



## Application documentation Servicenow

### Introduction

The TransTech application is a demo designed to manage the everyday operations of a fictional transportation company. Its key functionalities include incident and task management, all served through a user-friendly service portal and forms tailored to the needs of transportation companies (including *Business rules*, *Client scripts*, *Ui policies* etc.).

### Stakeholders

The end users of the application are the **drivers**, whose usage of the application is limited to the service portal. They can submit incidents and create orders. **Dispatchers** and the **maintenance workers** are responsible for handling these incidents and tasks. **Admins** are responsible for maintaining the technical background of the application.

### Groups/ roles

- **x\_1121691\_transtec.user** role: to see and use the TransTech application, every employee of the company has this role
- **x\_1121691\_transtec.incidents\_user** role: to fill out and modify incidents
- **x\_1121691\_transtec.tasks\_user** role: to fill out and modify incidents
- **approver\_user** role: to approve
- **x\_1121691\_transtec.admin** role: to set and modify employee roles, delete task records

The following groups have been created: **Drivers**, **Dispatchers**, **Maintenance**, **Break-down service**, **Customer service**, **Admin**.

## Tables and custom fields

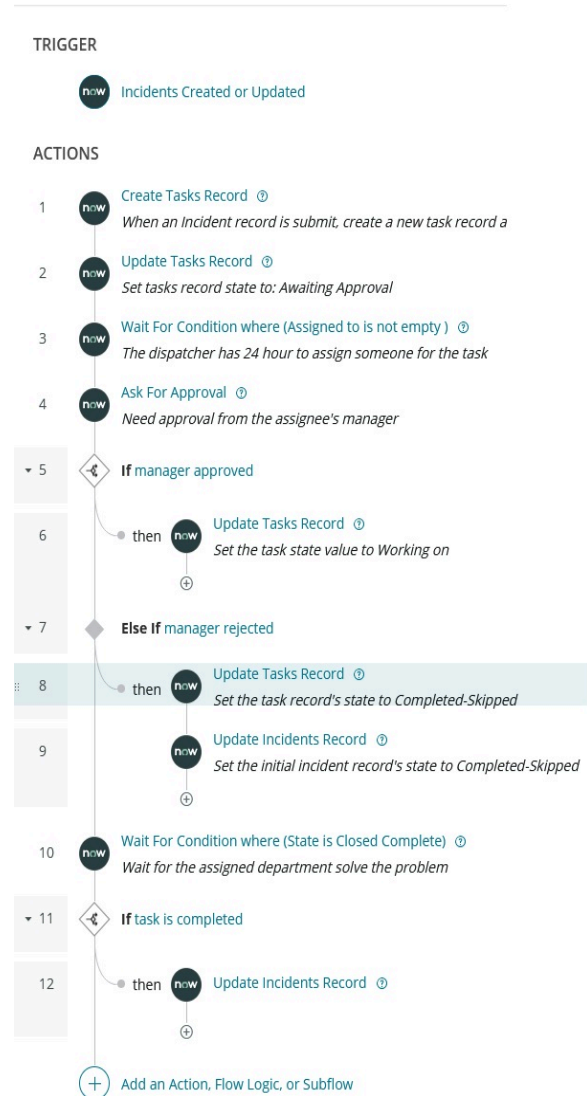
- **Drivers** (*x\_1121691\_transtec\_drivers*) - Extend User table (*sys\_user*)
  - Dispatcher (*u\_dispatcher*)  
Type: Reference
  - Driver licence(*driver\_licence*)  
Type: String
- **Vehicles** (*x\_1121691\_transtec\_vehicles*)
  - Registration number (*registration\_number*)  
Type: String
  - Brand (*brand*)  
Type: String
  - Under repair (*u\_under\_repair*)  
Type: True/False
  - Vehicle location (*vehicle\_location*)  
Type: String
- **Tasks** (*x\_1121691\_transtec\_tasks*) - Extend Task table (*task*)
  - Department (*department*)  
Type: Choice
  - Driver (*u\_driver*)  
Type: Reference
- **Incidents** (*x\_1121691\_transtec\_incidents*) - Extend Task table (*task*)
  - Affected vehicle (*u\_affected\_vehicle*)  
Type: Reference
  - Driver (*u\_driver*)  
Type: Reference
  - Category (*category*)  
Type: Choice
  - Dispatcher (*dispatcher*)  
Type: Reference

## Flows

- **TransTech Problem Managing**

The first flow purpose is to automate the process of incidents and tasks maintenance. The flow triggered by creating or updating a Transtech Incident record. In the first steps a new task is created, adding the trigger *Incident record* as a parent. Before the right department starts working on the task, it is necessary to approve it by the manager of assignee or assigned group.

