Deltek

Deltek WorkBook

Webhooks Guide

May 30, 2024

While Deltek has attempted to verify that the information in this document is accurate and complete, some typographical or technical errors may exist. The recipient of this document is solely responsible for all decisions relating to or use of the information provided herein.

The information contained in this publication is effective as of the publication date below and is subject to change without notice.

This edition published May 2024.

Contents

Important Information

Before you begin, you should understand the following:

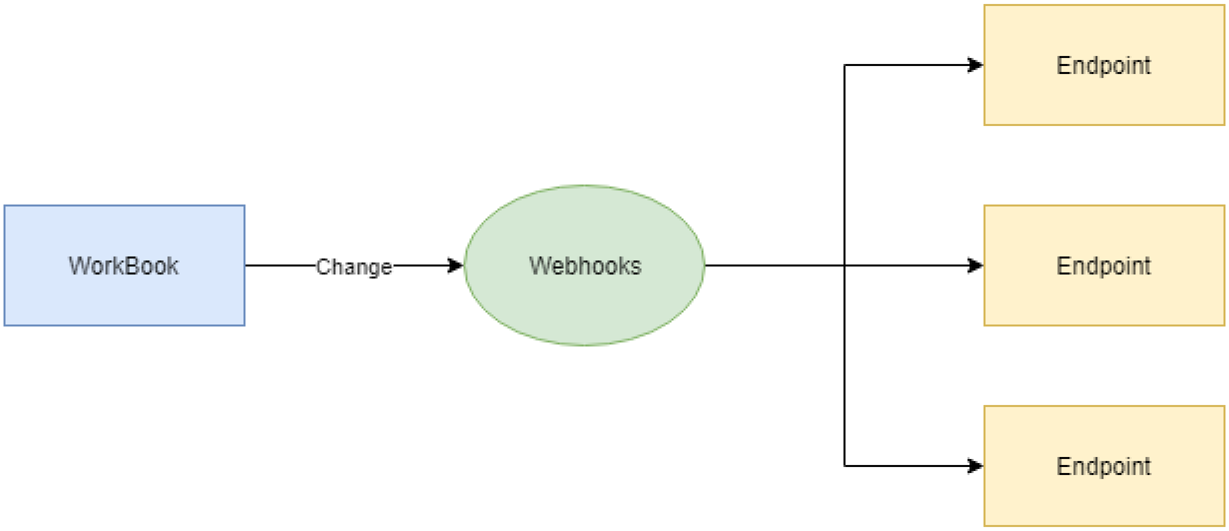
- Your system should **not** rely 100% on webhooks.
- Some events might **not** be triggered at all.
- Agents do not trigger events.
- It is recommended that your endpoints be publicly open.
- Information that is sent by the webhooks does not and should not contain sensitive information. Generally information consists of Event, ID, and Type.
- Results are not cleared automatically.
- This guide is addressed mainly to developers.

Important Terms

Term	Description
Endpoint	One end of a communication channel. This is the place where the HTTP request from WorkBook should be sent to notify you about a change. That looks like a URL, for example, www.yoursite.com/api/myendpoint. The application should have a request listener on that path to receive the information sent from WorkBook and process it according to needs.
Public Key	Used to encrypt the data inside so that the information is cyphered. If someone intercepts it, they cannot read it.
Request	The form in which information is transported from WorkBook to an Endpoint. This is an HTTP Request.
Secret Phrase	Used to prevent someone from changing information via attacks (integrity) and to validate that the information is sent by the webhook that you set up (authenticity).

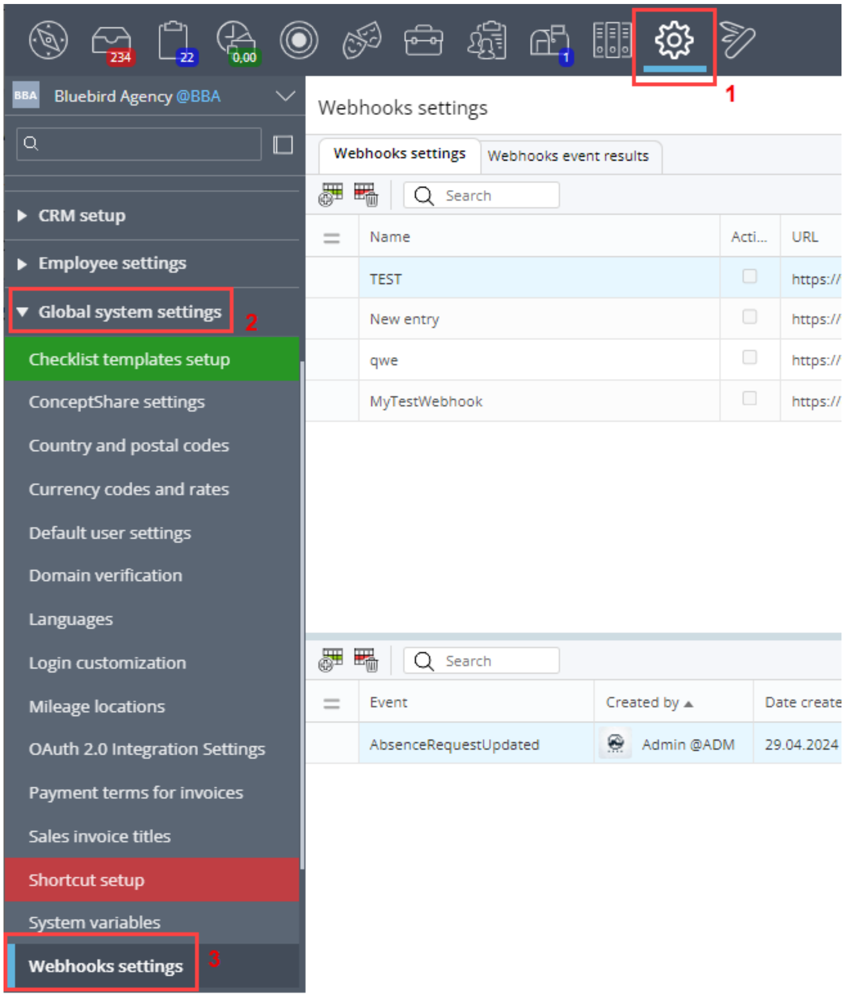
Introduction

You use webhooks to notify one or more endpoints about a change that took place in WorkBook.



