Technical Specifications for

FDR\_NDT\_Reboot\_ONT

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comment** |
| 0.1 | 19/06/2019 |  | Draft |

Business Process Owner

|  |  |  |
| --- | --- | --- |
| **Business owner** | **Onsite contact** | **Offshore contact (email/phone)** |
| Anurag Pant | Patki Raghavendra  [raghavendra.patki@verizon.com](mailto:raghavendra.patki@verizon.com) | Appaswamy, Venkatesh  [venkatesh.appasamy@verizon.com](mailto:venkatesh.appasamy@verizon.com)  Contact : 8139878055 |

**Table of Contents**

[1. Bonita Process 3](#_Toc12020027)

[1.1 Scope 3](#_Toc12020028)

[1.2 Process Implementation 4](#_Toc12020029)

[1.3 Process Variables 4](#_Toc12020030)

[1.4 Process Implementation 4](#_Toc12020031)

[1.4.1 Step 1: Start and Skip Reboot X-OR Gateway 4](#_Toc12020032)

[1.4.2 Step 2: Notify Reboot ONT 4](#_Toc12020033)

[1.4.3 Step 3: Reboot ONT 4](#_Toc12020034)

[1.4.4 Step 4: Break DHCP and Reboot 4](#_Toc12020035)

[1.4.5 Step 5: Gateway8 4](#_Toc12020036)

[1.4.6 Step 6: bdt Status 4](#_Toc12020037)

[1.4.7 Step 7: Dialtone Test 4](#_Toc12020038)

[1.4.8 Step 8: Registration Check 4](#_Toc12020039)

[1.4.9 Step 9: Gateway9 4](#_Toc12020040)

[1.4.10 Step 10: Validate Reboot Count 4](#_Toc12020041)

[1.4.11 Step 11: Check if issue resolved 4](#_Toc12020042)

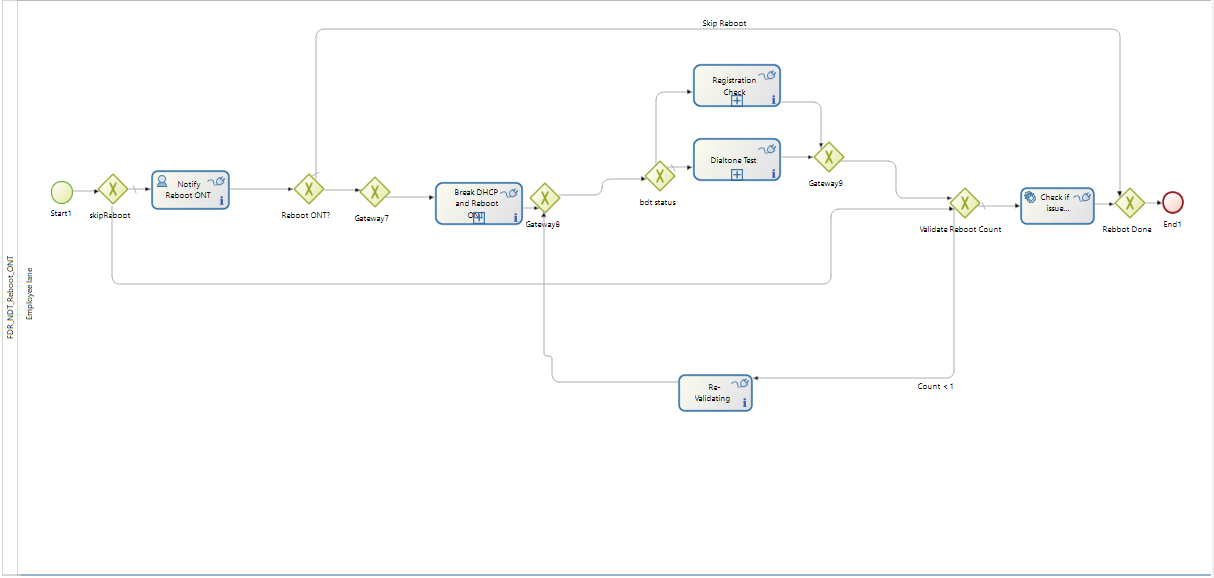
[1.4.12 Step 12: Re-Validating 4](#_Toc12020043)

