Guide

ADP Marketplace ESI - Event notifications

Published on
Apr 04, 2022, 11:53 AM

Last modified
Feb 11, 2026, 01:48 PM

Always Designing for People™

ADP Copyright Information

Published on
Apr 04, 2022, 11:53 AM

Published on
Feb 11, 2026, 01:48 PM

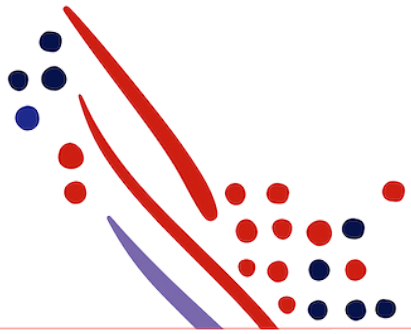


Table of Contents

Chapter 1

Event notifications

Event Notifications allow your application to be notified of data changes. They can be used to retrieve the latest data, to sync your system with ADP products and applications. When data is updated in an ADP application, that application generates an Event Notification message. This message is stored in the Event Manager database, waiting for a subscribed consumer.

Examples:

- Creating/disabling a user in the facilities ticketing system when the employee is joining or leaving the company
- Update the name of supervisor registered in a purchasing system when the employee transfers to another department
- Register a vacation day in a time sheet application when the request for leave is approved.

Event notifications give you the opportunity to work with small amounts of transaction data without the need to use batch processing.

About event notifications

ADP Representational State Transfer (REST) APIs use an event-based pattern for resource modification. Clients or partners need to know if there are any changes in the System of Record (SOR) at ADP to act upon or keep themselves aware . Events mark every change made in the system. Partners or clients will get to know about the changes after they subscribe to the specific changes for which they want to be notified. Event Notifications provide the details about the changes to partners. This prompts action by the partners and keeps their records updated.

Note

Event notifications are used in near real-time scenarios and it's recommended to check the event notification list on scheduled basis with at least a maximum of one hour.

Required setup steps

To subscribe to an event notification, contact your ADP Representative.

Data dictionary

This dictionary describes the fields you will typically find in the event notifications list and in a single event. For SOR specific examples, you can check the SOR documentation.

Fields in the event notification list

These are the field you can find in the list of notifications by doing a request to /core/v1/event- notification-messages. The events in that list need to be act upon by calling them with the callback.

Resource location	Description	Sample

/events		<pre>{
		  "events": []
		}</pre>

/events/eventID	Indicates the unique identifier of the event instance. This is set by the system of record (SOR) after an event is recorded as in progress or complete.	<pre>{
		  "events": [
		  {
		  "eventID": "12456-3445-22424"
		  }
		  ]
		}</pre>

Resource location	Description	Sample

/events/eventNameCode	Indicates the canonical name of the event. For example, dependent.add and worker.hire. This field is always present and valued based on a standard codelist.	{
		"events": [
		{
		"eventNameCode": {
		"codeValue": "worker.hire"
		}
		}
		]
		}

/events/recordDateTime	Specifies the date and time the event is recorded in the SOR with an event status code equal to complete. This value is set by the SOR.	{
		"events": [
		{
		"recordDateTime": "2022-02-28T11:09:46Z"
		}
		]
		}

/events/creationDateTime	Specifies the date and time the event is created. Necessary for the order in which events are consumed.	{
		"events": [
		{
		"creationDateTime": "2022-02-28T11:10:22.005Z"
		}
		]
		}

/events/originator	Indicates an originator that triggered the event. This can be either a user, machine or event.	{
		"events": [
		{
		"originator": {
		"associateOID": "ADPNL1234567"
		}
		}
		]
		}

/events/links	The callback uri you need to call to get the event details; the actual data	{
		"events": {
		"links"
		[
		{
		"rel": "full",
		"href": "/events/hr/v1/worker.hire/5f3d1d4895d89c19d8ae3034"
		}
		]
		}
		}

Common fields in the event notifications

These are the fields you will find when you GET a single event notification through its callback url found in the event notification list. It contains all the business information.

