opentext™

ArcSight SOAR CE 24.2 (v3.11)

Automation Bit Development Guide

opentext™

Contents

Description

This guide provides a practical reference for developing and using automation bits within ArcSight SOAR. It covers key concepts such as the scripting environment, usage patterns (manual, playbook-based, and nested), and provides detailed examples to interact with alerts, cases, scope items, and lists. The document also includes guidance on debugging and parameter handling to support robust and reusable automation bit development.

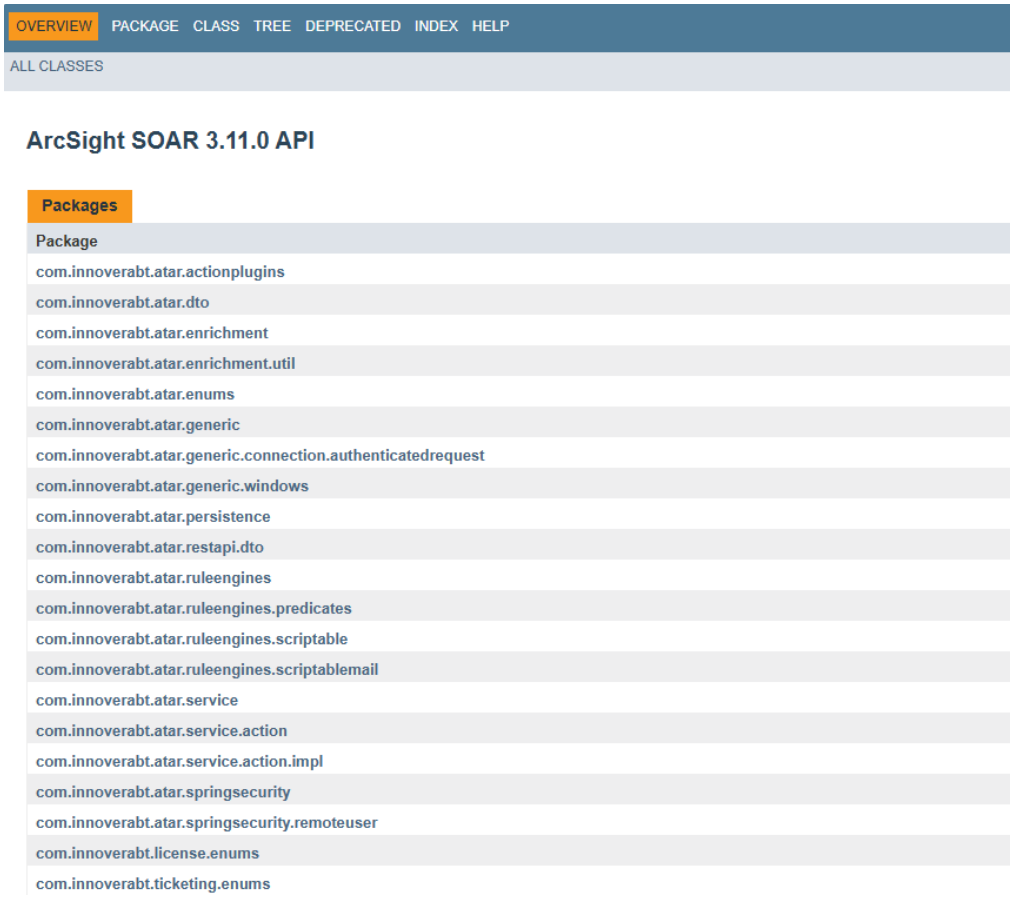
Intended audience

This document is intended for developers and analysts looking to extend SOAR functionality through custom automation.

Environment

Scripting API and Documentation

SOAR supports Jython compatibility in writing automation bits. To know more about the SOAR scripting API methods that you need while developing automation bits for SOAR refer to the ArcSight SOAR Scripting API, which is accessible from the following URL:
https://<CDF_Master_IP_or_FQDN>/soar/js-api-doc/index.html?tenant=default



