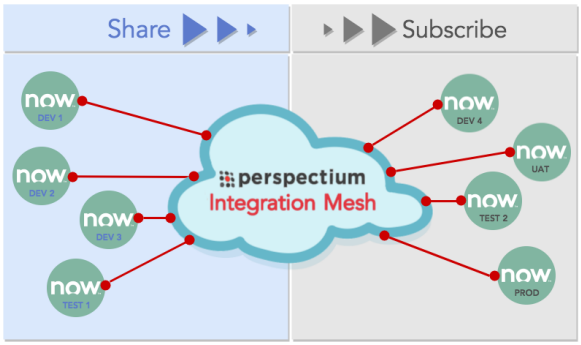
ServiceNow subscribe

The Perspectium application involves the transfer of data from a sharing instance flowing into a subscribing instance via the Integration Mesh. For this to happen, you need to set up both a shared queue on your sharing instance and a subscribing queue on your subscribing instance.

If you haven't set up these queues yet, please find more information and instructions to do so here.

Once you have set up these shared and subscribed queues, and have configured how your data will be shared from your sharing instance, you must also configure how your subscribing instance will consume the shared data. This is done in the **Subscribe** module through the use of scripts.

What's on this page?



Here are the ways in which you can use the **Subscribe** module to configure your ServiceNow subscribe instance.

Global subscribe is a **Subscribe** definition that when created, will allow all incoming messages to be subscribed. If there is an existing subscribe definition for a specific table, its definition will override the global definition.

About before and after subscribe scripts

Once you have configured a ServiceNow instance as a subscriber, you can use before/after subscribe scripts to customize how the data will be consumed and stored in your subscribing ServiceNow instance.

Before subscribe scripts

These scripts are created with server-side JavaScript and will execute right **before** a record insert or update, allowing the chance to modify the record before persisting. Within your before subscribe script, you have access to the following variables:

Variable	Description
current	Record that is being inserted or updated
repl_gr	Temporary inbound record. Will be mapped to **repl_gr** by default.
gr_before	Record before any update is made to it. If the record doesn't exist (i.e., for an insert), then this variable will be assigned to current.
qcurrent	Record within the **psp_in_message** table (pulled from the **Perspectium Mesh**). Use the message's **key** value to determine its source.
ignore	Can assign a value of **true** in order to stop the execution of the subscribe
qcurrentxml	Holds the xml object of the inbound record
xml_util	Holds an xml utility for working with **qcurrentxml**

Go to top of page

After subscribe scripts

These scripts are also created with server-side JavaScript and will execute right **after** a record from a sharing instance is synced on the subscribing instance. Within your after subscribe script, you have access to the following variables:

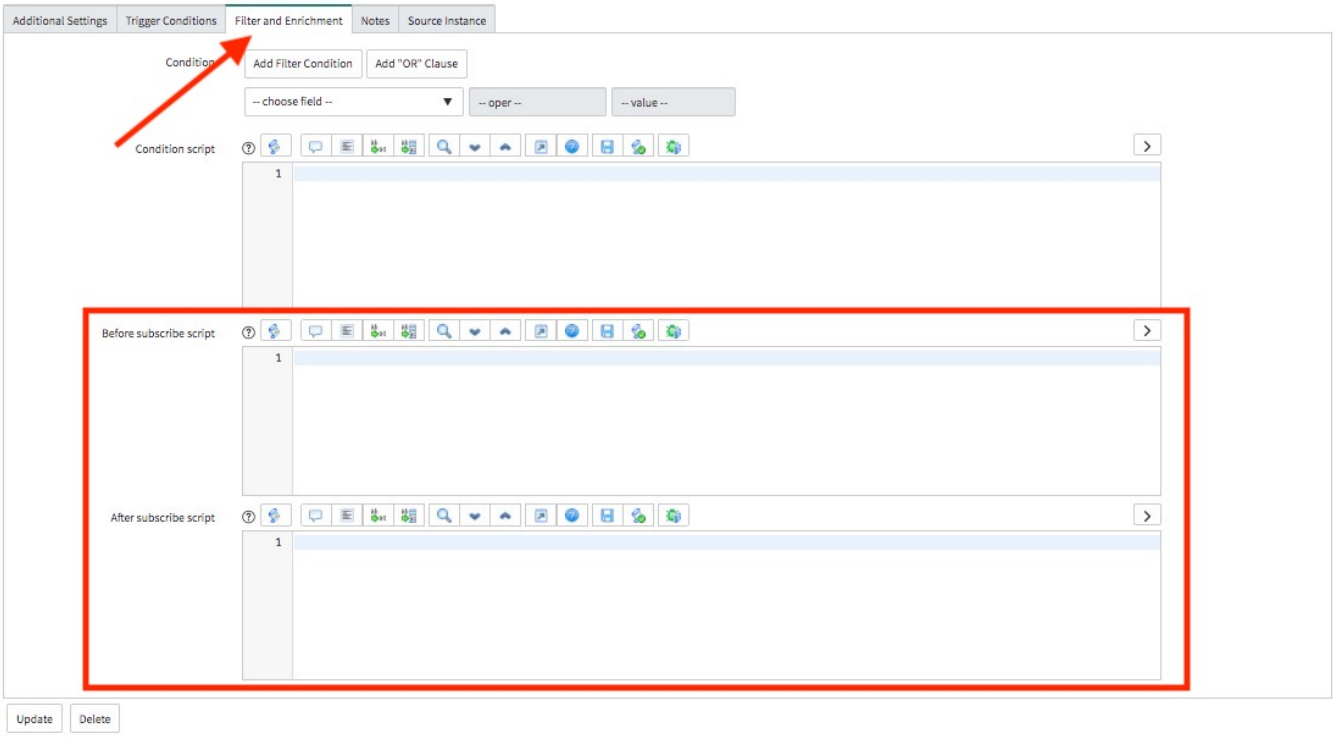
Variable	Description
current	Record that was inserted, updated or deleted, the destination record
qcurrent	Record within the **psp_in_message** table (pulled from the **Perspectium Mesh**). Use the message's **key** value to determine its source.
qcurrentxml	Holds the xml object of the inbound record
xml_util	Holds an xml utility for working with qcurrentxml
subscribe_gr	Holds the GlideRecord object corresponding to the subscribe record configuration itself (You can use **subscribe_gr.getTableName()** to access the name of the table you're trying to insert)

Go to top of page

Create before and after subscribe scripts

Here's how to create these scripts:

1. Log into your *sharing* ServiceNow instance that is sharing to the subscribing instance you want to configure, and go to **Perspectium** > **DataSync** > **Subscribe**.
2. In the resulting page, find and select the subscribe record that you want to configure.
3. In the resulting subscribe record, click the **Filter and Enrichment** tab. You should see the **Before subscribe script** and **After subscribe script** fields.
4. Use the available fields to enter your desired scripts. See below for some examples. Once you're done, click **Update**.



Go to top of page

Examples of before and after subscribe scripts

Below are some examples of scripts you can use to achieve specific results:

Populate the data using the incoming key

Perhaps you are receiving data from multiple sources and you want to denote the source in some manner. You can use the *qcurrent.key* value when updating your record.

The following script prefixes the short description with the key

```
current.short_description = qcurrent.key + " - " + current.short_description;
```

Map data within a record

For example, to set the *short_description* field to the incoming record's *number* field, you would do this:

```
current.short_description = repl_gr.number;
```

Dot-walking

You can use dot-walking to display values from a referenced field.

For example, if you were subscribing to a ticket record, in the *before subscribe script* text box, you can reference a field by using the following script:

```
current.name = repl_gr.assigned_to.name
```

This will display the *display_value* versus just the *sys_id*.

Use gr_before to fire an Event

For the case where you are updating a record that already exists in the system, “gr_before” is a GlideRecord object for this record before doing the update. This is useful for when you want to compare a record's values as they currently exist in the system with the values in the incoming record (repl_gr) and do any processing as a result.

