1 | User Activity Logging REST API

1.1 | Concept: User Activity Logging REST API

The User Activity Logging REST API enables developers and administrators to retrieve log entries that contain information about Workday user activities. This REST API provides access to the **User Activity** report data source.

Before submitting User Activity Logging REST API requests, begin logging the user activity in the tenant:

1. Access the **Edit Tenant Setup - System** task.
2. Select the **Enable User Activity Logging** option.

For reference information about the User Activity Logging REST API, see the `activityLogging` resource in the *Privacy* REST web service documentation.

URL Base Path

The User Activity Logging REST API uses this base path:

```
https://<tenantHostname>/api/privacy/<version>/<tenantName>
```

Example:

```
https://company.tenant.com/api/privacy/v1/gms
```

Usage

- Security Considerations
- Getting the Count of User Activity Instances
- Retrieving User Activity Instances
- Filtering the User Activity Instances
- Best Practices

Security Considerations

Workday secures the User Activity Logging REST API to the *System Auditing* domain in the System functional area.

When you submit the User Activity Logging REST API, you can view the activity logs for all users. However, if your security profile doesn't permit you to access the reported tasks in the activity logs, you don't have access to the tasks in Workday.

Review your internal security and privacy (GDPR) procedures to determine how you want to use the User Activity Logging REST API.

Getting the Count of User Activity Instances

To estimate the size of the user activity records efficiently without actually fetching the data, you can get the count of the records using the `returnUserActivityEntryCount` parameter.

To get the count of user activity instances in a given time frame, use this endpoint and parameters:

```
GET /activityLogging?from={fromDateTime}&to={toDateTime}&returnUserActivityEntryCount=true
```

Where:

{fromDateTime}
The required date and time of the earliest log entry. Use the UTC time zone.

{toDateTime}
The required date and time of the latest log entry. Use the UTC time zone.

returnUserActivityEntryCount=true
If true, the endpoint returns only the count of user activity instances in the specified time frame.

The response data contains a `userActivityEntryCount` value.

Sample request:

Get the count of the user activity instances from May 15, 2022 to June 3, 2022.

```
GET /activityLogging?from=2022-05-15T00:00:00Z&to=2022-06-02T23:59:59Z&returnUserActivityEntryCount=true
```

Sample response:

```
{
    "total": 1,
    "data": [
        {
            "userActivityEntryCount": 10219
        }
    ]
}
```

Retrieving User Activity Instances

To retrieve user activity instances within a given time frame, use this endpoint:

```
GET /activityLogging?from={fromDateTime}&to={toDateTime}&instancesReturned={instancesReturned}
```

Where:

{fromDateTime}
The required date and time of the earliest log entry. Use the UTC time zone.

{toDateTime}
The required date and time of the latest log entry. Use the UTC time zone.

{instancesReturned}
A number between 1 and 25 for fine-tuning the retrieval mechanism. Workday recommends a value of `1`, which gives the best performance.

`instancesReturned=1` retrieves 10,000 instances. If there are more than 10,000 instances, iteratively call the endpoint using the datetime of the last record as the `{fromDateTime}` parameter. For details about the iterative calls, see Retrieve All User Activities in a Time Period.

The `GET /activityLogging` endpoint also provides the `limit` and `offset` pagination parameters. The default `limit` is 20 (maximum is 1000), and the default `offset` is `0`.
Note: To ensure that you retrieve the complete set of data, iteratively call the `GET /activityLogging` endpoint with the `limit` and `offset` parameters. Set `limit=1000`. Initially set `offset=0`, then increment it by 1000 in subsequent calls. For details about the iterative calls, see Retrieve All User Activities in a Time Period.

Sample request:

Get the user activity instances from May 15, 2022 to June 3, 2022.

```
GET /activityLogging?from=2022-05-15T00:00:00Z&to=2022-06-02T23:59:59Z
```