Getting Started

You navigate to webhooks via WorkBook » Settings » Global System Settings » Webhooks Settings, as the following figure shows.



Name	Acti...	URL
TEST	□	https://
New entry	□	https://
qwe	□	https://
MyTestWebhook	□	https://

Event	Created by ▲	Date create
AbsenceRequestUpdated	Admin @ADM	29.04.2024

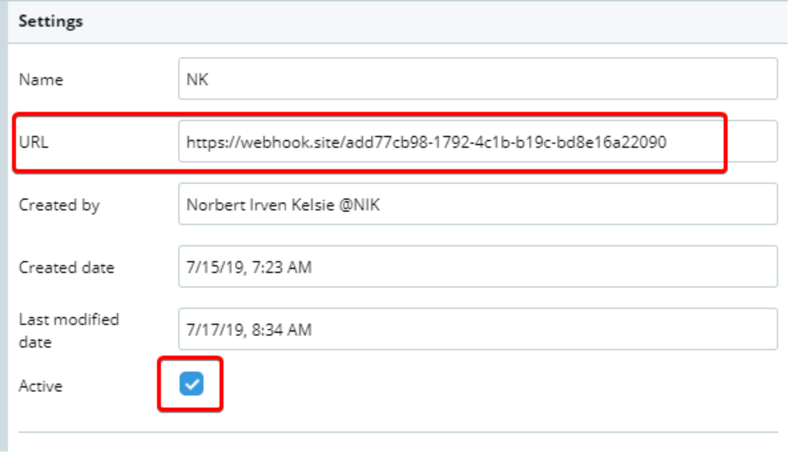
Create a Webhook

To create a webhook:

1. Navigate to the Webhooks Settings page via **Settings » Global System Settings » Webhooks Settings**.
2. On the toolbar, click **Add New Webhook**.
3. In the panes at the right side of the page, enter or change the details for the webhook. (The Webhooks page grid is read-only.)

   When you first create a webhook, it is **not** active by default. Before you can activate the webhook, you must provide the URL for the endpoint that is notified when an event occurs in WorkBook.

4. Enter the URL for the endpoint in the **URL** field and select the **Active** check box to activate the webhook.



Settings

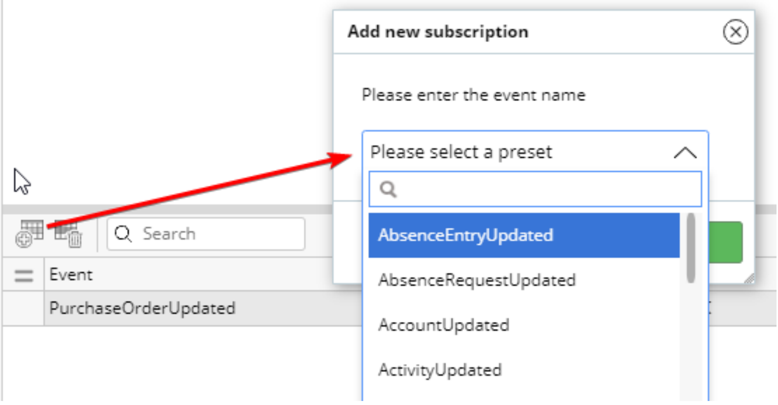
Name	NK
URL	https://webhook.site/add77cb98-1792-4c1b-b19c-bd8e16a22090
Created by	Norbert Irven Kelsie @NIK
Created date	7/15/19, 7:23 AM
Last modified date	7/17/19, 8:34 AM
Active	☑

Subscribe to Events

After you set up a webhook, you can subscribe to various events so that you are notified when they occur.

To subscribe to events:

1. Click **Add New Subscription** in the bottom grid. The Add New Subscription dialog displays a list of events to which you can subscribe.



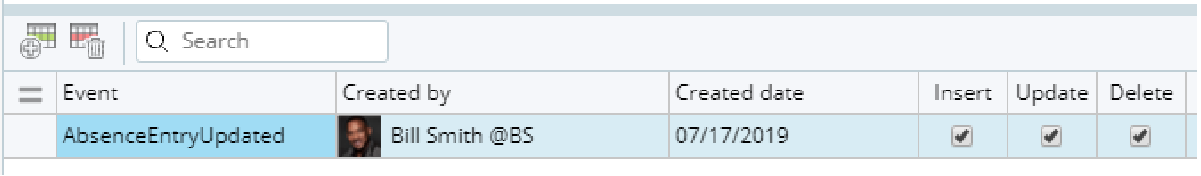
Add new subscription

Please enter the event name

Please select a preset

2. Choose the event for which you want to be notified on the endpoint URL that you entered when you created the webhook.

For example, if you choose **AbsenceEntryUpdated**, that event name appears in the grid as shown in the following figure.