[1.4.13 Step 13: “Reboot ONT?” to “Reboot Done” 4](#_Toc12020044)

[1.4.14 Step 14: “Reboot Done” to “End Node” 4](#_Toc12020045)

1.4.15 Step 15: All Groovy scripts used in the process

1. Bonita Process - FDR\_NDT\_Reboot\_ONT



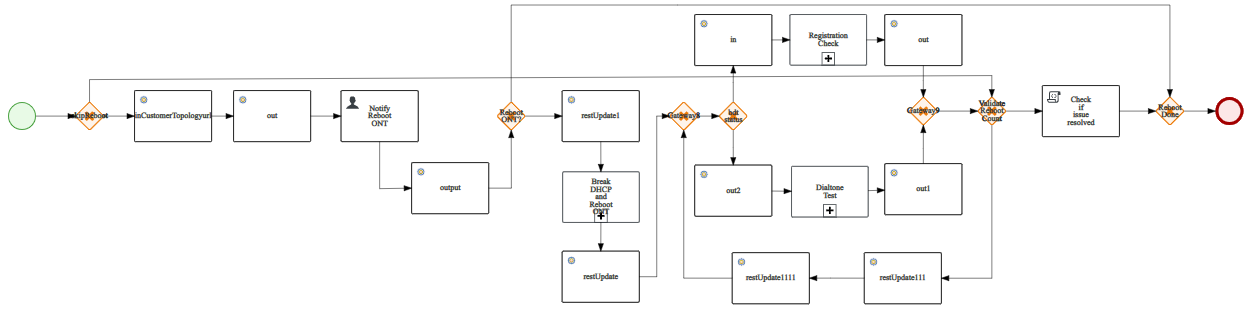
* 1. Scope

This workflow does reboot on ONT device. A command will be sent to device in the form of request and device will send the response back by returning trouble code. Depending upon trouble code we can determine whether ONT is rebooted or not by revalidating.

Question: Why do we require human task to notify reboot (Does request have to be approved?)

Answer: Yes, customer has to approve it or agent has to perform that actions, sometimes it may be required to reboot or we can skip the reboot

* 1. Process Implementation



* 1. Process Variables

|  |  |
| --- | --- |
| Variable Name | Type |
| actionItems | (Java\_object) |
| actionId | (Text) |
| breakDialToneStatus | (Text) |
| Ckttype | (Text) |
| Customertopology | (Text) |
| Command | (Text) |
| dataPortStatus | (Boolean) |
| Description | (Text) |
| dialToneIssueResolved | (Boolean) |
| Eltype | (Text) |
| finalActionResult | (Text) |
| hasGoodDialTone | (Boolean) |
| hasGoodPing | (Boolean) |
| hasSipPortIssues | (Boolean) |
| inLineId | (Text) |
| initialBreakDialToneStatus | (Text) |
| isDataPortUp | (Boolean) |
| isManualRebootDone | (Boolean) |
| isParallel | (Boolean) |
| isPortUp | (Boolean) |
| isStatic | (Boolean) |
| issueResolved | (Boolean) |
| l3ICMPPingStatus | (Text) |
| manualFlag | (Integer) |
| manualTaskResult | (Text) |
| Microserviceurl | (Text) |
| newRegDate | (Java\_object) |
| ontType | (Text) |
| ontTypePub | (Text) |
| ontVendorType | (Text) |
| pingBHRMap | (Text) |
| pingStatus | (Text) |
| portType | (Text) |
| registrationDate | (Java\_object) |
| registrationStatus | (Text) |
| responseString | (Text) |
| Resultsetpublic | (Java\_object) |
| sipAlarmStatus | (Text) |
| sipAlarmTroubleCode | (Text) |
| skipVoipPortCheck | (Boolean) |
| Title | (Text) |
| Transactionid | (Text) |
| Troubletype | (Text) |
| validateCounter | (Integer) |
| voiceServiceStatus | (Boolean) |

* 1. Process Implementation
     1. Step 1: Start and Skip Reboot X-OR Gateway

|  |  |
| --- | --- |
| **Type of Task** | Manual Start Event and X-OR Event Gateway |
| **Actor** | NA |
| **Variables Used** | |  |  | | --- | --- | | Variables | Data Type | |  |  | |
| **Condition** | NA |
| **Connectors In** | NA |
| **Connectors Out** | NA |
| **Description** | Start node will call skipReboot (XOR Gateway). It will have two decision paths in which one would be the default gateway and other will have a condition on the connector from skipReboot to Validate Reboot Count. |
| **Diagrams** |  |