Note: This example uses the minimum required parameters. For best practice, see the Best Practices section.
Sample response:

```
{
    "total": 10219,
    "data": [
        {
            "sessionId": "f3f7eb",
            "target": {
                "descriptor": "wd-environments / Workday Production Automation",
                "id": "2e89495897384ea9b2eab4341dc24dca"
            },
            "taskDisplayName": "Start New Session (Web Service)",
            "requestTime": "2022-05-16T11:55:38.338Z",
            "taskId": "03f043df676e40cfad46bc576f89b07f",
            "ipAddress": "64.124.173.190",
            "deviceType": "Desktop",
            "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.64
             Safari/537.36",
            "activityAction": "READ",
            "systemAccount": "wd-environments"
        },
        ...
        ...
        {
            "sessionId": "f3f7eb",
            "target": {
                "descriptor": "Test",
                "id": "921606317e300154c4558ae0995d0000"
            },
            "taskDisplayName": "View Workbook",
            "requestTime": "2022-05-16T14:17:13.649Z",
            "taskId": "fb47dca0a89210001e2115c2bb6c0257",
            "ipAddress": "64.124.173.190",
            "deviceType": "Desktop",
            "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.64
             Safari/537.36",
            "activityAction": "READ",
            "systemAccount": "wd-environments"
        }
    ]
}
```

The sample `data[]` array contains only 20 records because the default `limit` is 20. To retrieve all 10,219 records, iteratively call the `GET /activityLogging` endpoint with the `limit` and `offset` parameters. For details about how to call the `GET /activityLogging` endpoint iteratively, see Retrieve All User Activities in a Time Period.

Filtering the User Activity Instances

You can filter the log entries by users and tasks. To get the log entries for specific users and tasks, specify 1 or more `systemAccount` or `taskId` query parameters, respectively.

Sample request:

Return all user activity instances from May 15, 2022 to June 3, 2022, where Logan McNeil or Teresa Serrano accessed the **My Reports** or **Request Time Off** reports.

1. Send the `GET /activityLogging` endpoint with the specified timeframe:

   ```
   GET /activityLogging?from=2022-05-15T00:00:00Z&to=2022-06-02T23:59:59Z&limit=1000
   ```

2. Search the response to find an entry where Logan McNeil or Teresa Serrano accessed the **My Reports** and **Request Time Off** reports. Then, copy the taskIds associated with those reports:

   ```
   {
     "total": 10219,
     "data": [
   ...
   ...
   {
               "deviceType": "Desktop",
               "taskId": "a2b801281b5f100013d6f884ceb50047",
               "requestTime": "2022-06-02T20:48:57.693Z",
               "taskDisplayName": "My Reports",
               "systemAccount": "lmcneil",
               "ipAddress": "173.226.103.218",
               "sessionId": "56dc20",
               "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63
                Safari/537.36",
               "activityAction": "READ"
           },
   ...
   ...
           {
               "deviceType": "Desktop",
               "taskId": "eb0bf32f15b1498c9d7d213523811770",
               "requestTime": "2022-06-02T21:28:21.341Z",
               "taskDisplayName": "Request Time Off",
               "systemAccount": "lmcneil",
               "ipAddress": "173.226.103.218",
               "target": {
                   "descriptor": "Logan McNeil (CHRO, HRExe, Mgr 4000, MatMgr, ProMgr,TalMgr,VPRept)",
                   "id": "3aa5550b7fe348b98d7b5741afc65534"
               },
               "sessionId": "57b83f",
               "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63
                Safari/537.36",
               "activityAction": "READ"
           },
   ...
   ...
      ],
   }
   ```

3. Send the `GET /activityLogging` endpoint again with these additional query parameters:
   - `systemAccount`: lmcneil
   - `systemAccount`: tserrano
   - `taskId`: a2b801281b5f100013d6f884ceb50047
   - `taskId`: eb0bf32f15b1498c9d7d213523811770

   ```
   GET /activityLogging?from=2022-05-15T00:00:00Z&to=2022-06-02T23:59:59Z&systemAccount=tserrano&systemAccount=lmcneil&taskId=a2b801281b5f100013d6f884ceb50047
   ```

Best Practices

To tune the performance and to ensure that you retrieve the complete set of records in a time period, iteratively call the `GET /activityLogging` endpoint with these recommended parameter values:

Parameter	Recommended Value
`limit`	`1000`
`offset`	Initially set to `0`, then increment by `1000` in subsequent calls.
`instancesReturned`	`1`

Examples of iterative calls:

```
GET /activityLogging?from=2022-05-15T00:00:00Z&to=2022-06-02T23:59:59Z&limit=1000&instancesReturned=1&offset=0
```

```
GET /activityLogging?from=2022-05-15T00:00:00Z&to=2022-06-02T23:59:59Z&limit=1000&instancesReturned=1&offset=1000
```

```
GET /activityLogging?from=2022-05-15T00:00:00Z&to=2022-06-02T23:59:59Z&limit=1000&instancesReturned=1&offset=2000
```

Related Information

Concepts
Concept: User Activity Logging
Reference
Workday REST Services Directory

1.2 | Retrieve All User Activities in a Time Period

Prerequisites

- Select the **Enable User Activity Logging** option in the **Edit Tenant Setup - System** task.
- Security: *System Auditing* in the System functional area.