In the next step the state of the *Task record* is updated depending on the approval decision. In the last steps a *Wait for condition* indicates that the task is completed and the triggered *Incident record State field* is also updated.

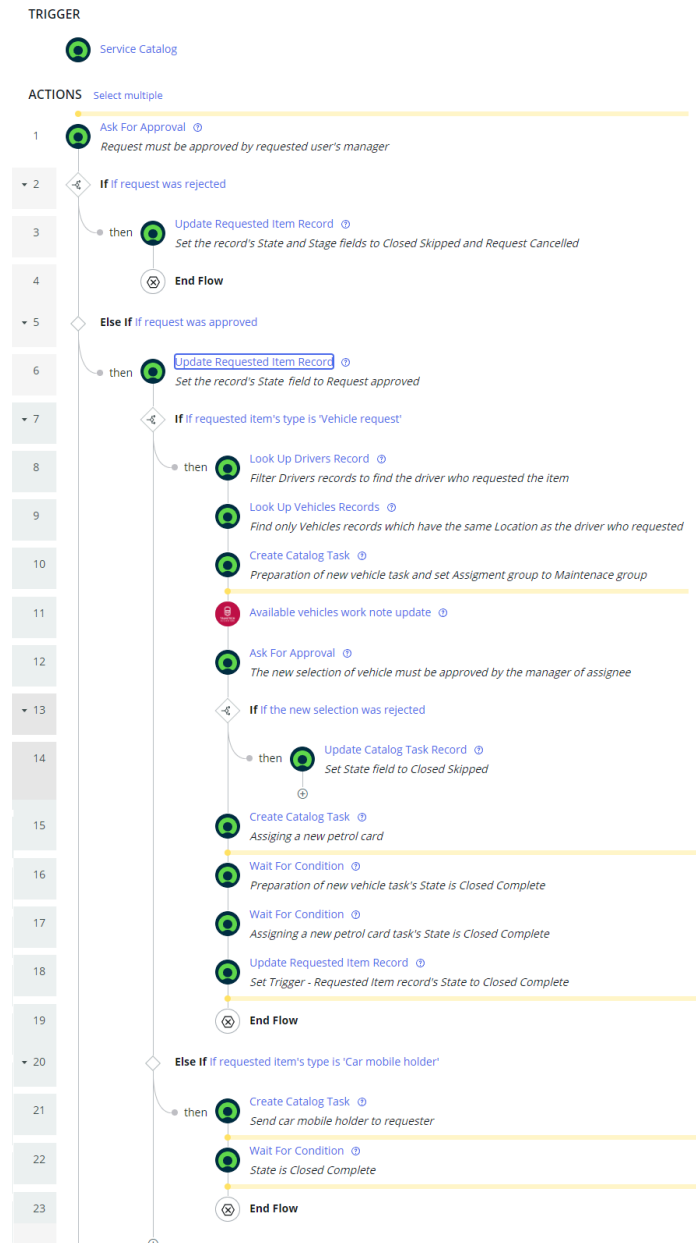


## • TransTech Request Managing

This flow automates the process of requests. A *Service Catalog* order triggers the flow (this flow is added to the TransTech application items only) and in the first step the requester's manager has to approve the request before any *Catalog task* is generated by the flow. If the request is approved the *State field* is updated. The company provides different kinds of items so firstly an If condition creates a branch for the 'Vehicle request' type of orders which means the driver needs a new vehicle.

Inside this branch two *Catalog tasks* are created and both assigned to the Maintenance department group. Two *Look Up records* and a custom action (*Available vehicles work note update*, see on the next page) related to the first task: '*Preparation of new vehicle*'. These are helping the decision making process by printing the list of vehicles which have the same location as the requester driver. When the new vehicle is selected it needs to be approved by the maintenance department manager.