Resource location	Description	Sample

/events/data/ output	Specifies the event result.	{
		"events": [
		{
		"data": {
		"eventContext": {
		"worker": {
		"associateOID": "G3Q7JN6K3CNJJRQH "
		}
		},
		"output": {
		"worker": {
		"businessCommunication": {
		"mobile": {
		"itemID": "40439828_3139",
		"nameCode": {
		"codeValue": "Work Cell",
		"shortName": "Work Cell"
		},
		"countryDialing": "1",
		"areaDialing": "973",
		"dialNumber": "7133458",
		"access": "1",
		"formattedNumber": "(973) 713- 3458"
		}
		},
		"associateOID": "G3Q7JN6K3CNJJRQH ",
		"workerID": {
		"idValue": "0NHSXHM4I"
		}
		}
		}
		}
		}
		]
		}

/events/data/ eventContext	Specifies the data which sets the context for the event. That is, the keys which set context.	{
		"events": [
		{
		"data": {
		"eventContext": {
		"worker": {
		"associateOID": "ADP1234567"
		}
		}
		}
		}
		]
		}

Process Overview

When data is updated in an ADP application, the application generates an event notification message. That message is stored in the event manager database. To regularly check and retrieve event notification messages you can subscribe your application to data change events. For example, you can subscribe to the worker.hire event to be notified when a new hire is added to an ADP product, so you can do relevant work with the new hire's data.

	Actor	Description
1	A human user or a system	Makes a change to a record. For example, an employee updates a worker's email address in an ADP product.
2	ADP product	Generates an Event Notification message and sends copies to a subscriber's message queue. Messages are stored based on the first in, first out (FIFO) method.

	Actor	Description
3	Consumer application	On scheduled time, retrieves the list of Event Notifications.
4	Consumer application	Performs a callback on the first Event Notification in the list and acts on the received data
5	Consumer application	Deletes the Event Notification message from its message queue and repeats step 4 until the list is exhausted.

Chapter 2

Retrieving the list of event notifications

When an associate completes a change in an ADP system, a notification is pushed to a database for you to query. Let's assume that your application subscribes to the Worker Personal communication email Add notification. When someone adds a new personal email for a worker, it creates a notification and pushes it to your message queue. After retrieving and processing the message, you can delete it so you will not see that notification again.

API Usage

Method:	GET
URI:	/core/v1/event-notification-messages
Description:	Retrieve the list of event notifications
Note:	Refer to the specific product's ADP API data dictionary and API guide for a list of supported event notifications related to your use case.