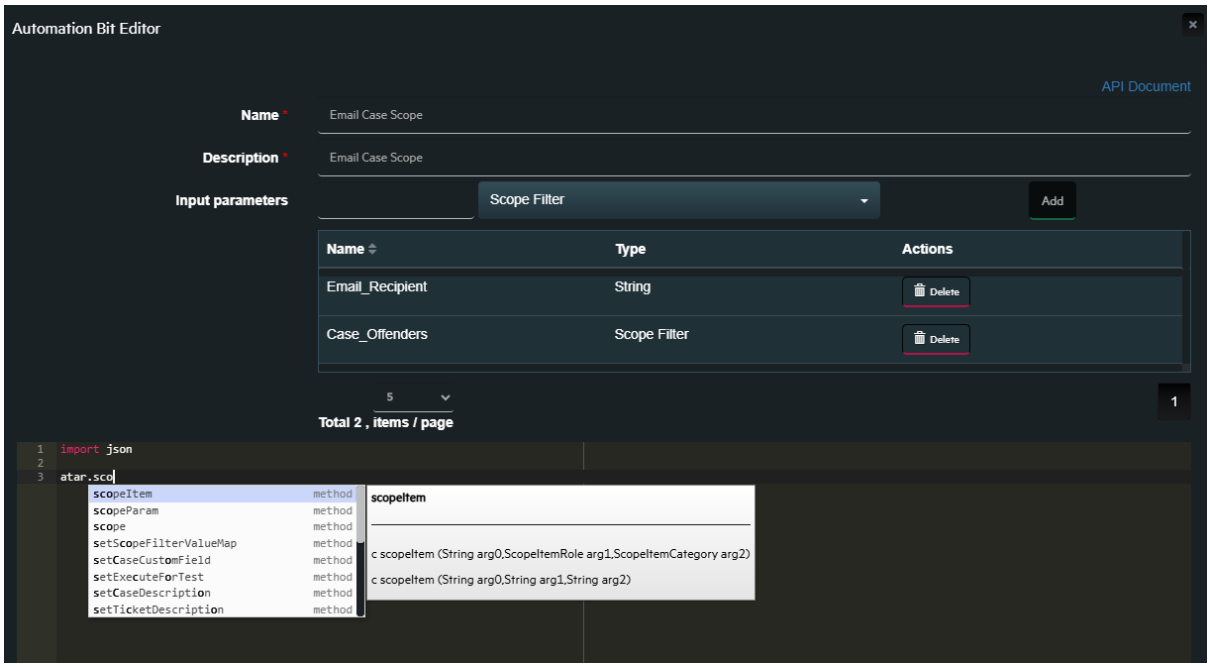
The ArcSight SOAR Scripting API contains all the packages that SOAR provides for scripting of automation bits, email parsers, plug-ins, etc., You can interact with alerts, cases and scope items within automation bits by using the methods defined under the following package:

com.innoverabt.atar.ruleengines.scriptable.ScriptableRuleEngineUtil

Automation Bit Editor

Automation bits can be written by using ArcSight SOAR's Automation Bit Editor. The editor allows you to do the following:

- Provide a name and short description for your automation bit code
- Define input parameters for your code
- Write code with auto-suggesting available scripting API methods and syntax highlighting
- Access technical documentation for scripting API.



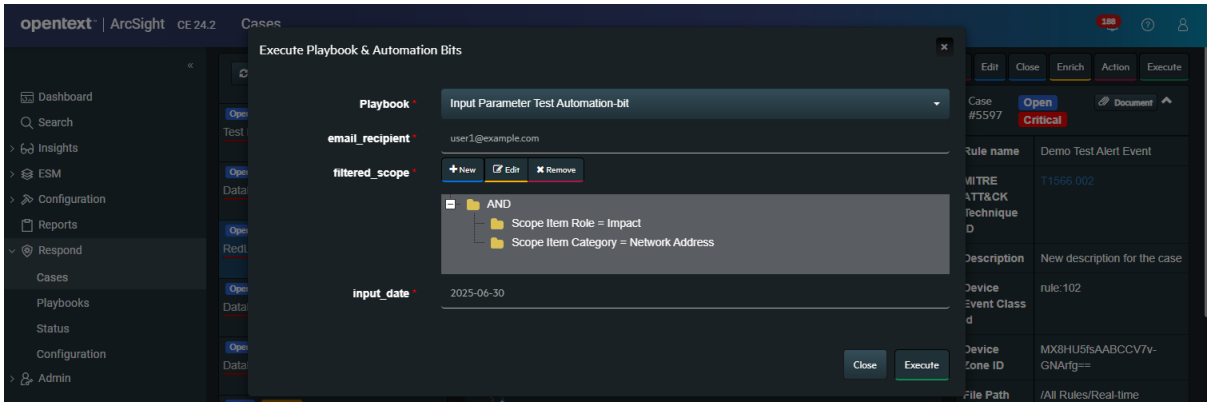
Using Automation Bits

You can use an automation bit on ArcSight SOAR in one of the following ways:

- Executing it manually while responding to an incident
- Using it as an automation step in a playbook
- Including it in another automation bit

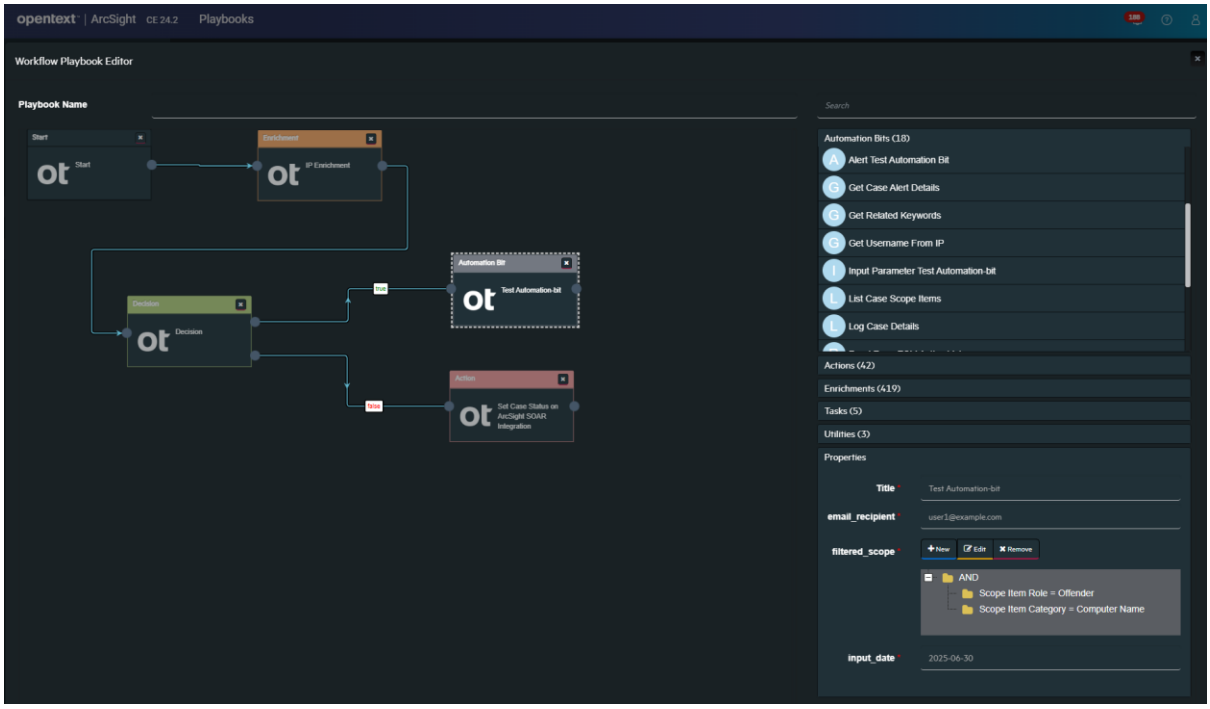
Executing manually while responding to an incident

You can execute an automation bit manually while investigating or responding to an incident. To trigger the execution, use the **Execute** button on the top right corner of the case screen and select the automation bit from the drop-down menu. If the automation bit requires input parameters, then you will be prompted to specify the values:



Using as an automation step in a Playbook

Your automation bit can be used as a workflow item in playbook like any other components such as enrichments and actions. You can access all your automation bits on the Workflow Playbook Editor and if the automation bit you use requires input parameters, you will be prompted to specify the parameters.



Calling within Another Automation Bit

An automation bit can be included into another one using the **atar.require()** method. This approach is useful for maintaining automation bits as a library and using them in other automation bit codes as needed.

