectively prints the "Response size".

No matter in which mode the service call has been executed (parallel or serialized), the amount of collected data has to be identical (same response size).

Accordingly, the "Response size" of our service calls, printed in line 2 and 20, is equal.

Starting from line 8 (resp. line 26 for the second call), the operation itself is being measured (method "READ").

7.1) Let's focus on the READ section of the serialized mode

This is the relevant excerpt from the above screenshot:

8	<u>4</u>	2	READ	590	152
9		3	GET_DATA_PROVIDER	7	7
10	<u>1</u>	3	Call Remote BEP - GITCLNT800_T	34	16
11	<u>1</u>	4	/IWBEP/FM_MGW_READ_ENTITYSET	18	17
12		5	GET_ENTITYSET	1	1
13	<u>1</u>	3	Call Remote BEP - GWQCLNT800_T	183	96
14	<u>2</u>	4	/IWBEP/FM_MGW_READ_ENTITYSET	87	11
15		5	GET_ENTITYSET	14	14
16		5	CONVERT_ENTITYSET_OUTB	62	62
17		3	GET_ENTITY_PROV_BY_FEED_DATA	214	214

We can see that the complete READ method took 590 milliseconds.

This is an aggregated number and it is calculated as follows:

- 1) The duration of the implementation of this “READ” method (line 8) is 152 ms (Net Time).
- 2) Within the “READ” method (line8), we’re in a lower level (compared to the starting point in line 1), which is level 2.

The “READ” method aggregates the durations of the subcalls of the lower level 3 (line 9 to 17).

Meaning that we have to sum up the duration of all level 3 methods:

line 9 (7 ms) + line 10 (34 ms) + line 13 (183 ms) + line 17 (214 ms)

Note: the method “Call Remote BEP” is invoked for every configured Backend system.

- 3) The total duration of the “READ” method:

$152 + 7 + 34 + 183 + 214 = 590$

Which is the expected result, just like it is printed in line 8

In words:

GetDataProvider	
+ Call Remote BEP	
+ Call Remote BEP	
+ GetEntityProv	
= READ	

7.2) Now let’s have a closer look at the READ section for parallelization

26	<u>3</u>	2	READ	436	124
27		3	GET_DATA_PROVIDER	2	2
28	<u>2</u>	3	Parallelize Read EntitySet	134	9
29	<u>1</u>	4	Call Remote BEP - GITCLNT800_T	66	30
30	<u>1</u>	5	/IWBEP/FM_MGW_READ_ENTITYSET	36	35
31		6	GET_ENTITYSET	1	1
32	<u>1</u>	4	Call Remote BEP - GWQCLNT800_T	125	35
33	<u>2</u>	5	/IWBEP/FM_MGW_READ_ENTITYSET	90	12
34		6	GET_ENTITYSET	14	14
35		6	CONVERT_ENTITYSET_OUTB	64	64
36		3	GET_ENTITY_PROV_BY_FEED_DATA	176	176

The complete READ method took 436 milliseconds.

This aggregated number is calculated as follows:

1) The duration of the implementation of this method is 124 ms (Net Time, line 26).

2) Again, we sum up the duration of all level 3 methods:

line 27 (2 ms) + line 28 (134 ms) + line 36 (176 ms)

$124 + 2 + 134 + 176 = 436$

3) But this time we have to dive one level deeper.

One of the level-3-methods was “Parallelize Read EntitySet”

This method is responsible for computing the parallelization of every call to the Backends.

Now we have to check its subcalls (level 4)

Within level 4, we have the same two calls to the Backends, one for each System Alias, with quite similar durations as above.

But this time, both calls are executed in parallel, therefore the duration of the complete READ is not calculated as sum of both calls !

Instead, the duration of the READ is equal to the duration of only **one** “Call Remote”,

to be more precise: the duration of the longest call, which in our example is line 32.

To be even more precise, we have to add the (short) time which is spent for computing the parallelization.

This can be seen in line 28, method “Parallelize Read EntitySet”, the “Net time” is: 9 ms

Finally, the duration of the “Parallelize Read EntitySet” is calculated as follows:

$$125 + 9 = 134$$

4) The total duration of the “READ” method:

$$124 + 2 + 125 + 9 + 176 = 436$$

This is the expected result, as it is printed in line 26

In words:

GetDataProvider	
+ Parallelize	
+ (maximum)Call Remote BEP	
+ GetEntityProv	
= READ	

7.3) Comparing both calls

Some of the numbers are expected to be the same, or at least similar, since it is the same code, the same call.

For example the “net time” of the “READ” method, the duration of the “Call RemoteBEP”. But since these methods are remote calls, the numbers cannot be 100% identical.

Most important, the duration of the parallelized request is expected to be faster.

Indeed: this is the case, as we can see comparing the lines 8 and 26 (Method READ, level 2)

8) Advanced Topics

8.1) Configuring parallelization

Within the spro activity described in step 2, there are 2 configuration options:

1. “Minimum Number of Backend Systems”

Leaving the input field empty is equivalent to entering 0, which is also the default number.