Request header parameters

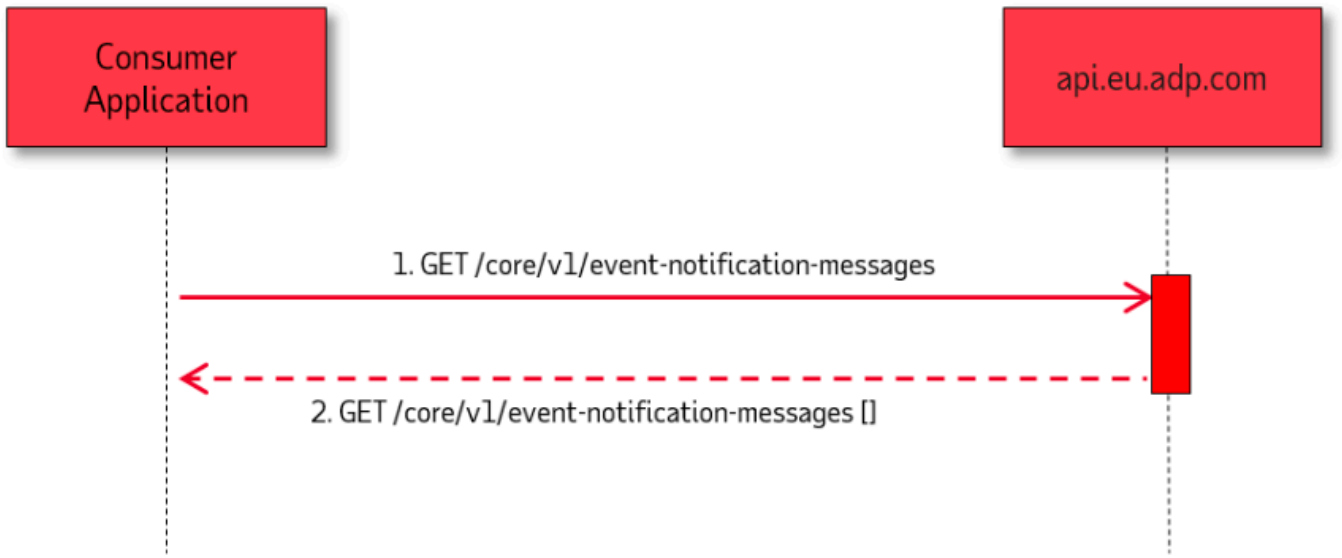
Parameter name	Authorization

Value	Bearer {token}
Required	Yes
Usage	Type of the authorization

Note

Each event notification message has a unique ID located in the event message as property eventId. It needs to be used for message deletion.

Sequence of Interactions



The following are the steps shown in the previous diagram:

1. Your consumer application makes a /core/v1/event-notification-messages request to the ADP API Endpoint for a notification.
2. The ADP API Endpoint responds with a list of event notifications to your consumer application.

Responses

Response code	Condition	Message body	Tips
200 OK	Request is processed successfully.		
200 OK	When there are no messages in the queue.	empty events array	There are no messages in the queue.
403	The request is not authorized.	empty	Contact your ADP representative to add event notifications to

Response code	Condition	Message body	Tips
Forbidden			your application scope.

Example


```
{
  "events": [
    {
      "effectiveDateTime": "2020-10-01T00:00:00Z",
      "eventID": "12345643345",
      "eventNameCode": {
        "codeValue": "worker.hire"
      },
      "links": [
        {
          "rel": "full",
          "href": "/events/hr/v1/worker.hire/5f3d1d4895d89c19d8ae3034"
        }
      ],
      "recordDateTime": "2020-09-23T12:34:00Z",
      "creationDateTime": "2020-08-19T12:38:32.035Z",
      "originator": {
        "associateOID": "ADPNL12345667",
        "formattedName": "Will Jones"
      }
    }
  ]
}
```


Important

There's a 30 days retention limit on the events in the queue. Events older than 30 days, that are not processed or deleted from the queue, are removed from the database and will not appear in the event list.

Please be aware individual ADP HCM systems may differ from this guideline, so please also check your ADP HCM system's API guides.

Chapter 3

Getting an event notification

This step loops through the list of event notifications received during the GET Event Notifications. Each event notification in this list has a callback URI, that can be used to get the data of the event notification.