* + 1. Step 2: Notify Reboot ONT

|  |  |
| --- | --- |
| **Type of Task** | User Task |
| **Actor** | Employee actor—This is an example of actor that is mapped to any ACME users. |
| **Variables Used** | |  |  |  | | --- | --- | --- | | **Variables** | **Data Type** | **In / Out** | | ManualRebootDone | Boolean |  | | FinalAction | TEXT - java.lang.String | in | | Actions | COMPLEX - java.util.Map | in | | manualTaskResults | TEXT - java.lang.String | in | | manualFlag | Integer |  | | pingBHRMap | TEXT - java.lang.String |  | | hasGoodPing | Boolean |  | |
| **Condition** |  |
| **Connectors In** | |  |  |  |  | | --- | --- | --- | --- | | **Id** | **Type** | **Event** | **Ignore Errors** | | inCustomerTopologyurl | RESTConnector | ON\_ENTER | False | | command | scripting-groovy-script | ON\_ENTER | False | | Out | RESTConnector | ON\_ENTER | false | |
| **Connectors Out** | |  |  |  |  | | --- | --- | --- | --- | | **Id** | **Type** | **Event** | **Ignore Errors** | | output | RESTConnector | ON\_FINISH | False | | getBHRMap111 | scripting-groovy-script | ON\_FINISH | False | |
| **Description** | * Default condition path will reach to Human task node (Notify Reboot ONT). It will have an Actor assigned to it when you select general properties of Notify Reboot ONT node as shown below. * Also in the Execution properties pane we can see Connectors in and Connectors out on the node as shown below. * Also there are operations that needs to be done on this node and assign values to the variables accordingly as shown below * After Notify Reboot ONT, it reaches to Reboot ONT? Gateway node. |
| **Diagrams** | Connectors in and Out: |

* + 1. Step 3: Reboot ONT

|  |  |
| --- | --- |
| **Type of Task** | X-OR Gateway Event |
| **Actor** |  |
| **Variables Used** | |  |  |  | | --- | --- | --- | | **Variables** | **Data Type** | **In / Out** | | finalActionResult | Text |  | |
| **Condition** | Default |
| return finalActionResult.toUpperCase().contains("REBOOT ONT") |
| **Connectors In** |  |
| **Connectors Out** |  |
| **Description** | * **Gateway 7** is not required * \*\*As shown below, we don’t need **Gateway7 (Immediate XOR Gateway** without any conditional paths wouldn’t serve any purpose). So we will eliminate the Gateway7 from flow and connect directly from RebootONT? To Break DHCP and Reboot ONT. |
| **Diagrams** |  |

* + 1. Step 4: Break DHCP and Reboot

|  |  |
| --- | --- |
| **Type of Task** | Reusable sub-process |
| **Actor** |  |
| **Variables Used** | |  |  |  | | --- | --- | --- | | **Variables** | **Data Type** | **In / Out** | | Trouble type | String | in | | svctype | String | in | | Action id | Integer | in | | rebootOntResult | Java\_object | out | | Validate Counter | Integer |  | |
| **Condition** |  |
| **Connectors In** | |  |  |  |  | | --- | --- | --- | --- | | **Id** | **Type** | **Event** | **Ignore Errors** | | restUpdate1 | RESTConnector | ON\_ENTER | False | |
| **Connectors Out** | |  |  |  |  | | --- | --- | --- | --- | | **Id** | **Type** | **Event** | **Ignore Errors** | | wait | scripting-groovy-script | ON\_FINISH | False | | RestUpdate | RESTConnector | ON\_FINISH | False | |
| **Description** | * Break DHCP and Reboot ONT is a reusable sub-process and it calls Ex\_DelphiPool workflow. Also it has Connectors in, Connectors out and Operations on the node. * Data to send and receive from Sub-process has to be mapped as shown below: * Operations on the node are to be assigned after the completion of the activity as shown below * After that Break DHCP and Reboot ONT. It reaches Gateway 8. * Gateway 8 acts here as converging gateway. Finally, it will be reaching to bdt status XOR gateway. |
| **Diagrams** | **Connectors:** |