Context

This procedure describes how to call the `GET /activityLogging` endpoint iteratively with the `instancesReturned`, `limit`, and `offset` parameters. This iterative approach enables you to retrieve the complete set of user activity records within a time period efficiently.

The examples in this procedure assume that the endpoint URL in `<base_url>` is:

```
http://<tenantHostname>/api/privacy/v1/<tenant>/activityLogging
```

Steps

1. Set these query and pagination parameters for the `GET /activityLogging` endpoint.

Option	Description
from	The date and time of the earliest log entry. Use the format `{yyyy}-{mm}-{dd}T{hh}:{MM}:{ss}Z`. Ensure that the time precision is second (not millisecond). The time is in Zulu time/UTC. Example: *{2022}-{04}-{17}T{08}:{15}:{00}Z*
to	The date and time of the latest log entry. Use the format `{yyyy}-{mm}-{dd}T{hh}:{MM}:{ss}Z`. Ensure that the time precision is second (not millisecond). The time is in Zulu time/UTC. Example: *{2022}-{04}-{17}T{09}:{00}:{00}Z*
limit	Specify 1000, which is the maximum. The API is more performant for complete processing if the `limit` is higher. We recommend 1000.
instancesReturned	Specify 1. For improved performance, set `instancesReturned` to 1, which fetches 10,000 records per request.
offset	Initially set to `0`. To retrieve the first set of records, set `offset` to `0`. You'll increment `offset` by 1000 in subsequent calls.

2. Initialize a data collection for accumulating the endpoint response.
3. Repeat these steps *until* the size of the endpoint response is less than the `limit`, which is 1000.
   a. Call `GET /activityLogging` with the query and pagination parameters.
      The endpoint returns the requested user activity records within the `from` and `to` times, starting with the record at the *offset* position, sorted by time.
   b. Add the endpoint response to the data collection.
   c. Save the timestamp of the 1,000th record.
      You'll use the saved timestamp as the `from` parameter for fetching the next set of 10,000 records. To ensure that the time precision of the `from` parameter is second, remove the millisecond portion of the saved timestamp.
   d. Increment the `offset` parameter by 1000.

   This example illustrates the iterative calls with the incremented `offset` parameter:
   First call:

   ```
   GET <base_url>?from=2022-04-17T08:15:00Z&to=2022-04-17T09:15:00Z&limit=1000&instancesReturned=1&offset=0
   ```

   Second call:

   ```
   GET <base_url>?from=2022-04-17T08:15:00Z&to=2022-04-17T09:15:00Z&limit=1000&instancesReturned=1&offset=1000
   ```

   Third call:

   ```
   GET <base_url>?from=2022-04-17T08:15:00Z&to=2022-04-17T09:15:00Z&limit=1000&instancesReturned=1&offset=2000
   ```

4. If the 10th call (with `offset=9000`) still returns 1000 rows, you've reached the 10,000th record. To retrieve the next set of 10,000 records:

a. Change the `from` parameter to the saved timestamp (with second precision) of the last record.
b. Reset the `offset` pagination parameter to `0`.
c. Repeat the iterative `GET /activityLogging` calls *until* the number of records in the endpoint response is less than the `limit`.

This example illustrates the 10th call and the subsequent iteration of calls starting with the new `from` value and `limit=0`.
Tenth call:

```
GET <base_url>?from=2022-04-17T08:15:00Z&to=2022-04-17T09:15:00Z&limit=1000&instancesReturned=1&offset=9000
```

If the size of the 10th endpoint response is 1000, and the timestamp of the last record is `2022-04-17T08:35:17Z`, then these queries are the subsequent calls:

```
GET <base_url>?from=2022-04-17T08:35:17Z&to=2022-04-17T09:15:00Z&limit=1000&instancesReturned=1&offset=0

GET <base_url>?from=2022-04-17T08:35:17Z&to=2022-04-17T09:15:00Z&limit=1000&instancesReturned=1&offset=1000

GET <base_url>?from=2022-04-17T08:35:17Z&to=2022-04-17T09:15:00Z&limit=1000&instancesReturned=1&offset=2000
```

Continue the iterative calls until the number of records in the endpoint response is less than 1000.