API usage

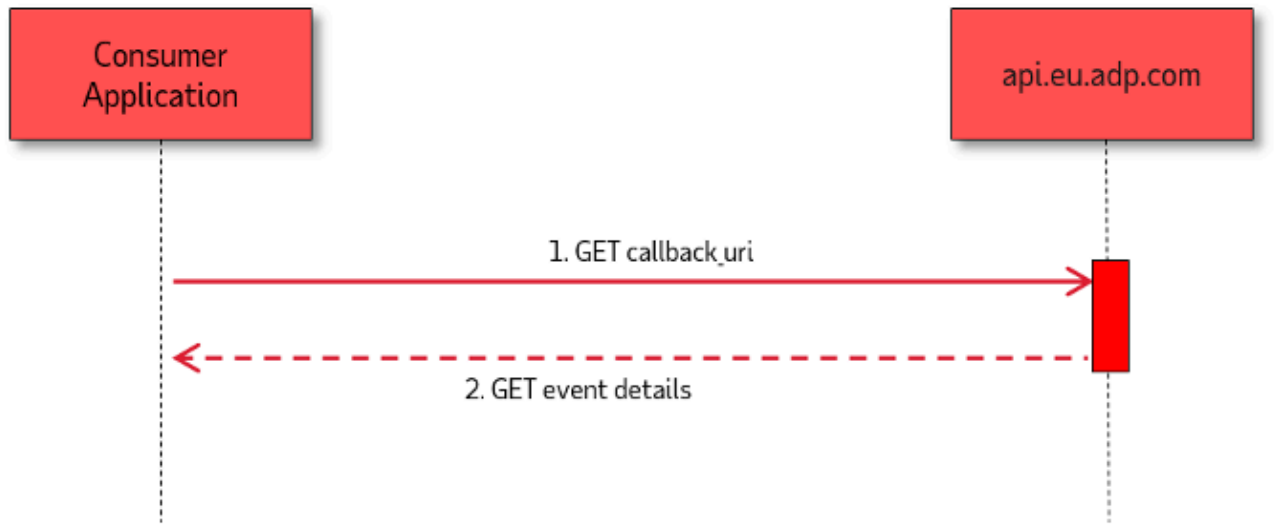
Method	GET
URI	callback_uri
Description	Get event notification details

Request header parameters

Parameter name	Authorization

Value	Berarer {token}
Required	Yes
Usage	Type of the authorization

Sequence of interactions



The following are the steps shown in the previous diagram:

1. Your consumer application makes a callback_uri request to the ADP API Endpoint for the event details
2. The ADP API Endpoint responds to your consumer application with the events details.

Responses

Response code	Condition	Body	Tips to handle
200 OK	Request is processed successfully.		
404 Not Found	When invalid callback_uri is passed in the request, or event is deleted in the sor	Depends on the system of records, but something like 'the callback doesn't exist'.	Make sure a valid callback_uri is passed in the request.
403 Unauthorized	The api user is not authorized to access the data	Depends on sor	delete the event

Note:

When you get a 403 Unauthorized response, you can safely assume the event is not related to the worker population that was assigned to you in the system of record. In the current state, we can filter the worker output, but not the event generation. Therefore, you can get events that are not related to your population. You can just delete the event with a DELETE request and ignore any further processing.