Event	Created by	Created date	Insert	Update	Delete
AbsenceEntryUpdated	Bill Smith @BS	07/17/2019	☑	☑	☑

Tip: Almost every change in WorkBook triggers a new event, which is then sent to a subscription. This can result in many events being delivered to a subscription. You can use event aggregation to reduce the number of calls to a webhook, which in turn reduces the number of events that are delivered to a subscription. See Use Event Aggregation for information about using event aggregation to reduce the number of events that are delivered to a subscription.

Event Types

You can also subscribe to different Event Types (Insert/Update/Delete).

Note: Some events might not trigger some of the types. For example, JobKeyFiguresUpdated only triggers Update; it never triggers Insert/Delete.

By default, all of the Event Types are enabled.

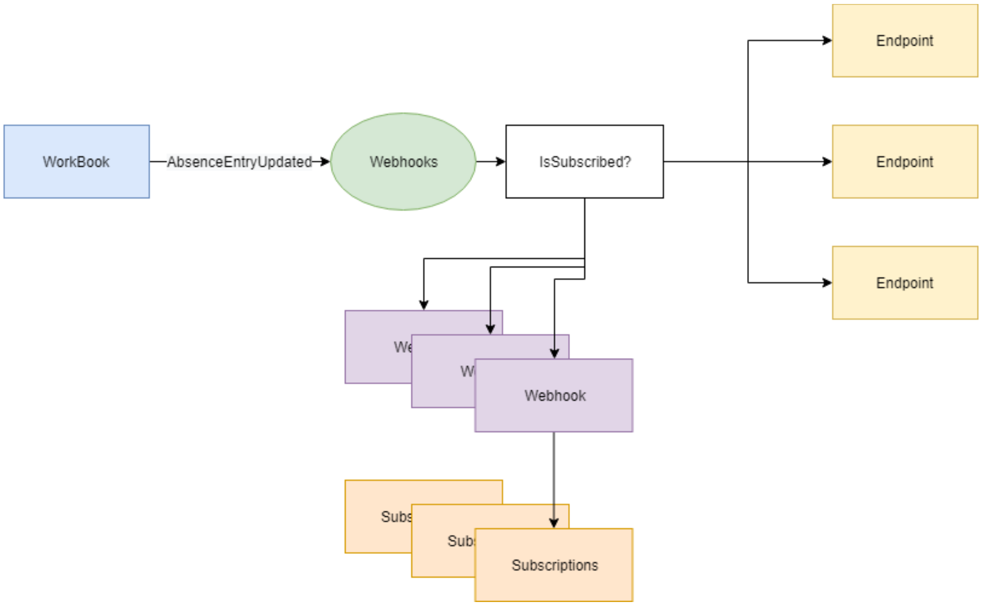
In the example of subscribing to **AbsenceEntryUpdated** and selecting all of the check boxes, you will be notified on your endpoint by the webhooks when:

- An absence entry was INSERTED (created)
- An absence entry was UPDATED
- An absence entry was DELETED

If, for example, you deselect the **Delete** check box, you will not be notified when an absence entry is deleted.

Attention: If you subscribe to the same event twice, you are notified twice when that event occurs.

The following diagram includes the subscriptions as well.



When an AbsenceEntryUpdated event occurs in WorkBook, a check runs through all of the webhooks and subscriptions to see if any of them are subscribed to that event. If any are, the endpoint(s) are notified that the event has occurred.

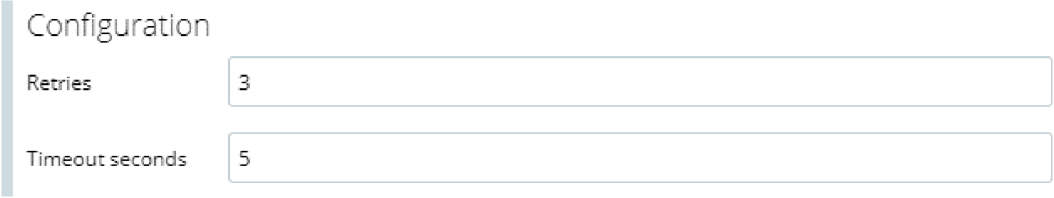
Configuration

You configure the number of retries and the timeout duration:

- **Retries** – This is the number of times that WorkBook retries to notify an endpoint that an event occurred if a webhook tries to do a notification and that notification attempt fails. These retries happen consecutively. WorkBook attempts to perform the notification for this number of times before it creates the event result.
- **Timeout** – This is the maximum length of time (in seconds) that WorkBook should wait after trying to notify an endpoint before it considers the attempt to have failed.

To configure the webhook retries and timeout duration:

1. Navigate to **Settings » Global System Settings » Webhooks Settings » Configuration pane** (at the right side of the Webhook Settings page).
2. Enter the number of retries in the **Retries** field. The default value is 3. The value must be within the range 1-10.
3. Enter the timeout duration in the **Timeout Seconds** field. The default value is 5. The value must be within the range 1-10.



Configuration

Retries: 3

Timeout seconds: 5

Security

You must perform the following to set up security for your webhooks. You can use a secret phrase and/or a public key (RSA Key) to set up security.

Secret Phrase

Webhooks use an HMAC signature to verify the integrity and authenticity of the message. The secret phrase that you enter is used as the key for creating the signature.