The second automatically generated task is '*Assigning a new petrol card*'. In the last steps two *Wait for conditions* indicate that the tasks are completed and the *Trigger - Requested Item record's State field* updated to *Closed Complete*. The process is very similar for the other type of requested item, the complexity of steps could be different for each type.



Input Variables

Name	Value			
taskRecord	action ▶ Catalog Task Record X	🔍	📄	⊖
lookUpRecords	action ▶ Look Up Records X	🔍	📄	⊖

⊕ Create Variable

Script

```

1  (function execute(inputs, outputs) {
2    // ... code ...
3
4    var vehicleNumbers = [];
5    var vehicleRecords = inputs.lookUpRecords;
6
7
8    vehicleRecords.query();
9    while(vehicleRecords.next()){
10     vehicleNumbers.push(' ' + vehicleRecords.registration_number);
11   }
12
13   var note = 'Available vehicles: ' + vehicleNumbers;
14
15   var taskGR = inputs.taskRecord;
16   taskGR.work_notes = note;
17   taskGR.update();
18
19 })(inputs, outputs);
20

```

The *'Available vehicles work note update'* custom action takes a *Catalog Task Record* and a *Look Up Record* as inputs. The Look Up Record provides the records of vehicles which have the same location value as the driver has. The script queries through the records and pushes every *registration\_number* attribute to the *vehicleNumbers* array. In the last part the formatted string is saved into the *note* variable and the task record's *Work Note* field is updated with this formatted string.

## Service Portal

URL: <https://dev207263.service-now.com/ptt>

Widgets:


- **Create Incident:**  
user can create TransTech incident records
- **Order Something:**  
user can order items from Service Catalog
- **Knowledge Base**
- **My Requests:**  
widget shows requests related to the user
- **Incidents:**  
widget shows the recently created TransTech incident records
- **Tasks:**  
widget shows the recently created TransTech task records
- **My approvals:**  
widget shows the approvals related to the user
- **User connections:**  
widget shows with filter set to show only TransTech employees

Request	State	Updated
New vehicle request RE-0010092 System Administrator	Open	4d ago
New vehicle request RE-0010046 System Administrator	Open	30d ago
Apple iPhone 13 RE-0010005 System Administrator	Open	about a month ago


Name	Mobile phone	Email
Abel Tuter		abel.tuter@example.com
Alice Johnson		
Benjamin Schkade		benjamin.schkade@example.com
Bess Marso		bess.marso@example.com
Billie Cowley		billie.cowley@example.com

### New vehicle request

New vehicle request for drivers



Verification for new vehicle request

 Add attachments

## Access Control

- **x\_1121691\_transtec\_tasks (delete)**

The Access Control created to prevent deleting *Tasks* records except users who has TransTech admin role or assignee of the particular record. The default delete Access Control Rule of *Tasks* table has been disabled.

```
1 answer = false;
2 var userID = gs.getUserID();
3 // The assegee is the current user or has TransTec admin role can delete the record
4 if (current.assigned_to == userID || gs.hasRole('x_1121691_transtec.admin')) {
5 |   answer = true;
6 } else {
7 |   answer = false;
8 }
```