* + 1. Step 5: Gateway8

|  |  |
| --- | --- |
| **Type of Task** | X-OR Gateway |
| **Actor** |  |
| **Variables Used** | |  |  |  | | --- | --- | --- | | **Variables** | **Data Type** | **In / Out** | |  |  |  | |
| **Condition** | It has no conditions specified on connector and there is only one way path to reach next node. In this case Gateway8 acts as converging gateway. |
| **Connectors In** |  |
| **Connectors Out** |  |
| **Description** | * Gateway 8 has no conditioned paths. |
| **Diagrams** |  |

* + 1. Step 6: bdt Status

|  |  |
| --- | --- |
| **Type of Task** | X-OR Gateway |
| **Actor** |  |
| **Variables Used** | |  |  |  | | --- | --- | --- | | **Variables** | **Data Type** | **In / Out** | | initialBreakDialToneStatus | Text |  | |
| **Condition** | Default |
| return "NOTSUPPORTED".equalsIgnoreCase(initialBreakDialToneStatus) |
| **Connectors In** |  |
| **Connectors Out** |  |
| **Description** | * From the bdt status there are two decision paths. Default one will be to Dialtone Test Sub process and conditioned path would be to Registration check. |
| **Diagrams** |  |

* + 1. Step 7: Dialtone Test

|  |  |
| --- | --- |
| **Type of Task** | Reusable sub-process |
| **Actor** |  |
| **Variables Used** | |  |  |  | | --- | --- | --- | | **Variables** | **Data Type** | **In / Out** | | troubletype | String | in | | microserviceurl | String | in | | actionId | Integer | in | | transactionid | Integer | in | | eltype | String | in | | inLineId | String | in | | responseString | String | out | | hasGoodDialTone | Boolean |  | | breakDialToneStatus | String |  | | pingStatus | String |  | | dialToneIssueResolved | Boolean |  | | initialBreakDialToneStatus | String |  | |
| **Condition** |  |
| **Connectors In** | |  |  |  |  | | --- | --- | --- | --- | | **Id** | **Type** | **Event** | **Ignore Errors** | | out2 | RESTConnector | ON\_ENTER | ` | |
| **Connectors Out** | |  |  |  |  | | --- | --- | --- | --- | | **Id** | **Type** | **Event** | **Ignore Errors** | | getBreakDialToneStatus | scripting-groovy-script | ON\_FINISH | False | | out1 | RESTConnector | ON\_FINISH | False | |
| **Description** | * **Dialtone Test** is a reusable sub-process and it calls Ex\_DelphiPool workflow. Also it has Connectors in, Connectors out and Operations on the node. * Data to send and receive from Sub-process has to be mapped as shown below: * Operations on the node are to be assigned after the completion of the activity as shown below * After **Dialtone Test** is executed it will reach Gateway 9. |
| **Diagrams** | **Connectors:** |

* + 1. Step 8: Registration Check

|  |  |
| --- | --- |
| **Type of Task** | Reusable sub-process |
| **Actor** |  |
| **Variables Used** | |  |  |  | | --- | --- | --- | | **Variables** | **Data Type** | **In / Out** | | troubletype | String | in | | microserviceurl | String | in | | actionId | Integer | in | | transactionid | Integer | in | | eltype | String | in | | svctype | String | in | | inLineId | String | in | | responseString | String | out | | result | Java Object | out | | initresponse | String | Out | | registrationStatus | String |  | | dialToneIssueResolved | Boolean |  | |
| **Condition** |  |
| **Connectors In** | |  |  |  |  | | --- | --- | --- | --- | | **Id** | **Type** | **Event** | **Ignore Errors** | | in | RESTConnector | ON\_ENTER | False | |
| **Connectors Out** | |  |  |  |  | | --- | --- | --- | --- | | **Id** | **Type** | **Event** | **Ignore Errors** | | getRegDate | scripting-groovy-script | ON\_FINISH | False | | getRegistrationStatus | scripting-groovy-script | ON\_FINISH | False | | out | RESTConnector | ON\_FINISH | False | |
| **Description** | * **Registration Check** is a reusable sub-process and it calls Ex\_DelphiPool workflow. Also it has Connectors in, Connectors out and no Operations on the node. * Data to send and receive from Sub-process has to be mapped as shown below: * After **Registration Check** is executed it will reach Gateway 9. |
| **Diagrams** | **Connectors:** |