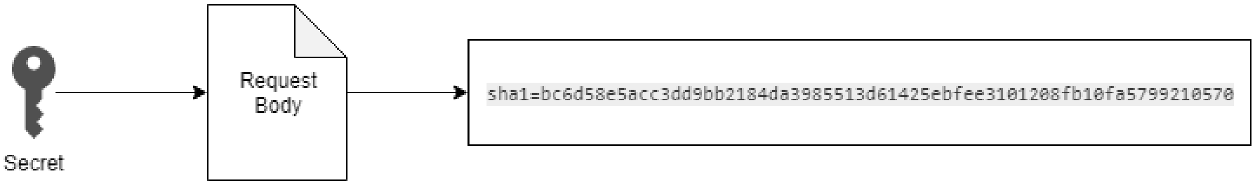
If you provide a secret phrase, WorkBook generates an HMAC signature of the payload and includes that signature in the request headers. The X-Hub-Signature header's value is in the form

```
sha1=signature
```

where `signature` is a 40-byte, hexadecimal representation of a SHA256 signature. The signature **must** be computed using the HMAC algorithm, with the request body as the data and the secret phrase as the key.

The hash signature starts with **sha1=** followed by the computed hash of your secret token and the payload body.

When the client application validates the signature, using a plain **==** operator is not advised. A method like **secure compare** performs a constant-time string comparison, which renders it safe from certain timing attacks against regular equality operators.


Secret
Request Body
sha1=bc6d58e5acc3dd9bb2184da3985513d61425ebfee3101208fb10fa5799210570

This hash signature is included in the request headers as **x-hub-signature**.

The following is an example of how you should validate the signature in your application if you have set a Secret Phrase in WorkBook.

Example

```
if (ValidateSignature(Request.GetRawBody(), Request.Headers["x-hub-signature"]))
{
   DoWork();
}

private bool ValidateSignature(string body, string signature)
{
   string secret = LoadYourSecret(); // This is the secret token you set in WorkBook.
   string expectedSignature = CreateSignature(body, secret);

   return expectedSignature.Equals(signature, StringComparison.OrdinalIgnoreCase);
}

private string CreateSignature(string body, string secret)
{
   var encoding = new UTF8Encoding();
   var bodyByte = encoding.GetBytes(body);
   var secretByte = encoding.GetBytes(secret);

   using (var HMAC = new HMACSHA256(secretByte))
   {
      var signature = HMAC.ComputeHash(bodyByte);
      var hash = "sha1={0}".Replace("{0}", BitConverter.ToString(signature).Replace("-", string.Empty));
      return hash;
   }
}
```

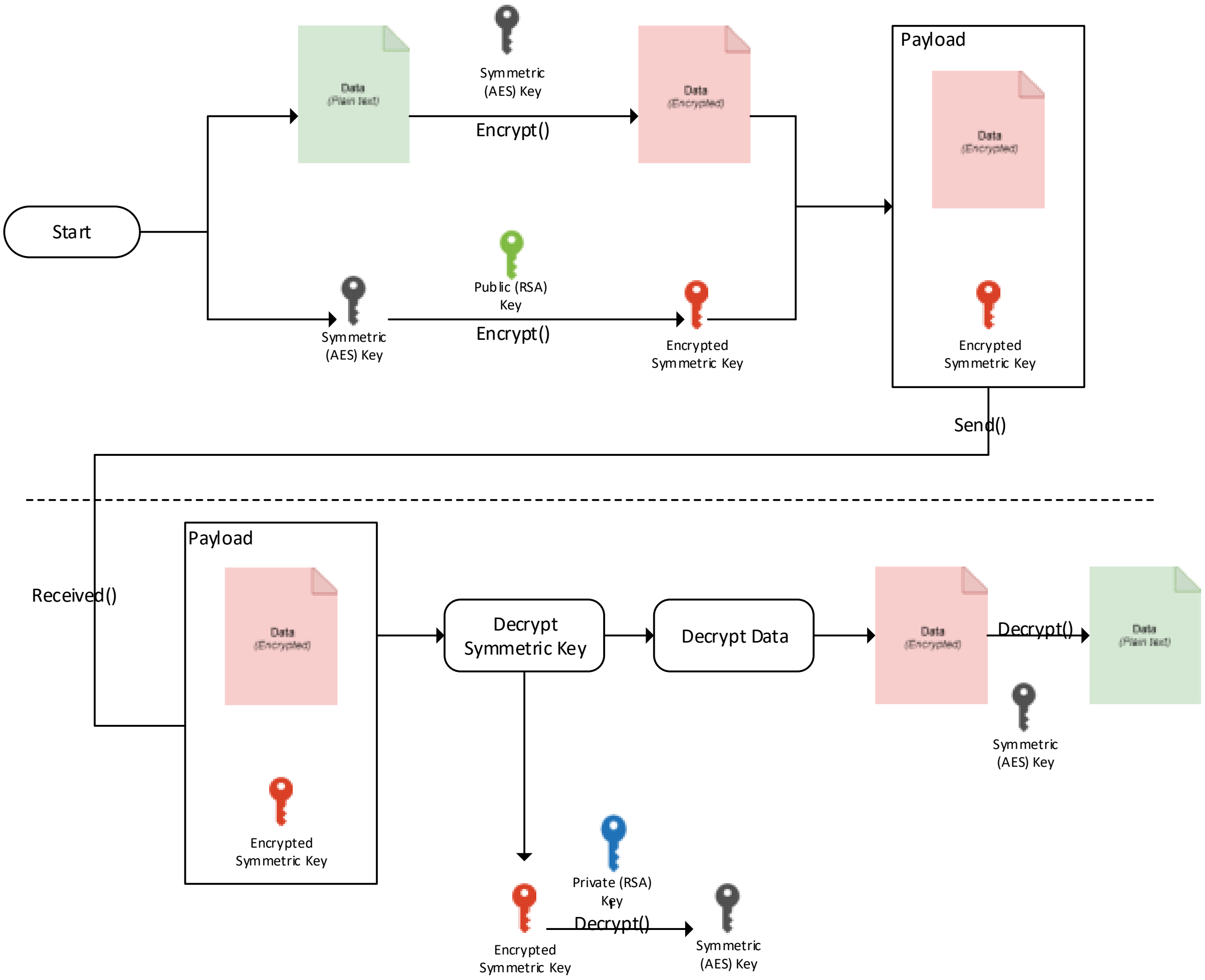
Public Key

- The public key is an **RSA Key**.
- The symmetric key encrypted is an **AES Key**.