For example, to fire the incident table's event when the assignment group is changed (the “incident.assigned.to.group” event as specified in the incident table's “incident events” business rule), you would do the following:

```
if (gr_before.assignment_group != repl_gr.assignment_group) {
  gs.eventQueue("incident.assigned.to.group", current, current.assignment_group, previous.assignment_group.
getDisplayValue());
}
```

Use ignore

For example, the following script from a ticket subscribe will ignore a specific ticket with the *number* value of **TKT0010001**.

```
if (current.number == "TKT0010001") {
    ignore = true;
}
```

You can also filter records from different instances by checking the *key* value. For example, the following script will ignore any records with the *key* valu e of “**dev14945**”.

```
if (qcurrent.key == "dev14945") {
    ignore = true;
}
```

Run script before a Delete

Support has been added so you can run script before doing a delete such as inserting a record into another table. In these cases you should use “repl_gr” to access the incoming record in case the incoming record doesn't exist in the instance.

For example, say you receive a delete but you want to insert a record into another table instead, and not do an actual delete:

```
var tgr = new GlideRecord("incident");
tgr.short_description = repl_gr.short_description + " new";
tgr.insert();
ignore = true;
```

Fire replication through subscribe

There are some cases where you want to replicate a message right after Subscribing it in. In most cases you can select “Run Business Rules” to treat this replication as a normal record transaction and our Dynamic Share's Business Rule should fall in line. If however you cannot use this on the Subscribe than you can execute the following command.

```
var pspRepl = new PerspectiumReplicator();
//sys_id of desired Dynamic Share
pspRepl.shareRecord(current, "incident", "bulk", "d24f961b4f043200daa12ed18110c72d");
ignore = false;
```

This is essentially the code that gets executed from the Business Rule for a Dynamic Share. With this you will put in the current record, the table name, the flag “bulk”, and the sys_id of the Dynamic Share you want this to route through.

It is also important to add in the (ignore = false;) statement to the Subscribe. There are cases where the execution of the Dynamic Share's “Before Share Script” can cause the Subscribe to skip the update. Adding in this command will ensure the record gets committed to the Subscribing instance.

Maintain a column's value

If you want to lock a column from being changed through the Subscribe you can put in the following line to set this column to it's “previous” value. Effectively setting it to itself so any change coming in won't take affect.

```
current.short_description = gr_before.short_description;
```

Access the XML object of the inbound record

For cases where the inbound record is different from the subscribing table (i.e. it has extra fields that the table on the subscribing instance doesn't have), you can now reference the XML object directly using the qcurrentxml variable.

The xml_util variable allows you to access values from the qcurrentxml object using getElementValueByTagName() function. This functions takes in two parameters, the XML object (qcurrentxml) and the name of the field in the XML.

For example, if you want to access the 'dv_priority' field's value in the XML and save it into the inbound record's short_description field:

```
if (qcurrentxml) {
    var elemValue = xml_util.getElementValueByTagName(qcurrentxml, 'dv_priority');
    current.short_description = elemValue;
}
```

We suggest that you check if qcurrentxml exists in case of any issues decrypting the value and getting the XML object.

Go to top of page

Sync all source fields

This feature is useful in cases when your sharing application has custom fields that do not typically exist in ServiceNow table schemas. Enabling the feature will ensure that any custom or outlying fields in your source instance are properly synced in your subscribing instance by inserting missing fields.

NOTES:

- If you use this option, the fields that exist in your source instance and not in your subscribing instance will be created in your subscribing instance *with the initial letter of the field label capitalized only*. For example, if you have a field in your sharing instance labeled custom FIELD, that field will be labeled in your target instance as **Custom field**.
- The **Sync all source fields** option will not work if multiple subscribes are created for the same table and one subscribe has the **Skip incoming records with field discrepancies** option checked.

Here's how to sync all source fields with the fields in your subscribing ServiceNow instance:

1. Log into your *subscribing* ServiceNow instance and go to **Perspectium** > **DataSync** > **Subscribe**.
2. In the resulting page, find and select the subscribe record that you want to sync all source fields for.
3. In the resulting subscribe record, click the **Source Instance** tab.
4. Check the **Sync all source fields** box. This will reveal some new fields.
5. Type the name of your sharing instance (e.g., **dev12345**) in the **Source Instance** field, and **username** and **password** of your sharing instance in the appropriate fields.
6. Click **Update**.

Go to top of page