* + 1. Step 9: Gateway9

|  |  |
| --- | --- |
| **Type of Task** | X-OR Gateway |
| **Actor** |  |
| **Variables Used** | |  |  |  | | --- | --- | --- | | **Variables** | **Data Type** | **In / Out** | |  |  |  | |
| **Condition** | It has no conditions specified on connector and there is only one way path to reach next node. In this case Gateway8 acts as converging gateway. It will reach Validate Reboot Count XOR gateway. |
| **Connectors In** |  |
| **Connectors Out** |  |
| **Description** | * Gateway 9 has no conditioned paths. |
| **Diagrams** |  |

* + 1. Step 10: Validate Reboot Count

|  |  |
| --- | --- |
| **Type of Task** | X-OR Gateway |
| **Actor** |  |
| **Variables Used** | |  |  |  | | --- | --- | --- | | **Variables** | **Data Type** | **In / Out** | | validateCounter | Text |  | | dialToneIssueResolved | Boolean |  | | finalActionResult | Text |  | | registrationStatus | Text |  | |
| **Condition** | Default |
| **try**{  **return** validateCounter < 1 && !dialToneIssueResolved && finalActionResult.toUpperCase().contains("REBOOT ONT") && !("SUCCESS".equalsIgnoreCase(registrationStatus))  }  **catch**(Exception e){  **return** **false**;  } |
| **Connectors In** |  |
| **Connectors Out** |  |
| **Description** | * From the Validate Reboot Count there are two decision paths. Default one will be to **Check if issue resolved** and conditioned path would be to **Re-Validating**. |
| **Diagrams** |  |

* + 1. Step 11: Check if issue resolved

|  |  |
| --- | --- |
| **Type of Task** | Script Task |
| **Actor** |  |
| **Variables Used** | |  |  |  | | --- | --- | --- | | **Variables** | **Data Type** | **In / Out** | | dialToneIssueResolved | String |  | | registrationStatus | String |  | |
| **Condition** |  |
| **Connectors In** | |  |  |  |  | | --- | --- | --- | --- | | **Id** | **Type** | **Event** | **Ignore Errors** | | checkPingAndPortStatus | scripting-groovy-script | ON\_ENTER | False | |
| **Connectors Out** | |  |  |  |  | | --- | --- | --- | --- | | **Id** | **Type** | **Event** | **Ignore Errors** | |  |  |  |  | |
| **Description** | * **Check if issue resolved** is a Script Task. It has only one Connector in. * After **Check if issue resolved** is executed it will reach **Reboot Done**. |
| **Diagrams** | **Connectors:** |

* + 1. Step 12: Re-Validating

|  |  |
| --- | --- |
| **Type of Task** | Service Task |
| **Actor** |  |
| **Variables Used** | |  |  |  | | --- | --- | --- | | **Variables** | **Data Type** | **In / Out** | | microserviceurl | String |  | | statusurl | String |  | | validateCounter | Integer |  | |
| **Condition** |  |
| **Connectors In** | |  |  |  |  | | --- | --- | --- | --- | | **Id** | **Type** | **Event** | **Ignore Errors** | | restUpdate111 | RESTConnector | ON\_ENTER | False | |
| **Connectors Out** | |  |  |  |  | | --- | --- | --- | --- | | **Id** | **Type** | **Event** | **Ignore Errors** | | waiting1 | scripting-groovy-script | ON\_FINISH | False | | restUpdate1111 | RESTConnector | ON\_FINISH | False | |
| **Description** | * **Check if issue resolved** is a Script Task. It has one Connector in and one operation. * After **Check if issue resolved** is executed it will reach **Reboot Done**. * After completion of this Activities on the node, The flow execution resumes   and will reach **Gateway8** and thereby will reach to bdt status (XOR Gateway) to  perform operations again.   * This loop of revalidating flow ends once it reaches the counter in the condition   specified on connector from **ValidateRebootCount** gateway. |
| **Diagrams** |  |