The following is an example of RSA key format (XML format):

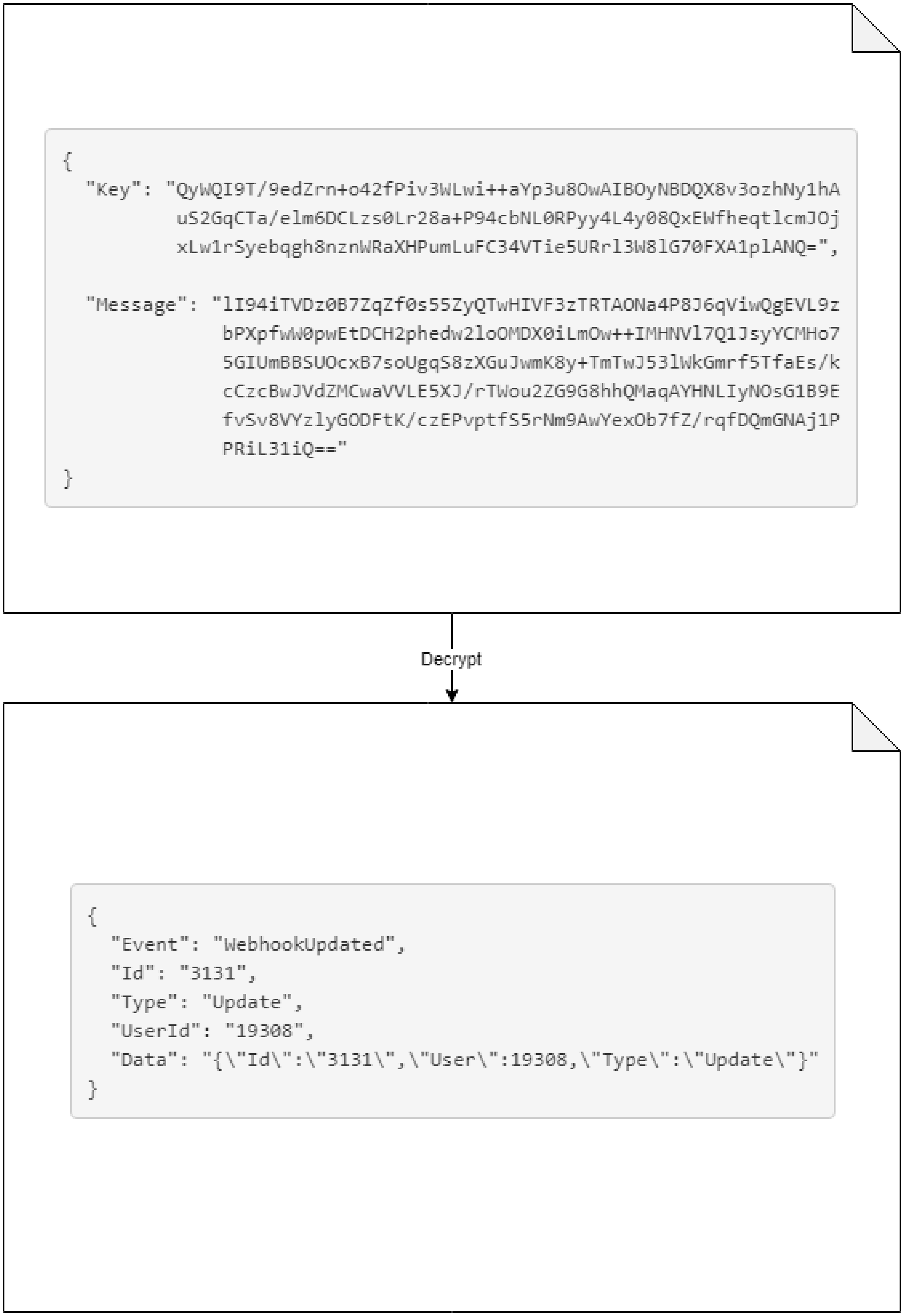
```
<RSAKeyValue><Modulus>tcVzwKITirQzeRfpenD32TfgKRyFmRZUbe8pmbpK5QiT1IgOicIDqSbwfaBjJ
l8vklT3D2VNfyTjnBIC2l6ssq4P506uIbSLtSwWwA4qv++5pSLxCIfjiePk/KpMPMEtT4VQ1Ua4kaCXoEHJ
O4on6vNnqQytqNvm1eMTKQfr7C0=</Modulus><Exponent>AQAB</Exponent></RSAKeyValue>
```

How Data is Encrypted



As illustrated by the preceding diagram:

- An AES Key is generated in WorkBook, and the data is encrypted with this key. Next, WorkBook encrypts the generated AES Key with the Public Key that you entered and sends it in the payload.
- When you receive the request, with one of your Private Keys you must decrypt the AES Key first and then use the AES Key to decrypt the message. The following is an example of how the encrypted payload looks when it is received, and how it looks after it is decrypted.



```
{
  "Key": "QyWQI9T/9edZrn+o42fPiv3WLwi++aYp3u8OwAIBOyNBDQX8v3ozhNy1hA
          uS2GqCTa/elm6DCLzs0Lr28a+P94cbNL0RPyy4L4y08QxEWfheqtlcmJOj
          xLw1rSyebqgh8nznWRaXHPumLuFC34VTie5URrl3W8lG70FXA1plANQ=",

  "Message": "lI94iTVDz0B7ZqZf0s55ZyQTwHIVF3zTRTAONa4P8J6qViwQgEVL9z
              bPXpfwW0pwEtDCH2phedw2loOMDX0iLmOw++IMHNVl7Q1JsyYCMHo7
              5GIUmBBSUOcxB7soUgqS8zXGuJwmK8y+TmTwJ53lWkGmrf5TfaEs/k
              cCzcBwJVdZMCwaVVLE5XJ/rTWou2ZG9G8hhQMaqAYHNLIyNOsG1B9E
              fvSv8VYzlyGODFtK/czEPvptfS5rNm9AwYexOb7fZ/rqfDQmGNAj1P
              PRiL31iQ=="
}
```

```
{
  "Event": "WebhookUpdated",
  "Id": "3131",
  "Type": "Update",
  "UserId": "19308",
  "Data": "{\"Id\":\"3131\",\"User\":19308,\"Type\":\"Update\"}"
}
```

The following is an example of how to decrypt the payload when you have encrypted it using a Public Key.

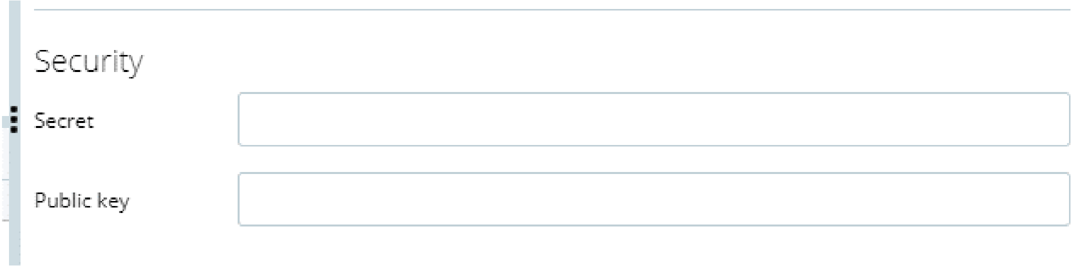
Example

```
var decryptedMessage = DecryptAll(Request.GetRawBody());

private string DecryptAll(string rawBody)
{
   Body body = JsonConvert.DeserializeObject<Body>(rawBody);
   string privateKey = LoadYourPrivateKey();

   AESKey aesKey = JsonConvert.DeserializeObject<AESKey>(DecryptRSA(body.Key, privateKey));
   var decryptedMessage = DecryptAES(body.Message, aesKey.Key, aesKey.Iv);

   return decryptedMessage;
}

private string DecryptRSA(string text, string privateKey)
{
   byte[] encryptedBytes = Convert.FromBase64String(text);
   string decryptedData = null;

   using (RSACryptoServiceProvider RSA = new RSACryptoServiceProvider(2048))
   {
      RSA.FromXmlString(privateKey);
      byte[] bytes = RSA.Decrypt(encryptedBytes, true);
      decryptedData = Encoding.UTF8.GetString(bytes);
   }
   return decryptedData;
}

private string DecryptAES(string text, byte[] Key, byte[] Iv)
{
   byte[] encryptedBytes = Convert.FromBase64String(text);
   string decryptedData = null;

   using (AesManaged AES = new AesManaged())
   {
      ICryptoTransform decryptor = AES.CreateDecryptor(Key, Iv);
      using (MemoryStream ms = new MemoryStream(encryptedBytes))
      {
         using (CryptoStream cs = new CryptoStream(ms, decryptor, CryptoStreamMode.Read))
         {
            using (StreamReader reader = new StreamReader(cs))
            {
               decryptedData = reader.ReadToEnd();
            }
         }
      }
   }
   return decryptedData;
}

public class Body
{
   public string Key { get; set; }
   public string Message { get; set; }
}

public class AESKey
{
   public byte[] Key { get; set; }
   public byte[] Iv { get; set; }
}
```

Set Up Security

To set up security:

1. Navigate to **Settings » Global System Settings » Webhooks Settings » Security pane** (at the right side of the Webhooks Settings page).
2. Enter a string in the **Secret** field. The string that you enter here is used as the key to create the HMAC signature (see Security.)
3. Enter the RSA Public Key (XML format) in the **Public Key** field.

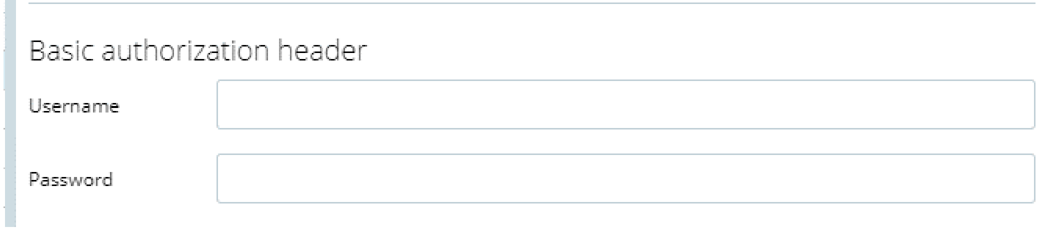


Define a Basic Authorization Header

If your endpoint requires authentication you must provide a Username and Password to allow the webhook to notify the endpoint with authentication.

To define a basic authorization header:

1. Navigate to **Settings » Global System Settings » Webhooks Settings » Basic Authorization Header pane** (at the right side of the Webhooks Settings page).
2. Enter the username for authentication in the **Username** field.
3. Enter the password for this username in the **Password** field.