Example

This pseudocode illustrates the iterative calls that retrieve all user activity records within the `from` and `to` datetimes.

```
limit = 1000
instancesReturned = 1
data = []
repeat
  offset = 0
  repeat
    results = callAPI(from, to, limit, offset, instancesReturned)
    Add results to data
    offset += limit
  until length(results) < limit
  from = timestamp_of_the_last_entry
until 0 entries
return data
```

1.3 | Example: Call the User Activity Logging REST API Using Python

This example illustrates how a Python client can call the User Activity Logging REST API using the iterative approach described in Retrieve All User Activities in a Time Period. This iterative approach ensures that the User Activity Logging REST API efficiently retrieves the complete set of user activity records in a given time period.

Scenario

Your company uses Splunk to monitor user activities in Workday. A Splunk plugin can get the complete set of user activity records for a given time period using a Python client. The Python code iteratively calls the User Activity Logging REST API with query and pagination parameters.

Note: Attempting to get all user activity records for a time period in a single call is inefficient and might not return complete data. Instead, write a client that calls the User Activity Logging REST API with query and pagination parameters. You can then iteratively call the client to return paginated responses.

Steps

1. In Python, create a `UserActivityClient` class.
2. Define the attributes and function for client authentication.
   You can use a bearer token to authenticate calls to the User Activity Logging REST API. The token depends on various connection parameters such as the user, server, password, and others.
   Specify these authentication attributes:

Option	Description
token_endpoint	The URL for authenticating the service.
_client_id	The client ID.
_client_secret	The client password.
access_token	The last refresh token. Initialize as `None`. For subsequent calls, pass the last refresh token.

This sample function, `refresh_access_token()`, posts the credentials to the authentication URL and returns an access token for subsequent API calls. Alternatively, you can obtain the token in another way and assign it to `self.access_token`. Note that the functions in this topic use the `refresh_access_token()` function for client authentication and for refreshing the token when needed.

```
class UserActivityClient:
    ...
    def refresh_access_token(self):
        resp = requests.Session().post(
            self.token_endpoint,
            auth=(self._client_id, self._client_secret),
            data={"refresh_token": self.access_token,
                  "grant_type": "refresh_token"},
            timeout=(self.CONNECT_TIMEOUT, self.READ_TIMEOUT)
        )
        resp.raise_for_status()
        data = resp.json()
        if "access_token" in data:
            self._access_token = data["access_token"]
            return self._access_token
        else:
            raise ValueError("<appropriate error message>")
```

3. Define the attributes for the User Activity Logging REST API calls.
   Consider the recommended values:

Option	Description
CONNECT_TIMEOUT	The timeout (in seconds) for connecting to the API server. Recommended value: 5
READ_TIMEOUT	The timeout (in seconds) for reading data from the API server, after the server connection. Recommended value: 60
MAX_RETRIES	The maximum number of times to retry if a call fails. Recommended value: 2
limit	The number of user activity records in a response. The maximum is 1000. A higher value makes the overall process more efficient. Recommended value: 1000
instances_returned	The value multiplied by 10,000 determines the number of records to fetch. The value of 1 gives the best performance. The `instancesReturned` parameter of the `GET /activityLogging` endpoint uses this value. Recommended value: 1
rest_url	The full URL of the User Activity Logging REST API endpoint. `http://<tenantHostname>/api/privacy/v1/<tenant>/activityLogging`
access_token	The bearer token for authentication. In this example, the `refresh_access_token` function returns the token.