* + 1. Step 13: “Reboot ONT?” to “Reboot Done”

|  |  |
| --- | --- |
| **Type of Task** | Flow Execution for happy path scenario. |
| **Actor** |  |
| **Variables Used** | |  |  |  | | --- | --- | --- | | **Variables** | **Data Type** | **In / Out** | |  |  |  | |
| **Condition** |  |
| **Connectors In** | |  |  |  |  | | --- | --- | --- | --- | | **Id** | **Type** | **Event** | **Ignore Errors** | |  |  |  |  | |
| **Connectors Out** | |  |  |  |  | | --- | --- | --- | --- | | **Id** | **Type** | **Event** | **Ignore Errors** | |  |  |  |  | |
| **Description** | * This is happy path scenario to the end of the flow. |
| **Diagrams** | **Connectors:** |

* + 1. Step 14: “Reboot Done” to “End Node”

|  |  |
| --- | --- |
| **Type of Task** | Flow Execution |
| **Actor** |  |
| **Variables Used** | |  |  |  | | --- | --- | --- | | **Variables** | **Data Type** | **In / Out** | |  |  |  | |
| **Condition** |  |
| **Connectors In** | |  |  |  |  | | --- | --- | --- | --- | | **Id** | **Type** | **Event** | **Ignore Errors** | |  |  |  |  | |
| **Connectors Out** | |  |  |  |  | | --- | --- | --- | --- | | **Id** | **Type** | **Event** | **Ignore Errors** | |  |  |  |  | |
| **Description** | * Once Reboot Done XOR Gateway is reached then it will be the end of the flow. |
| **Diagrams** | **Connectors:** |