Example

```
{
  "events": [
    {
      "actor": {
        "applicationID": {
          "idValue": "iHCM"
        },
        "associateOID": "DHARRISON_088",
        "deviceID": "iHCM",
        "formattedName": "Daniel Harrison",
        "geoCoordinate": {
          "latitude": 0.0,
          "longitude": 0.0
        }
      },
      "creationDateTime": "2022-03-08T13:47:27Z",
      "data": {
        "eventContext": {
          "associateOID": "DHARRISON_088"
        },
        "output": {
          "worker": {
            "associateOID": "DHARRISON_088",
            "workAssignment": {
              "assignedOrganizationalUnits": [
                {
                  "nameCode": {
                    "codeValue": "ADLMAIN",
                    "shortName": "Alliance Demo Limited"
                  },
                  "typeCode": {
                    "codeValue": "0",
                    "longName": "Organisational",
                    "shortName": "Company"
                  }
                }
              ],
              "assignmentCostCenters": [
                {
                  "costCenterID": "105",
                  "costCenterName": "Human Resources"
                },
                {
                  "costCenterID": "101",
                  "costCenterName": "Sales",
                  "costCenterPercentage": 100.0
                }
              ],
              "assignmentStatus": {
                "statusCode": {
                  "codeValue": "Current",
                  "shortName": "Current"
                }
              },
              "fullTimeEquivalenceRatio": 1.0,
              "hireDate": "2013-01-01",
              "homeWorkLocation": {
                "nameCode": {
                  "codeValue": "MAN01",
                  "shortName": "Manchester"
                }
              },
              "jobCode": {
                "codeValue": "021",
                "shortName": "UAT Manager"
              },
              "jobTitle": "UAT Manager",
              "legalEntityID": "ADLM001",
              "payrollFileNumber": "001",
              "primaryIndicator": true,
              "reportsTo": [
                {
                  "itemID": "e7bb9232-86f4-4195-90ec-4f44af9beb79",
                  "reportsToRelationshipCode": {
                    "codeValue": "REPORTSTO",
                    "shortName": "REPORTSTO"
                  },
                  "reportsToWorkerName": {
                    "formattedName": "Belinda Newton"
                  },
                  "workerID": {
                    "idValue": "ADL001"
                  }
                },
                {
                  "itemID": "e7bb9232-86f4-4195-90ec-4f44af9beb79",
                  "reportsToRelationshipCode": {
                    "codeValue": "SUPERVISOR",
                    "shortName": "SUPERVISOR"
                  },
                  "reportsToWorkerName": {
```

```
                        "formattedName": "Belinda Newton"
                     },
                     "workerID": {
                        "idValue": "ADL001"
                     }
                  }
               ],
               "standardHours": {
                  "hoursQuantity": 37.5,
                  "unitCode": {
                     "codeValue": "F",
                     "shortName": "Fixed"
                  }
               },
               "workerTypeCode": {
                  "codeValue": "Permanent",
                  "shortName": "Permanent"
               },
               "workLevelCode": {
                  "codeValue": "Full_Time",
                  "shortName": "Full Time"
               }
            },
            "workerID": {
               "idValue": "ADL007"
            }
         }
      },
      "transform": {
         "effectiveDateTime": "2013-01-01T00:00:00",
         "eventStatusCode": {
            "codeValue": "Complete"
         },
         "workAssignment": {
            "assignedOrganizationalUnits": [
               {
                  "nameCode": {
                     "codeValue": "ADLMAIN",
                     "shortName": "Alliance Demo Limited"
                  },
                  "typeCode": {
                     "codeValue": "0",
                     "longName": "Organisational",
                     "shortName": "Company"
                  }
               }
            ],
            "jobCode": {
               "codeValue": "021",
               "shortName": "UAT Manager"
            },
            "jobTitle": "UAT Manager",
            "legalEntityID": "ADLM001"
         }
      }
   },
   "effectiveDateTime": "2022-03-08T13:47:27Z",
   "eventID": "e06df106-e995-4316-934a-3c702e15f862",
   "eventNameCode": {
      "codeValue": "worker.workAssignment.modify",
      "shortName": "worker.workAssignment.modify"
   },
   "eventStatusCode": {
      "codeValue": "Complete"
   },
   "eventTitle": "Change Work Assignment",
   "originator": {
      "applicationID": {
         "idValue": "iHCM"
      },
      "associateOID": "DHARRISON_088",
      "deviceID": "iHCM",
      "eventID": "e06df106-e995-4316-934a-3c702e15f862",
      "eventNameCode": {
         "codeValue": "worker.workAssignment.modify",
         "longName": "worker.workAssignment.modify",
         "shortName": "worker.workAssignment.modify"
      },
      "formattedName": "Daniel Harrison",
      "personOID": "DHARRISON_088"
   },
   "recordDateTime": "2022-03-08T13:47:22Z",
   "serviceCategoryCode": {
      "codeValue": "HR",
      "shortName": "HR"
   }
  }
 ]
}
```

Chapter 4

Deleting an event notification

Delete an event notification

API usage

Method	**DELETE**
URI	/core/v1/event-notification-messages/{eventID}
Description	Delete an event notification from the event notification list.