4. Define the function that submits the REST API request and gets the JSON response.
   Import the `requests` package for the REST API request and response methods.
   This `get()` function returns a forward-iterable cursor, `resp`, which contains a paginated set of user activity records. This code assumes that when the token is invalid, it automatically calls the `self.refresh_access_token()` function to obtain a new token. If you use an alternate method for authentication, modify the code appropriately.

```
import requests
import random
class UserActivityClient:
    ...
    def get(self, *args, **kwargs):
        if "timeout" not in kwargs:
            kwargs["timeout"] = (self.CONNECT_TIMEOUT, self.READ_TIMEOUT)
        headers = kwargs["headers"]
        headers["Authorization"] = "Bearer {}".format(self.access_token)

        retry_count = 0
        url = self.rest_url
        while True:
            try:
                resp = requests.Session().get(url, **kwargs)
                resp.raise_for_status()
            except requests.exceptions.Timeout as e:
                if retry_count < self.MAX_RETRIES:
                    retry_count += 1
                    time.sleep(random.randint(1, 5))
                    continue
                else:
                    raise

            except requests.exceptions.HTTPError as e:
                if retry_count < self.MAX_RETRIES:
                    retry_count += 1
                    time.sleep(random.randint(1, 5))
                    if e.response.status_code == 401:
                        self.refresh_access_token()
                    continue
                else:
                    raise

            else:
                return resp
```

5. Define the function that calls the User Activity Logging REST API.
   This `make_one_call()` function calls the `get()` function, which returns an iterable cursor, `resp`.

   The `make_one_call()` function accepts the `from_time` and `to_time` datetime parameters that use the format `{yyyy}-{mm}-{dd}T{hh}:{MM}:{ss}Z`.

   The `self.instances_returned` value is 1, which indicates that the endpoint will attempt to fetch 10,000 records within the given time period.

   The `offset` parameter specifies from which offset position in the 10,000 records to return as the first record in the cursor. The `self.limit` specifies the number of records in the cursor. To retrieve the complete data, you'll call the `make_one_call` function multiple times with incremental `offset` values.

```
class UserActivityClient:
    ...
  def make_one_call(self, from_time, to_time, offset):
        resp = self.get(headers = { "Content-Type": "application/json" },
                        params = {"from": from_time,
                                  "to": to_time,
                                  "type": "userActivity",
                                  "instancesReturned": self.instances_returned,
                                  "limit": self.limit,
                                  "offset" : offset,
                                  }
                        )
        return resp
```

   Optionally, you can use these query parameters to filter the data by user or task. Example:

```
"systemAccount": "lmcneil",
"taskId" : "dc38ed1c446c11de98360015c5e6daf6"
```

6. Define a function that iteratively gets all 10,000 records in a time period.
   This `process_batch()` function iteratively calls the `make_one_call()` function. The `process_batch()` function processes up to 10,000 records at a time (or less in the last batch). In each batch, the `process_batch()` function calls the API multiple times with the incremented `offset`, and fills the `data` list with the response data. The `process_batch()` function returns the timestamp of the last record or `0` if the cursor is empty.

```
class UserActivityClient:
    ...
    def process_batch(self, st, et, data):
        offset = 0
        while True:
          resp = make_one_call(st, et, offset)
          if (len(resp)) == 0:
             # If there is no records, set last_timestamp to 0
             # to indicate the end of data.
             last_timestamp = 0
             break
          for r in response:
             # Extract data from each record to data.
             data.append(r.jason())
             last_timestamp = r['requestTime']
          offset += self.limit
       return last_timestamp
```

7. Define a function that iteratively gets a batch of 10,000 records.
   This `process_duration()` function iteratively calls the `process_batch()` function. Given a start time `st` and an end time `et`, `process_duration()` gets all user activity records in the time frame by processing 10,000 records at a time. For each new batch of 10,000 records, use a new start time that is equal to the timestamp of the last record retrieved.

```
class UserActivityClient:
    ...
    def process_duration(self, st, et):
        t = st
        data = []
        while t < et:
          last_timestamp = process_batch(self, t, et, data)
          if last_timestamp == 0:
            break
          t = truncate_to_second(last_timestamp)
```

8. From the Splunk monitoring tool, call the Python client.

   These steps describe how to monitor user activity in Splunk using the Python client. You can adjust these steps according to your requirements.

   a. Call `UserActivityClient.process_duration()`.
   b. For the initial run, set these parameters:

Parameter	Value
end_time	current_time - 5 minutes Example: If the current time is 2:20, end_time is 2:15.
start_time	end_time - 1 hour Example: If the current time is 2:20, start_time is 1:15.

   c. For subsequent runs, set these parameters:

Parameter	Value
end_time	current_time - 5 minutes
start_time	The time (with second precision) in the last record of the previous run.

Result

The Python client returns a JSON data collection containing the complete set of all user activity records within the requested time period.

Related Information

Tasks

Retrieve All User Activities in a Time Period