Basic authorization header

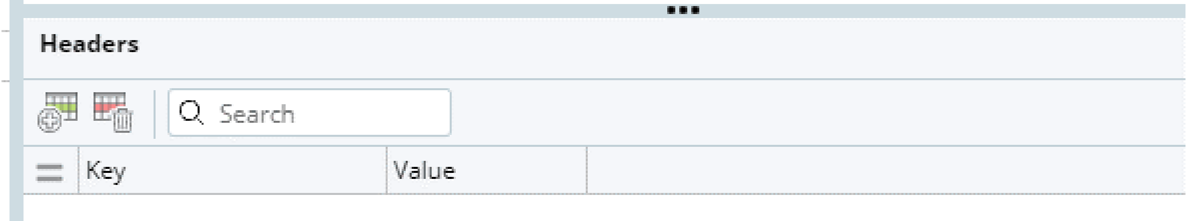
Attention: The user credentials that you provide here should be those of an API user—not a physical one—that has minimum access in your application, preferably with access only to the request listener that you have set up in your application.

Define a Custom Header

If you need to, you can add any custom header to the request that is being sent when the endpoint is notified.

To add a custom header to the request:

1. Navigate to **Settings » Global System Settings » Webhooks Settings » Headers pane** (at the right side of the Webhooks Settings page).



2. Add the key and value of the header, separated by a colon, in the format **key:value**.

   For example, to add a custom header with the key SendingFrom and the value WorkBook, enter the following:

   SendingFrom:WorkBook

You can add multiple custom headers.

Receiving the Request on the Endpoint

The following is an example of how a request looks when an absence entry is deleted, and the webhook is subscribed to AbsenceEntryUpdated.

Headers

```
connection                     close
x-forwarded-for                193.17.206.4
expect                         100-continue
content-length                 134
host                           webhook.site
content-type                   application/json
x-webhook-eventcreateddateutc  636990436599270000
x-webhook-event                AbsenceEntryUpdated
accept-encoding                gzip,deflate
user-agent                     ServiceStack .NET Client 5.50
accept                         application/json
```

Body

```
{
  "Event": "AbsenceEntryUpdated",
  "Id": "5572",
  "Type": "Delete",
  "UserId": "19308",
  "Data":"{\"Id\":5572,\"User\":19308,\"Type\":\"Delete\"}",
  "ItemCount": 1
}
```

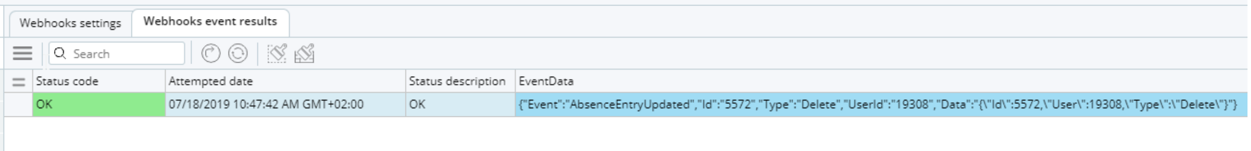
Note: You can see in the headers **x-webhook-eventcreateddateutc,** which is the value in ticks of the date and time (UTC) when the event took place in WorkBook. You might check that when you receive the request, because for various reasons you might receive it later than the time when it actually happened (because of possible timeouts, lost connections, connectivity issues, lag, and so on).

If you have added custom headers, those should appear there too.

As a general rule, the body should be a JSON type object that should contain:

- **Event** – The event name of the event that you subscribed to
- **ID** – The ID of the entry
- **Type** – The event type, such as Insert, Update, Delete
- **UserId** – The ID of the WorkBook user who triggered the event
- **Data** – The raw data, which might contain more information like *StatusId, PlanId, JobId, Taskid* where these might be relevant
- **ItemCount** – This property shows a value greater than 1 if you are using event aggregation. For example, if ItemCount = 3, it represents three events that have the same type and are for the same entity. See Use Event Aggregation for information about using event aggregation.

Event Results



Status code	Attempted date	Status description	EventData
OK	07/18/2019 10:47:42 AM GMT+02:00	OK	{"Event":"AbsenceEntryUpdated","Id":"5572","Type":"Delete","UserId":"19308","Data":"{\"Id\":5572,\"User\":19308,\"Type\":\"Delete\"}"}

After an event that has an active webhook with a subscription on it is triggered, and the endpoint is notified, you can see the result in a list on the Webhook Event Results page.

A result contains the Status Code, Attempted Date (UTC), Status Description, and the event data that was sent to the endpoint.

Tip: If the data was encrypted using a public key, the key is displayed here in plain text.