Note

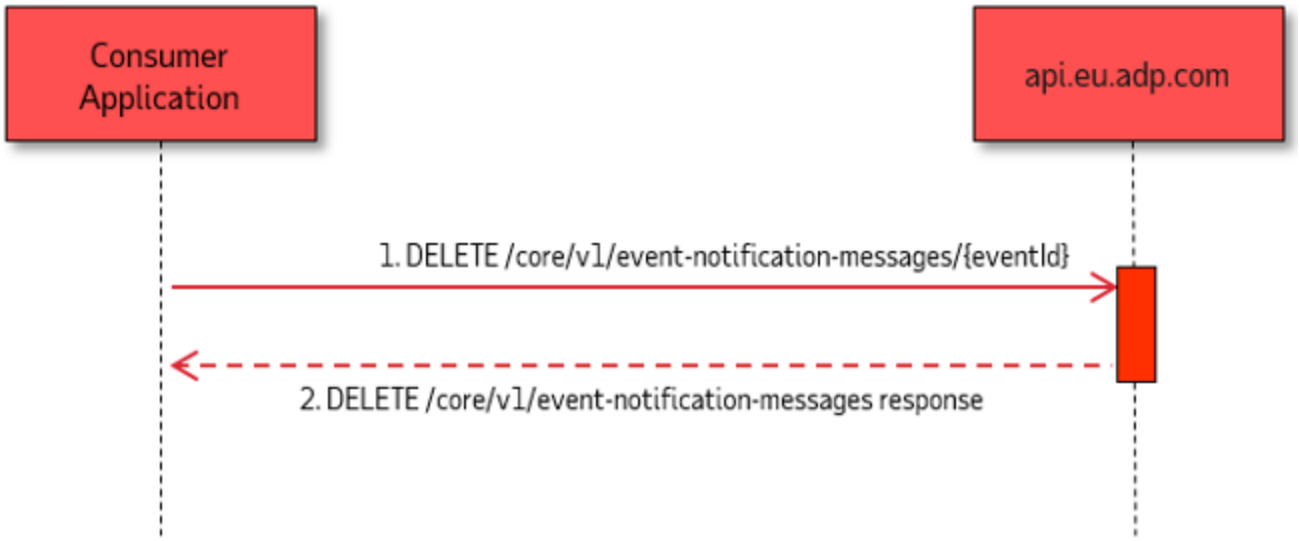
URI parameter **'eventID'** is returned from the list of Event Notifications and the event details.

For example: **"eventID" : "bdccd3a2-c722-449b-be13-f847e43d9c9f"**

Request header parameters

Parameter name	**Authorization**
Value	Bearer {token}
Required	Yes
Usage	Type of the authorization

Sequence of interactions



The following are the steps shown in the previous diagram:

1. Your consumer application makes a /core/v1/event-notification-messages/{eventId} request to the ADP API Endpoint for a notification.
2. The ADP API Endpoint responds to your consumer application with the events deleted.

Responses

Response code	Condition	Message Text	Tips to Handle
200 OK	Request is processed successfully.		
404 Not Found	When invalid eventID is passed in the request.	empty body	The event notification with that eventID cannot be found. Make sure a valid {eventId} is passed in the request.

Chapter 5

Implementation guidelines

What if you decide to use the ADP event mechanism to consume the events published by ADP. It will involves software development work and we like to present a of template with questions which has to be addressed for such an implementation.

Choose and validate events

Start with choosing the events you want to subscribe to. For instance:

- You want a new.hire event from ADP to result in a new user to be created in the facility management system so he can login and create requests
- You want the timeOffRequest.submit and timeOffRequest.cancel events to be processed in project management system so that days will be blocked/unblocked for project work
- You want an organizationCostCenter.add and organizationCostCenter.change to be processed in the financial system so the general ledger file with the employee costs can be imported without errors

For each user story you need to investigate the effort on processing the event in the destination system, ask yourself questions like:

- Which data is expected, is that available in the event message? If not: contact ADP for alternative solutions
- How about the mapping of unique keys? ADP will not store and manage keys of destination systems, best solution is reuse the ADP identifiers (like the employee id) or choose something which both systems have in common and is unique (for instance the email address).