## UI Policy

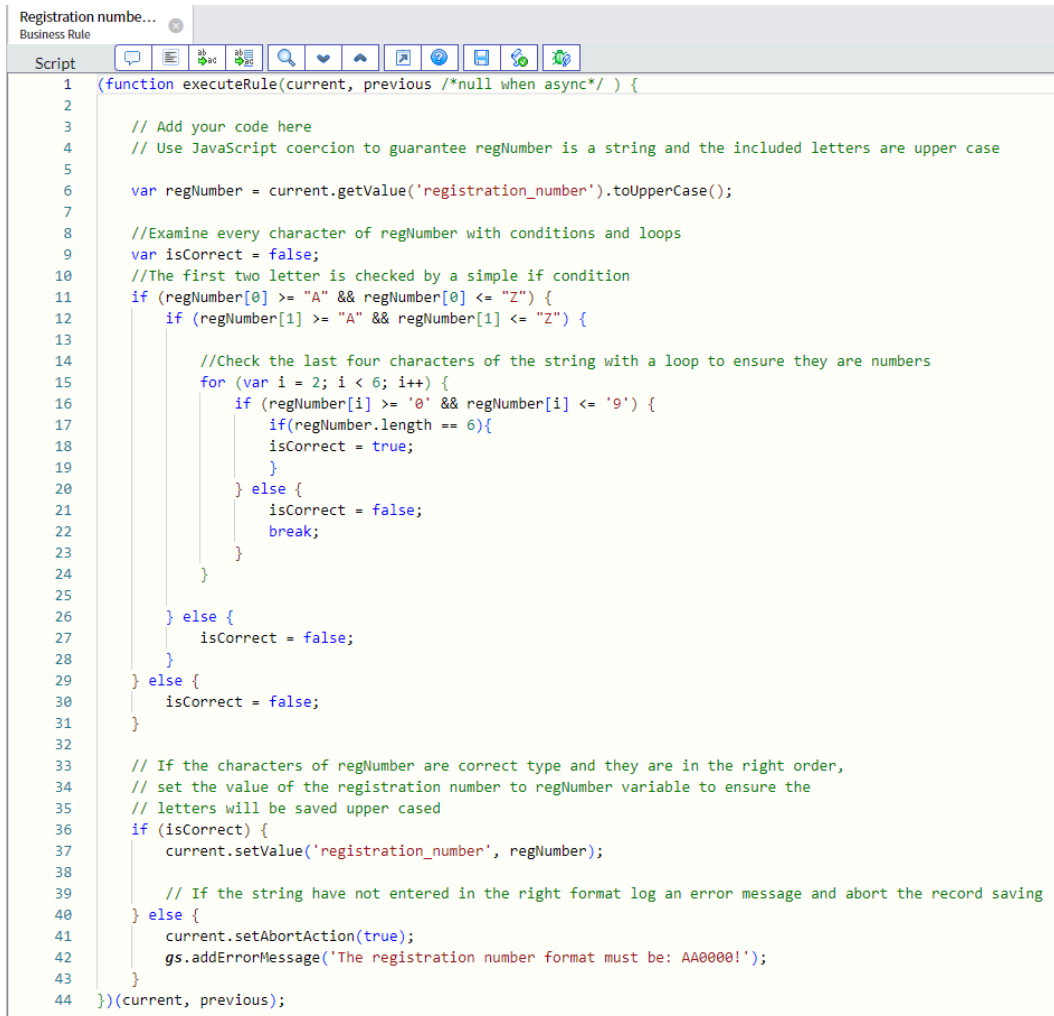
- **Show or hide inactive driver's return date field**

If a driver *Status field* is set to *inactive* an additional date type field appears on *Drivers* form. This new field shows the current date by default and it allows setting the return date of the driver. Also show a field message about the functionality of the new field.

## Business Rules

- **Registration number format**

This *Business rule* makes sure that the input format of the vehicle license number is correct (example: AA0000). The script handles the lowercase input and automatically returns the uppercased version of the letters that are included in the registration number. If the format is incorrect the script aborts submit of the record and logs an error message to notify the user.



```
1  <function executeRule(current, previous /*null when async*/ ) {
2
3      // Add your code here
4      // Use JavaScript coercion to guarantee regNumber is a string and the included letters are upper case
5
6      var regNumber = current.getValue('registration_number').toUpperCase();
7
8      //Examine every character of regNumber with conditions and loops
9      var isCorrect = false;
10     //The first two letter is checked by a simple if condition
11     if (regNumber[0] >= "A" && regNumber[0] <= "Z") {
12         if (regNumber[1] >= "A" && regNumber[1] <= "Z") {
13
14             //Check the last four characters of the string with a loop to ensure they are numbers
15             for (var i = 2; i < 6; i++) {
16                 if (regNumber[i] >= '0' && regNumber[i] <= '9') {
17                     if(regNumber.length == 6){
18                         isCorrect = true;
19                     }
20                 } else {
21                     isCorrect = false;
22                     break;
23                 }
24             }
25
26         } else {
27             isCorrect = false;
28         }
29     } else {
30         isCorrect = false;
31     }
32
33     // If the characters of regNumber are correct type and they are in the right order,
34     // set the value of the registration number to regNumber variable to ensure the
35     // letters will be saved upper cased
36     if (isCorrect) {
37         current.setValue('registration_number', regNumber);
38
39         // If the string have not entered in the right format log an error message and abort the record saving
40     } else {
41         current.setAbortAction(true);
42         gs.addErrorMessage('The registration number format must be: AA0000!');
43     }
44 })(current, previous);
```