* + 1. Step 15: **All Groovy scripts used in the process**

|  |  |
| --- | --- |
| **Name of the Script** | **SCRIPT** |
| Command | **import** groovy.json.JsonSlurper  **import** java.util.Map;  **import** org.json.JSONArray;  **import** org.json.JSONObject;  **def** jsonSlurper = **new** JsonSlurper()  Map ct = (Map)jsonSlurper.parseText(customertopology)  **def** tp\_ref;  **def** neid;  **def** tr\_type = troubletype;  **def** tsys = "EMBEDDED";  **def** isBGRequest = **true**;  **def** isAngular = **true**;  **def** cktstatus = jsonSlurper.parseText(customertopology).D\_CKT\_ELEMENTS.cktStatus  **def** stagedop = "NO";  **def** tranid = transactionid;  **def** cid\_sel;  **def** inSvcType = "DATA"  **def** inCktType = "BHR"  **def** opStatus = "INIT"  JSONObject json = **new** JSONObject(customertopology);  JSONObject dCktElements = json.getJSONObject("D\_CKT\_ELEMENTS");  JSONObject dktInfo = json.getJSONObject("D\_CKT\_INFO");  cktstatus = dktInfo.getString("cktStatus");  System.***out***.println("Check Status-->"+cktstatus);  //Iterator<String> itr = dCktElements.keys();  **for** (Iterator<String> iterator = dCktElements.keys(); iterator.hasNext();) {  String key = (String) iterator.next();  JSONObject jsonChild = dCktElements.getJSONObject(key);  **if**(inCktType.equals(jsonChild.getString("elementClass"))){  System.***out***.println("The key is ---->"+key);  neid= key;  tp\_ref = key;  **break**;  }  }  JSONObject dCommandParams = json.getJSONObject("D\_COMMAND\_PARAMS");  JSONObject ontJson = dCommandParams.getJSONObject("rtrvOntPortStatus");  **for** (Iterator<String> ontIterator = ontJson.keys(); ontIterator.hasNext();) {  String key = (String) ontIterator.next();  JSONArray ontInfoArray = ontJson.getJSONArray(key);  String[] ontArray = ontInfoArray.toString().split(",");  **for** (**int** i = 0; i < ontArray.length; i++) {  String ontInfo = ontArray[i];  ontInfo = ontInfo.replace("\"", "");  ontInfo = ontInfo.replace("[", "");  ontInfo = ontInfo.replace("]", "");  **if**(ontInfo.toString().split("-")[0].toString().trim().toUpperCase().equals(inSvcType)){  cid\_sel = ontInfo.toString().split("-")[1].toString().trim().replace(" ","" ).replace("/", "");    }  }    }  **return** "{\"reqType\":\"ASYNC\",\"actionId\":\"gwrPingBhr\",\"stagedOp\":\"NO\",\"cktStatus\":\""+cktstatus+"\",\"SVCTYPE\_SEL\":\""+inSvcType+"\",\"CID\_SEL\":\""+cid\_sel+"\",\"TP\_REF\":\""+tp\_ref+"\",\"NEID\":\""+neid+"\",\"EL\_TYPE\":\""+inCktType+"\",\"TR\_TYPE\":\""+tr\_type+"\",\"TSYS\":\""+tsys+"\",\"opStatus\":\""+opStatus+"\",\"transactionId\":\""+tranid+"\",\"isBonita\":\"true\",\"isAngularResult\":\"true\",\"description\":\"Ping BHR\"}"; |
| getBHRMap111: Value Assigned to pingBHRMap | **import** org.json.JSONObject;  **import** org.json.JSONArray;  **import** com.verizon.delphi.util.common.CheckONTProperties;  Map<String, Object> map = **null**;  JSONObject json = **null**;  CheckONTProperties checkOntPropsObj = **new** CheckONTProperties();  String responseStr = manualTaskResult;  **boolean** isSuccesOrInc = **false**;  String status ="";  String trblCode = "";  /\*if(responseStr != null && !responseStr.isEmpty()){  responseStr = responseStr.replace("{", "{\"");  responseStr = responseStr.replace("=", "\":\"");  responseStr = responseStr.replace(", ", "\",\"");  responseStr = responseStr.replace("}", "\"}");      }  Map<String, Object> map = new HashMap<String, Object> ();  map.putAll(checkOntPropsObj.parseResponse(responseStr));\*/  **if**(responseStr != **null** && !responseStr.trim().isEmpty()){  responseStr = responseStr.replace("<?xml version='1.0' encoding='UTF-8'?><org.bonitasoft.connectors.rest.RESTResult><entity>", "");  responseStr = responseStr.replace("</entity><header/><time>0</time><statusCode>0</statusCode></org.bonitasoft.connectors.rest.RESTResult>", "");  map = **new** HashMap<String, Object>();  json = **new** JSONObject(responseStr.trim());  **if**(json.toString() != **null** && !json.toString().isEmpty()){  map = checkOntPropsObj.parseResponse(json.toString());      }  }  //return checkOntPropsObj.parseResponse(json.toString());  **return** map; |
| getBHRMap111: Value Assigned to hasGoodPing | **if**(**null** != manualTaskResult && !manualTaskResult.isEmpty()){  **def** res = **new** groovy.json.JsonSlurper().parseText(manualTaskResult) // Get Troublecode  **switch**(res.RESULTS.ANALYSIS.TRBL\_CODE.trim()){  **case** "TSDELTOK":  **return** **true**  **break**;  **case** "TSNWPTOK":  **return** **true**  **break**;  **case** "TSDELG17":  **return** **true**  **break**;  **case** "TSDELG22":  **return** **true**  **break**;  **case** "TSDELTMO":  **return** **true**  **break**;  **case** "TSDELXXX":  **return** **true**  **break**;  **case** "TSDELZ04":  **return** **true**  **break**;  **case** "TSDELZ08":  **return** **true**  **break**;  **case** "TSBASBER":  **return** **true**  **break**;  **case** "TSBASRFD":  **return** **true**  **break**;  **default**:  **return** **false**  **break**;  }  }**else**{  **return** **false**  } |