This means, no backend system call is executed in parallel mode, parallelization is disabled.

2. “Maximum number of parallel Backend Calls

This setting only makes sense if parallelization is enabled.

The maximum number of parallel backend calls always bases on current resources of the gateway hub system.

Additionally, this field can be used to limit the use of current system resources.

Zero (0) means it only depends on current system resources.

This is also the default.

8.2 Parallelization and Skiptoken

Parallelization is realized in any backend data provider than the odata consumer will only receive a result up to this backend including a skip token. The next call with this skip token or any other call with a skip token won't be parallelized

Alert Moderator

Assigned tags

SAP Gateway

gateway

netweaver gateway

odata

sap netweaver gateway

8.3 Misc notes

– Parallelization is not cross-client.

– Parallelization is applied to all services which are configured with multiple System Aliases.

– Parallelization is only applied if the services are invoked with ;mo in the URL.

Similar Blog Posts



[Understanding SAP Performance Statistics for SAP Gateway service](#)

By Carlos Roggan Jan 28, 2016

9) Summary

Within this blog we've seen that with just one little setting it is possible to significantly increase the speed of QUERYs in multiple origin scenarios.

[Top 10 achievements for NetWeaver Gateway in 2013](#)

By Former Member Jan 07, 2014

We've proven this expectation with a detailed analysis of the performance traces.

[Using Netweaver Gateway Development Tool for XCode and Gateway 2.0 SP03](#)

By Paul Aschmann Apr 23, 2012 Thanks for your interest!

Related Questions



[Backup existing service?](#)

By Former Member Oct 22, 2014

[What is the benifit of exposing ODATA services from SAP PO rather than the gateway service in ECC?](#)

By Aridip Ghosh Dec 07, 2018

[No Entity Sets in service](#)

By Former Member May 15, 2015

7 Comments

You must be [Logged on](#) to comment or reply to a post.



Kalyan Chakravarthy Kesana

February 2, 2013 at 9:08 am

Thanks for the blog. Clear explanation with proof 😊 . Much needed feature for multi-system landscape.

Like 0 | Share



Carlos Roggan | Blog Post Author

February 4, 2013 at 12:28 pm

One more note:

The "parallelization"-feature relies on having a service configured for "Multiple Origin Composition" (MOC).

This MOC is only supported in "Standard Mode", means it isn't available for "Generic Channel". As such, parallelization is only possible in "OData Channel".

Like 0 | Share



Syam Babu

February 7, 2013 at 4:57 pm

Nice Blog.

When we are planning to implement MOC with using different Back-end Systems these are all the steps are required other than these steps anything is required.

Is it possible to implement in SAP NW GW SP04 System.

Thanks,

Syam

Like 0 | Share



Carlos Roggan | Blog Post Author

February 8, 2013 at 9:33 am

Hi Syam, thanks;-)

Regarding your second question, MOC is already available in SP04, see the SAP SP4 help page here:

http://help.sap.com/saphelp_gateway20sp04/helpdata/en/dd/f1ceb93a7d48fab4aa16efebc90e02/frameset.htm

Regarding the steps required to do in order to have MOC running:

This feature is a Gateway-feature: instead of having a service which fetches data from one Backend System in e.g. Europe and another service fetching similar data from a second Backend System in e.g. Asia - with MOC this can be combined in one service.

Thus, from Gateway side, nothing else needs to be done.

However, there might be some administrative prerequisites:

Since the Service has to be configured with several "System Aliases", these additional "System Aliases" have to exist or have to be created before.

In order to create a System Alias pointing to a Backend System, there has to be an RFC destination created before.

This step might imply granting authorization, etc

On the configured Backend, the used User has to be created and granted permissions, etc

And of course, the pieces of software, which are called in order to fetch the data, have to be in place, etc

Kind regards,

Carlos

Like 0 | Share



Syam Babu

February 8, 2013 at 12:49 pm

Hi Carlos,

Thanks for the quick response.

See when i am developing data model using Generic Model at that time i will get one system alias for that model rite.

Suppose if i want to add the 2 system alias for the same data model ..is it possible ?

What is the possible way when we are achieving the Generic channel using MOC.

Thanks,

Syam

Like 0 | Share



Carlos Roggan | Blog Post Author

March 12, 2013 at 1:24 pm

Hi Syam,

sorry for the late reply, I didn't receive a notification mail;-)

Regarding the Generic channel: a Gateway model based on the "Generic Channel" cannot be configured with multiple system aliases. Only the "OData Channel" supports MOC.

Anyway, it is recommended to use the "Odata channel" which is the "Standard Mode" and not the "Generic Channel", which remains available for compatibility reasons, but isn't further enhanced.

Kind Regards,

Carlos

Like 0 | Share



Syam Babu

March 14, 2013 at 1:33 pm

Hi Carlos,

Thanks for the information.

Regards,

Syam

Like 0 | Share

Find us on

Privacy	Terms of Use
Legal Disclosure	Copyright

Trademark	Cookie Preferences
Newsletter	Support