## Client Scripts

- **Populate the Driver or Dispatcher fields (Incident Table)**

This *Onload type Client Script* checks if the logged in user has a driver or dispatcher position and fills out the particular field automatically. It uses the *Check User Position* Script Include as a call back function.



```

Set Driver Or Dispat...
Client Script
Script
1 function onLoad() {
2
3     //Create a new GlideAjax call on 'CheckUserPosition' Script Include
4     var incidentGA = new GlideAjax('CheckUserPosition');
5
6     // Adding the recall function name from the Script Include
7     incidentGA.addParam('sysparm_name', 'getUserID');
8
9     // Adding the current user's sys_id as a parameter to send it to the server side script
10    incidentGA.addParam('sysparm_userID', g_user.userName);
11    incidentGA.getXML(setFieldValue);
12
13    // Depending on the response from the Script Include set the field's values
14    function setFieldValue(response){
15        var answer = response.responseXML.documentElement.getAttribute('answer');
16        if (answer === 'dispatcher'){
17            g_form.setValue('dispatcher', g_user.userID);
18        }else if (answer === 'driver'){
19            g_form.setValue('driver', g_user.userID);
20        }else{
21            g_form.addInfoMessage('You are logged in as system administrator, please fill out the dispatcher and driver field too!');
22        }
23    }
24 }

```

## Script Includes

- **Check User Position**

This *Script Include* is called by *Populate the Driver or Dispatcher fields Client Script*. It queries through the *Dispatcher* and *Drivers* table using the current user *sys\_id* from the *Client Script*. Then it returns the user's position (dispatcher or driver) as a string.

```

CheckUserPosition
Script Include
Script
1 var CheckUserPosition = Class.create();
2 CheckUserPosition.prototype = Object.extend(Object.prototype, {
3
4     // creating the call back function for the client side script
5     getUserID: function(){
6
7         // declare a variable to contain the user's name from the Client Script
8         var userID = this.getParameter('sysparm_userID');
9
10        // list of the group names
11        var groups = ['Dispatchers', 'Drivers'];
12
13        // for loop to iterate the query with the number of groups
14        for(var i=0; i<=1; i++){
15
16            // query through the Group table to decide that the user member any of the groups
17            var groupGR = new GlideRecord('sys_user_grmember');
18            groupGR.addQuery('user.user_name', userID);
19            groupGR.addQuery('group.name', groups[i]);
20            groupGR.query();
21            if(groupGR.next()){
22
23                // if the user is a member of any of the groups return the group name as a string
24                if(i == 0){
25                    return 'dispatcher';
26                }else if (i == 1){
27                    return 'driver';
28                }
29            }
30        }
31    },
32
33    type: 'CheckUserPosition'
34 });