| wait | **try**{  Thread.*sleep*(30000)  } **catch**(Exception e){} |
| getBreakDialToneStatus: Value Assigned to breakDialToneStatus | **import** org.json.JSONObject;  **import** com.verizon.delphi.util.Response  JSONObject obj = **new** JSONObject();  **def** response;  **def** res = **new** groovy.json.JsonSlurper().parseText(responseString)  **switch**(res.RESULTS.ANALYSIS.TRBL\_CODE.toUpperCase().trim()){  **case** "TSDELTOK":  **return** "SUCCESS"  **break**;    **case** "TSONTBDN":  **case** "TSONTDNS":  **return** "NOTSUPPORTED"  **break**;    **case** "TSDELA07":  **case** "TSDELE30":  **case** "TSDELE31":  **case** "TSDELE34":  **case** "TSDELE57":  **case** "TSDELE68":  **case** "TSDELE73":  **case** "TSDELE78":  **case** "TSDELNEB":  **case** "TSDELRVW":  **case** "TSNOKCOM":  **case** "TSNOKTMO":  **case** "TSDELE25":  **return** "INCONCLUSIVE"  **break**;    **case** "TSOLTBDB":  **case** "TSOLTBCO":  **case** "TSCPEPNA":  **case** "TSCPEROH":  **case** "TSOLTBNC":  **case** "TSOLTBNR":  **case** "TSOLTBNS":  **case** "TSOLTBTF":  **case** "TSOLTBTS":  **case** "TSOLTCNF":  **case** "TSOLTOIN":  **case** "TSOLTPPD":  **case** "TSONTTDM":  **case** "TSOLTSNR":  **case** "TSOLTBDD":  **return** "FAILURE"  **break**;    **default**:  **return** "FAILURE"  **break**;  } |
| getBreakDialToneStatus: Value Assigned to pingStatus | **breakDialToneStatus (java.lang.String) “This is not a script, It is a variable!”** |
| getBreakDialToneStatus: Value Assigned to dialToneIssueResolved | **return** "SUCCESS".equalsIgnoreCase(breakDialToneStatus) |
| getBreakDialToneStatus: Value Assigned to initialBreakDialToneStatus | **breakDialToneStatus (java.lang.String) “This is not a script, It is a variable!”** |
| getRegDate | **import** java.text.ParseException;  **import** java.text.SimpleDateFormat;  **import** org.json.JSONArray;  **import** org.json.JSONObject;  String regDate = **null**;  JSONObject json = **new** JSONObject(responseString);  JSONObject D\_DATA\_GRAPH = json.has("D\_GRAPH\_DATA") ? json.getJSONObject("D\_GRAPH\_DATA"): **new** JSONObject();  JSONArray data = D\_DATA\_GRAPH.has("SIP REG HISTORY") ? D\_DATA\_GRAPH.getJSONObject("SIP REG HISTORY").getJSONArray("data"): **new** JSONArray();  Outerloop:  **for**(**int** i =0; i<data.length(); i++){  JSONArray details = data.getJSONArray(i);  **for**(**int** j =0; j<details.length();j ++){  System.***out***.println(details.get(j));  **if**("Expiration".equalsIgnoreCase(details.getJSONObject(j).getString("v"))){  regDate = details.getJSONObject(j+1).getString("v");  **break** Outerloop;  }  }  }  **if**(regDate != **null**){  SimpleDateFormat dateFormat = **new** SimpleDateFormat("EEE MMM d HH:m:s");  regDate = regDate.substring(0, 19);  **try** {  **return** dateFormat.parse(regDate);  }**catch** (ParseException e) {  // **TODO** Auto-generated catch block  **return** **null**;  }  }**else**{  **return** **null**  }  **return** **null**; |
| getRegistrationStatus: Value Assigned to registrationStatus | **def** status = "FAILURE"  **def** res = **new** groovy.json.JsonSlurper().parseText(responseString) // Get Troublecode  **try**{  **switch**(res.RESULTS.ANALYSIS.TRBL\_CODE.trim()){    **case** "TSDELTOK":  status = "SUCCESS"  **break**;    **case** "TSDELS01":  **case** "TSDELS02":  **case** "TSDELS11":  **case** "TSDELS16":  **case** "TSDELZ08":  **case** "CFOLTVIL":  **case** "CFOLTPNF":  status = "INCONCLUSIVE"  **break**;    **case** "TSBSFDNR":  **case** "TSCSKDNR":  status = "FAILURE"  **break**;    **default**:  status = "INCONCLUSIVE"  **break**;    }  }**catch**(Exception e){  status = "FAILURE";  }  **return** status.toString(); |
| getRegistrationStatus: Value Assigned to dialToneIssueResolved | **import** java.text.ParseException;  **import** java.text.SimpleDateFormat;  **boolean** dateUpdated = **false**;  **if**(registrationDate == **null** && newRegDate != **null**){  dateUpdated = **true**;  }**else** **if**(registrationDate != **null** && newRegDate != **null**){  dateUpdated = newRegDate.after(registrationDate);  }  **return** dateUpdated;  //return !"FAILURE".equalsIgnoreCase(registrationStatus) |
| issueResolved | **return** dialToneIssueResolved || registrationStatus ; |
| waiting1 | **try**{  Thread.*sleep*(30000)  }**catch**(Exception e){} |