Sample automation bits:

```
# Save this automation bit with the name "Case Scope Library"
def list_scope_items(category):
  scope_items = atar.scope(category)
  for scope_item in scope_items:
    value = scope_item.getScopeItem().getValue()
    role = scope_item.getRole().getName()
    category = scope_item.getScopeItem().getCategory().getName()
    alert_id = scope_item.getAlertId()
    print("Scope item: {}".format(scope_item))
    print("Role: {}".format(role))
    print("Category: {}".format(category))
    print("Value: {}".format(value))
    print("AlertID: {}".format(alert_id))
```

```
# This code includes the automation bit named "Case Scope Library"
atar.require("Case Scope Library")
```

```
category = "HASH"
#Print scope items with "HASH" category
list_scope_items(category)
```

Automation Bit Parameters

Automation bits allow you to define input parameters for your code. You can provide the values for those parameters during the execution time. The input parameters can be in one of the following types:

Parameter Type	Method to access	Description
Date	atar.param()	Returns date input as a string
Scope Filter	atar.scopeParam()	Returns the filtered scope items. You can iterate over the list to access individual scope items
String	atar.param()	Returns the input as a string

The following is a sample automation bit:

```
# Sample automation bit to demonstrate input parameters. It requires the following parameters:
# filtered_scope (Type: Scope Filter)
# email_recipient (Type: String)
# input_date (Type: Date)

filtered_scope = atar.scopeParam("filtered_scope")
email_recipient = atar.param("email_recipient")
input_date = atar.param("input_date")

print("Input for the automation bit:\n")
print("Email Recipient = {}".format(email_recipient))
print("Input Date = {}".format(input_date))
if len(filtered_scope) > 0:
    for scope_item in filtered_scope:
        print("Scope Item = {}".format(scope_item))
else:
    print("No scope item found matching the provided filter")
```

Alerts and Cases

Using automation bits, you can access the alert and case related methods to create new. You can also access the existing alert and case details to use these methods in your response activities such as updating case details or notifying analyst about the case.

Working with Alerts

Using automation bits, you can create new alerts in the system or interact with the alerts within a case. The following are some of the useful methods:

Method	Description

atar.alert(alertParameters, ticketParameters)	Creates an alert with the specified alert parameters. If ticket parameters are provided, a new incident is created with that alert. Otherwise, alert is created and processed by consolidation rules to determine if it turns into a new case or consolidated into an existing case.
atar.getAlert()	Returns the "**Alert**" object(s) of the case.
getParams()	Returns the alert parameters in JSON format. You can iterate over JSON to reach a particular field.
getDate()	Returns alert time.
getAlertSource()	Returns the source of the alert.
getAlertSourceRuleNames()[0].getRuleName()	Returns the rule name of the alert.

The following is a sample automation bit to access alert details:

```
import json

alert = atar.getAlert()
alert_time = alert.getDate()
alert_source = alert.getAlertSource()
alert_rule_name = alert.getAlertSourceRuleNames()[0].getRuleName()
alert_params = alert.getParams()

alert_param_device_vendor = json.loads(alert_params)['details']['deviceVendor']

print("Alert={}".format(alert))
print("Alert Time={}".format(alert_time))
print("Alert Source={}".format(alert_source))
print("Alert Rule Name={}".format(alert_rule_name))
print("Alert Parameters={}".format(alert_params))
print("Alert Parameter - Device Vendor={}".format(alert_param_device_vendor))
```

The following is a sample code to create an alert and a case:

```
# Automation bit sample to create a test alert and a case based on this alert.
# Please make sure that alertSourceName parameter matches the alert source name on your environment.

alertSourceName = "ArcSight ESM"
scopeItems = []
scopeItems.append(atar.scopeItem("evil@example.com", "OFFENDER", "EMAIL_ADDRESS"))
scopeItems.append(atar.scopeItem("citizen@corp.com", "IMPACT", "EMAIL_ADDRESS"))

alertParams = {
   "ALERTSOURCE": atar.alertSource(alertSourceName),
   "PARAMS": {},
   "RULENAME": "Test Alert Event",
   "EXTERNALID": "24621211",
   "SCOPEITEMS": scopeItems
};

ticketParams = {
   "SUBJECT": 'Test Incident',
   "DESCRIPTION": 'This is a test case created by the automation bit',
   "SEVERITY": 'Medium'
};
```

```
atar.alert(alertParams, ticketParams)
```

Working with Cases

You can use automation bits to interact with SOAR cases, for example, to get case details or set case attributes. The script API method names containing “case”, “incident” and “ticket” were implemented at different stages of the product’s life cycle and they all reference to the same thing on SOAR.