- What capabilities does the destination system has in updating information from an external source? Does it provide API's? Is updating on database level an option?
- Which validations are implemented in the destination system? Do you need to map values?
- What if the destination system cannot process the data? This is a specific subject which is addressed in a separate paragraph of this document.

Ability to handle effective dated events

Events are effective dated. This means that event notifications (more accurately, the data in the event message) become valid in the future. You can also receive event notifications which impact changes in the past (back dated). To see on what date a change becomes effective, each event contains an effectiveDateTime field.

If you have a system which can only hold the current state (no historical data possible) we advise you to use the request-response API's. For instance: retrieve the actual status of a worker using the ADP Worker API when a user enters a dialog in the other application. Or use scheduled synchronizations with ETL or the API's.

It is not advised to buffer events locally and process them when they become effective. You will need to implement complex businesses logic to handle an example like:

- On January 10th the HR practitioner enters a new hire which will join the company on February 1st
- On January 11th the HR practitioner does change the hire date, it is set to January 15th, the employee can start earlier
- On January 15th the receiving system will receive a worker.change event with the new hire date and cannot process this event because the employee is not created

Our advice is to only use the event model if the receiving system is capable of handling effective dated changes.

Implement retrieving event notifications and event details

As described above, to consume event notifications you will have to implement five steps:

1. Posting a request to retrieve an OAuth2 token using your certificate and username/password
2. Requesting the list of event notification messages
3. For each notification message retrieve the event detail using the call back uri which is added in the notification message list
4. Process the event in your own system
5. Send a message the event is processed successfully so it will not be sent again by ADP

This is the "happy flow" and it assumes you will implement it for each IT system you want to update. ADP can create multiple API consumers per data owner each with their own subscriptions.
When you have large volumes of events (>1000 per day) or you want to update multiple internal systems we would advise you to implement a flow which retrieves all notification messages from ADP, store it in a local queuing system and distribute from there. This is typically a task for an enterprise service bus (ESB). It decouples the flow for retrieving the messages and processing the messages. The ESB will take care of sending right messages to the IT systems (subscriptions).

What if the event cannot be processed in the client system?

You should implement a process to handle the events which cannot be processed automatically. For instance, referential integrity errors: department code registered at employee level does not exist. The receiving system refuses to consume the event because some data is missing. All use cases should be described and anticipated upon. Can you create the department "on-the-fly", can you use default values? And what if a new event for that employee arrives, should you process it if you can? You need to build tooling for application managers to monitor the event processing with the possibility to intervene manually. If you use the standard implementation (not queuing the messages yourself) you should be aware that not handling the stuck events can stop the data synchronization process. ADP only returns the top 1000 and if those 1000 events are waiting to be handled manually no new event notifications will be delivered.

Chapter 6

Terminology and Acronyms

API	**(Application Programming Interface) A system of tools and resources in an operating system created by ADP, enabling developers to create software applications.**
Canonical ID	Unique IDs that ADP assigns to API end points to manage API access and authorization. Application developers need to request API access using canonical IDs.

CSR	(Certificate Signing Request) Required for accessing ADP APIs and authenticating users with SSO.
OAuth	(Open Authorization) Framework for token-based authentication and authorization for web-based applications.
Private Key	A text file used initially to generate a CSR, and later to secure and verify connections using the certificate created per that request. The client is the owner for this key and the server is not aware of the key.
Public Key	Distributed by the vendor and is included as part of the SSL certificate. It works together with a private key to make sure that data is encrypted, not tampered with, and verified.
SOR	(System of Records) Refers to the ADP HCM solution.
SSL	(Secure Sockets Layer) A global standard security technology that enables encrypted communication between a web browser and a web server.
SSO	(Single Sign-On) Authentication process that allows a user to access multiple applications with one set of login credentials.