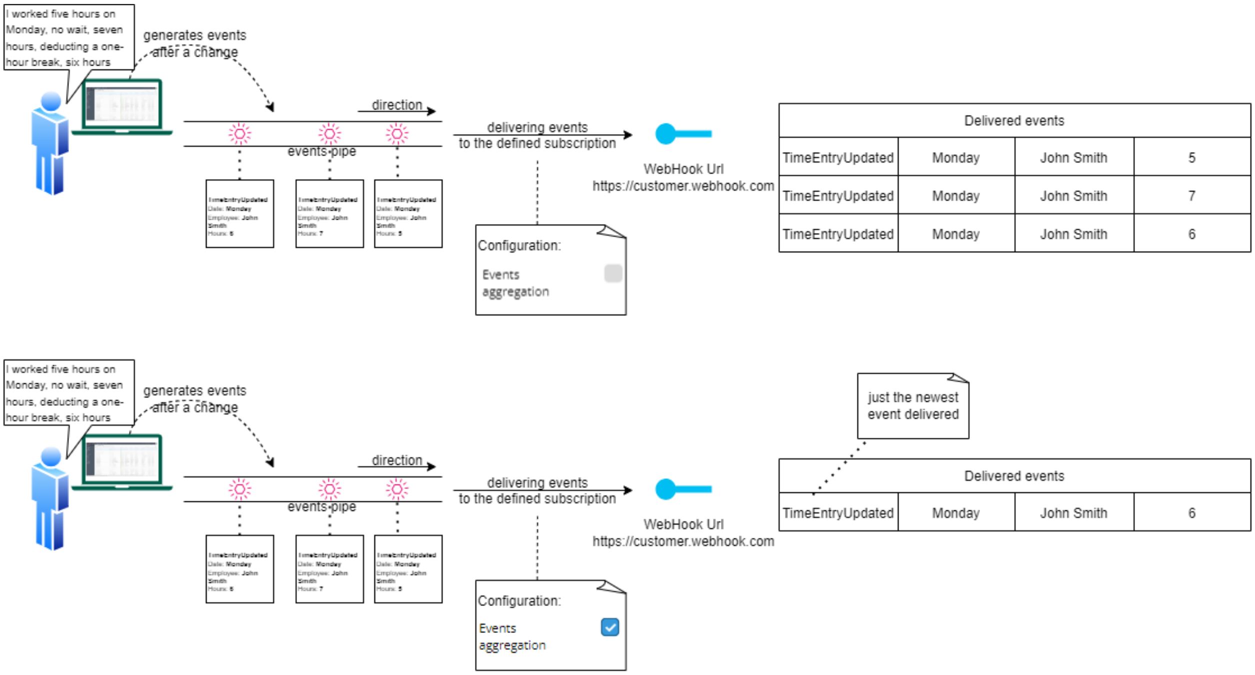
From the toolbar you can take one of the following actions:

- Retry Selected Event – Retry all of the failed results associated with that event (event across multiple webhooks)
- Retry All Failed Results – Retry all failed results across all webhooks
- Clear OK Results – Clear the OK results for the current webhook
- Clear All – Clear all results (including the failed ones)

Use Event Aggregation

Almost every change in WorkBook triggers a new event. Some webhook events might be triggered more often than they were previously. This can result in many events being delivered to a subscription. You can use event aggregation to reduce the number of calls to a webhook, which in turn reduces the number of events that are delivered to a subscription.

The following figure shows an example of how event aggregation works to reduce the number of events that are delivered to a subscription.


I worked five hours on Monday, no wait, seven hours, deducting a one-hour break, six hours
generates events after a change
direction
events pipe
TimeEntryUpdated Date: Monday Employee: John Smith Hours: 6
TimeEntryUpdated Date: Monday Employee: John Smith Hours: 7
TimeEntryUpdated Date: Monday Employee: John Smith Hours: 5
delivering events to the defined subscription
WebHook Url https://customer.webhook.com
Configuration:
Events aggregation ☐
I worked five hours on Monday, no wait, seven hours, deducting a one-hour break, six hours
generates events after a change
direction
events pipe
TimeEntryUpdated Date: Monday Employee: John Smith Hours: 6
TimeEntryUpdated Date: Monday Employee: John Smith Hours: 7
TimeEntryUpdated Date: Monday Employee: John Smith Hours: 5
delivering events to the defined subscription
WebHook Url https://customer.webhook.com
Configuration:
Events aggregation ☑
just the newest event delivered

Delivered events			
TimeEntryUpdated	Monday	John Smith	5
TimeEntryUpdated	Monday	John Smith	7
TimeEntryUpdated	Monday	John Smith	6

Delivered events			
TimeEntryUpdated	Monday	John Smith	6

Following this example, three events are delivered to your subscription if you do not use event aggregation. With event aggregation, only one event—the newest—is delivered.

Before you enable event aggregation, consider the following:

- You can use event aggregation with only the following events:
  - Events that are related to updates
  - Events that have the same types
  - Events that are associated with the same entity
  - Events that have been retried the same number of times

  For example:

Event ID	Event Type	Operation Type	Entity ID	Retries
1	JobUpdated	Update	10	0
2	TaskUpdated	Update	6	0

Event ID	Event Type	Operation Type	Entity ID	Retries
3	JobUpdated	Update	10	0
4	JobUpdated	Update	5	0
5	JobUpdated	Update	10	1
6	CashFlowCreated	Insert	67	0

Only the events with IDs 1 and 3 are aggregated.

- An aggregated event contains only information from the most recent/newest event from the aggregated collection. Then, you might need to call WorkBook API to get an entity in the current state.
- To ensure that the event aggregation process aggregates more events, you might need to increase the WorkBook setting Webhooks.EventsToLoadDelay. This setting is used to determine how often WorkBook loads the events for processing (in seconds). If that value is bigger, more events are aggregated, because the period on which aggregation operates is longer. However, a longer period means that the data that is delivered to the subscription can be older. The suggested value is 30 (seconds).

To determine whether event aggregation is enabled in your events, and to identify the number of events that are represented by each aggregated event, you can inspect the ItemCount property in the event body. If the value of ItemCount is 1, the event is a single, non-aggregated event. If the value of ItemCount is greater than 1, this is the number of individual events that have been aggregated into this single event. Understanding this property is critical for understanding the composition of the data that you are handling.

About Deltek

Better software means better projects. Deltek is the leading global provider of enterprise software and information solutions for project-based businesses. More than 23,000 organizations and millions of users in over 80 countries around the world rely on Deltek for superior levels of project intelligence, management and collaboration. Our industry-focused expertise powers project success by helping firms achieve performance that maximizes productivity and revenue. www.deltek.com