```

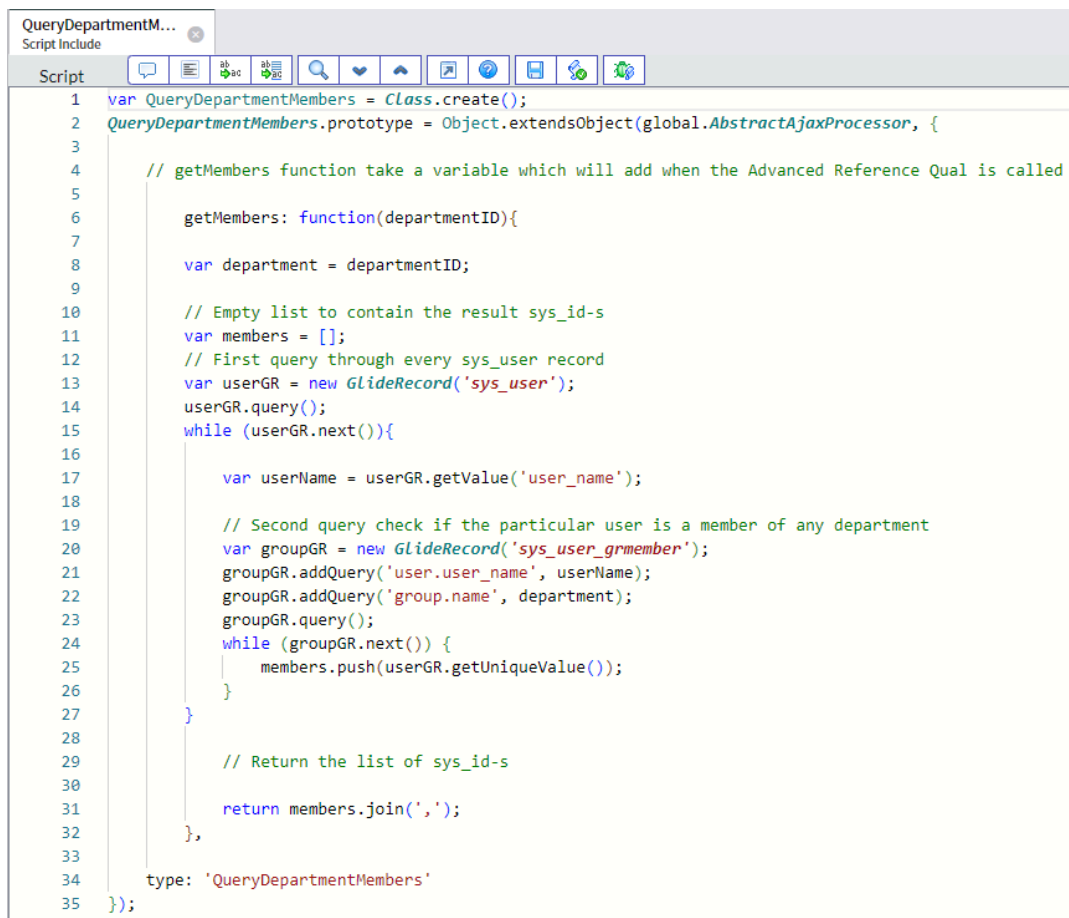
- **Query Department Members (Incident Table)**

This *Script Include* called by an override reference qualifier for *Assigned to field*. It returns a list of users who are members of the department group, selected dynamically by the user.

The reference qualifier code snippet:

```
javascript: "sys_id\\N" + new x_1121691_transtec.QueryDepartmentMembers().  
getMembers(current.getDisplayValue('department'));
```

Inside the script the function takes the name of the selected department as an input and queries through the *Users* table and checks if the user is a member of the selected group or not. Then it saves and returns the found records as a list.



```
QueryDepartmentM...  
Script Include  
Script  
1 var QueryDepartmentMembers = Class.create();  
2 QueryDepartmentMembers.prototype = Object.extendObject(global.AbstractAjaxProcessor, {  
3  
4     // getMembers function take a variable which will add when the Advanced Reference Qual is called  
5  
6     getMembers: function(departmentID){  
7  
8         var department = departmentID;  
9  
10        // Empty list to contain the result sys_id-s  
11        var members = [];  
12        // First query through every sys_user record  
13        var userGR = new GlideRecord('sys_user');  
14        userGR.query();  
15        while (userGR.next()){  
16  
17            var userName = userGR.getValue('user_name');  
18  
19            // Second query check if the particular user is a member of any department  
20            var groupGR = new GlideRecord('sys_user_grmember');  
21            groupGR.addQuery('user.user_name', userName);  
22            groupGR.addQuery('group.name', department);  
23            groupGR.query();  
24            while (groupGR.next()) {  
25                members.push(userGR.getUniqueValue());  
26            }  
27        }  
28  
29        // Return the list of sys_id-s  
30  
31        return members.join(',');  
32    },  
33    type: 'QueryDepartmentMembers'  
34 }  
35 };
```