The following are some useful methods:

Method	Description
atar.getCase() or atar.getTicket()	Returns “**Ticket**” object.
getSerial()	Returns case id.
getTicketSeverity()	Returns “**TicketSeverity**” of the case. You can access its attributes using the methods available under this class.
getTicketStatus()	Returns “**TicketStatus**” of the case. You can access its attributes using the methods available under this class.
getSubject()	Returns title of the case.
getDescription()	Returns description of the ticket.
getComments()	Returns list of “**TicketComment**”. You can iterate over the list to reach the individual comments.
getAssignee()	Returns “**Principal**” of the case. You can access its attributes using the methods available under this class.
getCreationTime()	Returns case creation time.
atar.setCaseDescription()	Updates the case description field.
atar.setCaseSubject()	Updates the case title.
atar.addCaseComment()	Adds a new comment to the case.

The following is a sample automation bit to access some case attributes:

```
# Get case details:
case = atar.getCase()

case_serial = str(case.getSerial())
case_subject = case.getSubject()
case_description = case.getDescription()
case_creation_time = str(case.getCreationTime())

case_status = case.getTicketStatus().getName()
case_severity = case.getTicketSeverity().getName()

if case.getAssignee() is not None:
   case_assignee = case.getAssignee().getName()
else:
   case_assignee = "Unassigned"

# Initialize an empty variable for case_comment_content
```

```
case_comment_content = ""
case_comments = case.getComments()
for comment in case_comments:
   comment_created_by = comment.getCreatedByUser()
   if comment_created_by is None:
      comment_created_by = "SOAR Automation"

   case_comment_content = case_comment_content + "User: " + comment_created_by + "  - Comment: " + comment.getContent() + "
| "

print("Case Details")
print("==========")
print("ID: " + case_serial)
print("Title: " + case_subject)
print("Description: " + case_description)
print("Severity: " + case_severity)
print("Status: " + case_status)
print("Assignee: " + case_assignee)
print("Creation Time: " + case_creation_time)
print("Comments: " + case_comment_content)
```

The following is a sample automation bit to set the case's description field and add a new comment to the case timeline:

```
# Automation bit sample code to extract DeviceCustomString1 from ESM correlation event
# and set as case description

import json

# fetch alert and convert alert params to dict
alert_params = json.loads(atar.getAlert().params)

if("cs1" in alert_params["details"]["extension"]):
   atar.setCaseDescription( alert_params["details"]["extension"]["cs1"])

atar.addCaseComment("This is a new comment.")
```

Case Scope

You can access the case scope and the scope items (in other words case artifacts) within your automation bits. Some useful methods are:

Method	Description
atar.scope()	Returns the case "**Scope**" (AlertScopeItemDTO) You can iterate over the list of scope items to access individual scope items
getRole().getName()	Returns the role of the scope item
getScopeItem().getValue()	Returns the value of the scope item
getAlertId()	Returns the alert id which the scope item is associated with
atar.addScopeItem()	Adds an item (artifact) to case scope with a given role and category. The role can be any of:

	• IMPACT • OFFENDER • RELATED The category can be any of: • COMPUTER_NAME • EMAIL_ADDRESS • FILEDATA • FILENAME • HASH • HOST • KEYWORD • MAC_ADDRESS • NETWORK_ADDRESS • PROCESS • UNKNOWN • URL • USERNAME

The following is a sample automation bit to list scope items in a case:

```
scope_items = atar.scope()

for scope_item in scope_items:
    value = scope_item.getScopeItem().getValue()
    role = scope_item.getRole().getName()
    category = scope_item.getScopeItem().getCategory().getName()
    alert_id = scope_item.getAlertId()

    print("Scope item: {}".format(scope_item))
    print("Role: {}".format(role))
    print("Category: {}".format(category))
    print("Value: {}".format(value))
    print("AlertID: {}".format(alert_id))
```

Working with Lists

In addition to ArcSight ESM's active lists, ArcSight SOAR also provides capabilities to manage your lists under **Respond > Configuration > Lists** menu. Using Scripting API methods, you can interact with SOAR lists in your automation bits.

