SESSION VARIABLES & CLIENT DATA

Session variables and **client data** in ServiceNow are closely linked, but are not the exact same thing. Both are user-specific - similar to **user preferences** - but are less persistent than user preferences.

Session variables are (as the name implies) unique to a specific **user login session**[1]. This means that they're more *ephemeral* than **user preferences**, which persist between sessions after the user logs off and on again later (even from another location). **Client data**, meanwhile, is even less persistent. Client data only persists until you save a record or reload a page. In other words, session variables are linked with your session, on the server. Client data is linked with a specific instance of a page in the client (your web browser).

This non-persistence can be a good thing! You may for example, want the user to be able to alter something about their experience, but not have that change persist each time they log in. For example, perhaps you want to have the user select something on one page in your Service Portal which determines the behavior or contents of the next page. For this, you hypothetically *could* use a "user preference", but to do so would be overkill. It may even be *undesirable* to have that selection persist through multiple sessions.

I mentioned above that session variables and client data are closely linked. That's because every time you load a page or form, ServiceNow gets all of your server-side **session variables**, and sets a **client data element** from it. In other words, if you have a **session variable** called `last_selection` set to the value `12`, when you load a ServiceNow form, you can access that session variable as a **client data** element, just as you would any other.

There is one caveat, though: **it doesn't go both ways**. Setting a session variable on the server will make that variable (and its value) available on all future page loads[2] as client data, but setting a client data element will not cause that value to persist as a session variable on the server; even if the client data element you set, was originally derived from a server-side session variable. For example, if you were to set that `last_selection` session variable mentioned above and load a new page, you'd have a client data element with that same name and value. However, if you were to – in a **Client Script** – change the value of that client data from `12` to `7`, that change would persist *only until you reloaded the page*, at which point you would find that the value of that client data element reverted to the value of its corresponding session variable on the server (`12`).

Let's have a look at a real-world example of using session variables and client data. This example will be a little *contrived*, but just go with it.

Let's say you've been tasked with the following requirement:

1. When a user **saves an Incident** to the database, if the **Category** field changes, get the **category** of that Incident, and save it to a **session variable**.
2. The next time a user loads the Incident "new record" form, the **Category** field should be set to the last category they saved an Incident with.

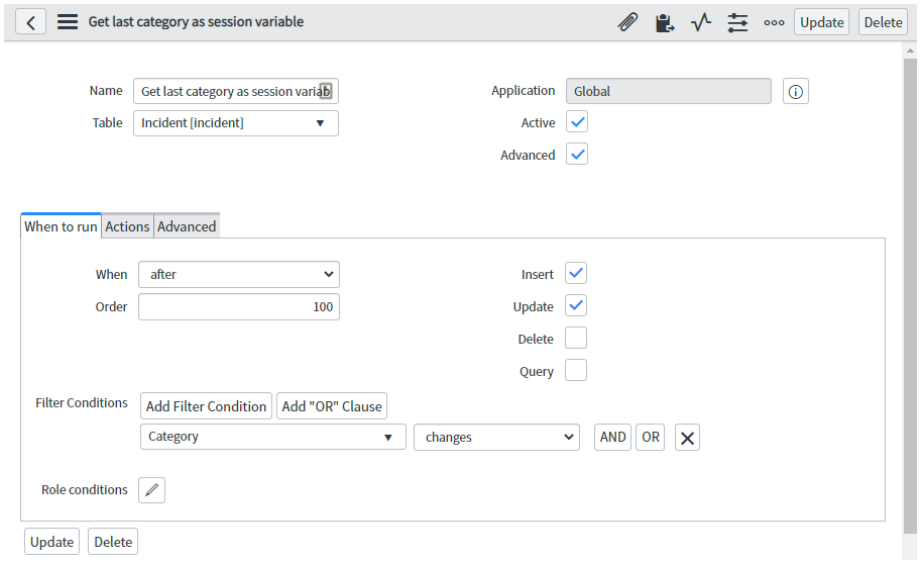
You might start by creating a Business Rule that runs on **Insert** or **Update**, whenever the **Category** field **Changes**. Since this BR doesn't need to modify anything about the record that triggers it, we'll make it an **After** Business Rule. Since we're going to need to write some code to set the session variable, we'll check the **Advanced** checkbox.

The initial configuration of our Business Rule might look something like this:

---

[1] *A "session" is essentially a specific instance of a user logging in. If you log in to your ServiceNow instance, that's one session. If you open a second tab, that second tab is still going to be logged into your first session. If, however, you load your instance in a new "incognito" window or in a separate browser, you will need to log in again. This constitutes a* ***new session****, which will have its own session ID. In this way, a single user can have multiple sessions, but no two users can share the same login session.*

[2] *Note that I said it'd be available on all* ***future*** *page loads. Setting a session variable on the server does not automatically update client data for any pages which were already loaded. That data only shows up on future page loads.*



Next, we need to write the script to set the session variable on the server in that BR, so we'll switch over to the **Advanced** tab and write a script that looks something like this:

```
(function executeRule(current, previous /*null when async*/) {

  var lastSelectedCategory = current.getValue('category');
  gs.getSession().putClientData(
      'last_selected_category',
      lastSelectedCategory
  );

})(current, previous);
```

Fig. 10.02

The next step is to write a **Client Script** to access the client data element whenever the Incident "new record" form is loaded, and use it to set the value of the **Category** field. You may notice in the above script that if the category field is blank, then the last_selected_category session variable will also be blank. Because of this, our Client Script will also have to check whether that variable is set to a *truthy* value (like a non-blank string), before updating the Category field on the form.

We'll create an **onLoad** Client Script on the Incident table, set the **Type** to **All** so it'll run on portal and workspace pages as well, and use something like the following code:

```
function onLoad() {
  //Hoist variable declarations
  var isvalidCategory, lastSelectedCategory;
  //Prevent script from running unless we're on the "new record" form.
  if (!g_form.isNewRecord()) {
      return;
  }
  lastSelectedCategory = g_user.getClientData(
      'last_selected_category'
  );
  //Halt if last selected category not set.
  if (!lastSelectedCategory) {
      return;
  }
  //Determine if last selected category is a valid.
  //Cast to boolean with !!.
```

```
  isvalidCategory = !!g_form.getOption(
      'category',
      lastSelectedCategory
  );
  if (isvalidCategory) {
      g_form.setValue('category', lastSelectedCategory);
  }
}
```

Fig. 10.03

In the above code, we're checking that we're on the new record form (line 5-7), getting the last selected category (ln 8-10), checking that lastSelectedCategory is set to a non-empty value (ln 12-14), checking that the value in lastSelectedCategory is a *valid choice* for the Category field (ln 17-20), and – if it is – setting the Category field value (ln 21-23). And that's all there is to it!