The following are some useful methods:

Method	**Description**
atar.list()	Returns the list as a JSON being the first column is "**key**" and the rest of the columns as "**value**" in array
atar.listGet()	Returns the row from a list with given key
atar.listAddItems()	Adds given items to the list
atar.listRemoveItems()	Removes items from the list

atar.listContainsAny()	Search the given list of keys on the list and returns True or False
atar.truncateList()	Truncates the list and deletes all the entries

The following is a sample automation bit to search for IP address in a list and add associated host and username to case scope:

```
# Automation bit for searching user and host information from an inventory list.\n
# If inventory is found, then associated username and hostname is added to case scope as "RELATED"
#
# Inventory List should be defined under ""Respond>Configuration>Lists"
# Columns: IP[Type:NETWORK ADDRESS], Username[Type:USERNAME], Host[Type:HOST]

inventory_list = "IP to User Inventory"

# In order to add a label to case when the IP address is not found in the inventory
# Define it under "Respond>Configuration>Cases>Labels"
inventory_not_found_label = "Inventory Not Found"

ip_scope_items = atar.scope("NETWORK_ADDRESS")
offender_ips = filter(lambda scope_item: scope_item.getRole().getName() == "OFFENDER", ip_scope_items)

if offender_ips:
  for offender_ip in offender_ips:
    offender_ip_value = offender_ip.getScopeItem().value
    inventory_row = atar.listGet(inventory_list, offender_ip_value)
    if len(inventory_row) > 0:
      username = inventory_row[0]
      host = inventory_row[1]
      atar.addScopeItem(username, "RELATED", "USERNAME")
      atar.addScopeItem(host, "RELATED", "HOST")
      print("IP address: {} is found in the inventory. Username={}, Host={}".format(offender_ip_value, username, host))
    else:
      print("IP address: {} is not found in the inventory.".format(offender_ip_value))
      atar.addTicketLabel(atar.getIncident(), inventory_not_found_label)
```

Encryption and Hashes

Scripting API provides methods to encrypt data and calculate hash values in automation bits. You can use the following methods:

Method	Description
atar.encryptAES()	Encrypts a message with given key
atar.decryptAES()	Decrypts an encrypted message with a given key
atar.md5()	Calculates MD5 hash value of a message
atar.sha256()	Calculates SHA-256 hash value of a message
atar.sha512()	Calculates SHA-512 hash value of a message

Sample automation bit to demonstrate encryption and hash calculation:

```
message = "This_is_my_message"
key = "cx5fo993df33fjcv"

# Encrypt & Decrype the message
encrypted_message = atar.encryptAES(message,key)
decrypted_message = atar.decryptAES(encrypted_message,key)

# Calculate Hash Values
md5_message = atar.md5(message)
sha256_message = atar.sha256(message)
sha512_message = atar.sha512(message)

print("Encrypted Message = {}".format(encrypted_message))
print("Decrypted Message = {}".format(decrypted_message))
print("MD5 of Message = {}".format(md5_message))
print("SHA-256 of Message = {}".format(sha256_message))
print("SHA-512 of Message = {}".format(sha512_message))
```

Notifications

In addition to ArcSight SOAR's SMTP server integration capabilities, you can also use **atar.notify()** method in automation bits to send email notifications to analysts and users.

The following is a sample automation bit:

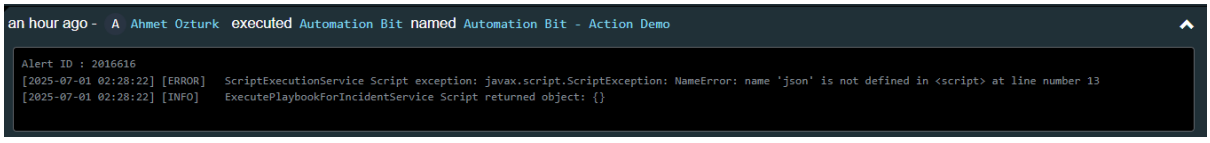
```
# Automation bit code sample to email incident details to a recipient.
# In order this automation bit to work, please add an input parameter "Recipient (Type: String)"
# while creating the automation bit.
# atar.notify() method uses the SMTP server defined in Respond>Configuration>Parameters>OutgoingMailIntegration

# Get parameter(s) provided:
email_to = atar.param('Recipient')

# Get case details
case  = atar.getCase()
case_serial = str(case.getSerial())
case_subject = case.getSubject()
case_severity = case.getTicketSeverity().name
if case.getAssignee() is not None:
   case_assignee  = case.getAssignee().getName()
else:
   case_assignee = "Unassigned"
case_status = case.getTicketStatus().getName()
case_description = case.getDescription()
rulename = atar.getAlert().getAlertSourceRuleNames()[0].getRuleName()

# Initialize an empty variable for case_comment_content
case_comment_content = ""
case_comments = case.getComments()
for comment in case_comments:
   comment_created_by = comment.getCreatedByUser()
   if comment_created_by is None:
      comment_created_by = "SOAR Automation"

   case_comment_content = case_comment_content + "User: " + comment_created_by + "  - Comment: " + comment.getContent() +
"<br>"
```

```
# Build email subject and body
email_subject = "New Comment on SOAR Case #(" +  case_serial  + ") - " + case_subject

email_body = "<html><head>" \
        "<style> table { font-family: arial, sans-serif; border-collapse: collapse; width: 100%; } td, th { border: 1px solid #dddddd; text-
align: left; padding: 8px; } tr:nth-child(even) { background-color: #dddddd; } </style>" \
        " <meta  charset=\"UTF-8\"></head>" \
        "<body>" \
        "Hello,  <br><br> " \
        "We want to inform you that a new comment has been added to the SOAR case #" + case_serial + ".<br><br>" \
        "Case Details:<br>" \
        "<table> " \
        "<tr><td><b>Case</b></td><td><a href=\"https://arcsight/soar/?tenant=default#/cases/" + case_serial + "\"> " + case_serial +
" - " + case_subject + "</a></td</tr>" \
        "<tr><td><b>Description</b></td><td>" + case_description +  "</td></tr> " \
        "<tr><td><b>Rule Name</b></td><td>" + rulename +  "</td></tr> " \
        "<tr><td><b>Assignee</b></td><td>" + case_assignee + "</td></tr> " \
        "<tr><td><b>Severity</b></td><td>" + case_severity + "</td></tr> " \
        "<tr><td><b>Status</b></td><td>" + case_status + "</td></tr> " \
        "<tr><td><b>Comments</b></td><td>" + case_comment_content + "</td></tr> " \
        "</table>" \
        "<br><br>Kind Regards, <br>SOC Team!<br></body></html>"

# Send Email
atar.notify("EMAIL", [email_to], email_subject, email_body)
```

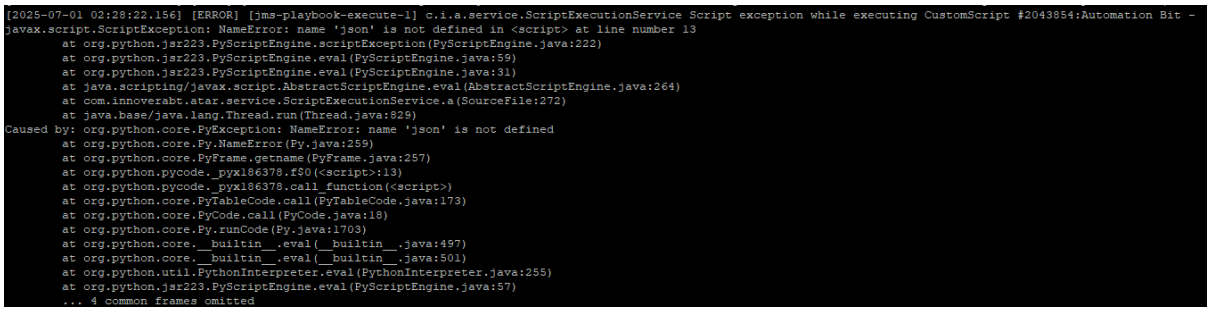
Debugging

Automation bits record the execution logs and error messages in case timeline as well as the soar-web-app pod's logs.

On case timeline, you can expand the Automation Bit execution entries to view the output of the automation bit:



The soar-web-app pod's logs provide a detailed stack trace of any error:



About OpenText

OpenText enables the digital world, creating a better way for organizations to work with information, on-premises or in the cloud. For more information about OpenText (NASDAQ/TSX: OTEX), visit opentext.com.

Connect with us:

OpenText CEO Mark Barrenechea's blog

Twitter | LinkedIn

