Sage 300 Web API

Developer Reference

December 2019

Contents

1. Introduction

Sage 300 has a longstanding history of interoperability with third-party integrations and systems. With the advent of the Sage 300 XAPI, third party developers were able to directly access the Sage 300 business layer, performing tasks such as data manipulation and system process invocation. Then came the Sage 300 COM interface which, on top of the XAPI, added the ability to cross machine boundaries via ActiveX objects on remote clients. Not soon after came our .NET interface which streamlined the way developers integrate with Sage 300.

However, using the aforementioned interfaces required intimate knowledge of the Sage 300 business layer. One requirement is to understand the View protocol which encapsulates accesses to the data layer. Another requirement is to have a complete understanding of what specific View components do and how they relate and interact with one another. Even though the technology is there to access the interfaces, the work involved is anything but trivial.

The Sage 300 Web API layer was created as a way to solve this complexity. It is based on OData which, aside from being a common standard for interfacing over the web, is self-documenting through XML+Atom Pub making it very easy to understand. Data is transferred in JSON format making it human readable while still maintaining good performance. Many frameworks and tools are available to make interfacing easier; in fact, it is possible to retrieve Sage 300 data with just a web browser.

2. Making a Sage 300 Web API request

2.1 Anatomy of a resource URL

Example: `http://localhost/Sage300WebApi/v1.0/-/SAMLTD/AR/ARCustomers`

The first step in making a Web API request for a Sage 300 resource is to construct the corresponding uniform resource locator (URL). Here are the components that comprise a Sage 300 resource URL:

{protocol}://{host-application-path}/{api-version}/-/{company}/{app-module}/{resource}

Component	Description	Examples
{protocol}	The application protocol enabled in IIS setup	`http, https`
{host-application-path}	The path to the Web API application	`localhost/Sage300WebApi, yourdomain.com`
{api-version}	The API version	`v1.0`
{company}	The org ID of the company being requested	`SAMLTD, SAMINC`
{app-module}	Sage 300 application module where the requested resource resides	`GL, AP, AR`
{resource}	The resource entity being requested	`GLAccounts, APVendors, ARCustomers`

For a complete list of resources that Sage 300 Web API supports, refer to the Sage 300 Web API Endpoint Reference document, available under “Technical Documentation” here:

http://cdn.na.sage.com/docs/en/customer/300erp/Documentation.htm

Additionally, the list of resources can be discovered through `$metadata` requests. Refer to the “Discovering resources” section of this document for more information.

2.2 HTTP headers

In addition to constructing the appropriate URL, a Sage 300 Web API request requires two HTTP headers to be specified: authorization and content type.

2.2.1 Authorization

Example:
`Authorization: Basic QURNSU46QURNSU4=`
(constructed for user "ADMIN" and password "ADMIN")

The Sage 300 Web API uses Basic Access Authentication for authorization control. Every request made must have an authorization header field with a value constructed using a valid Sage 300 username and corresponding password.

If security has been turned off for the given system database, an empty password should be used when constructing the Basic Access Authentication value.

In order to construct the value for the authorization header, the username must first be concatenated with the password with a colon as the separator (*username:password*). This value must then be encoded using Base64 (RFC2045-MIME) and be appended to the phrase "`Basic `" (Note the trailing <space> character) Thus, for the case of user ADMIN with a password of ADMIN, the Authorization header value would yield:

```
Basic QURNSU46QURNSU4=
```

For more information about Basic Authentication, refer to section 2 of the RFC 2617 documentation:

https://www.ietf.org/rfc/rfc2617.txt.

2.2.2 Content-Type

Example: `Content-Type: application/json`

All Sage 300 Web API requests are required to specify a Content-Type header value of `application/json`. Not doing so will result in an HTTP response code of 500, indicating an Internal Server Error.

2.3 Payload

Some types of requests, like those that update resources in Sage 300 will require a payload in the body of the request. Payloads use the JavaScript Object Notation (JSON) data-interchange format. Here is an example of what a typical payload looks like:

```
{
  "CustomerNumber": "1200",
  "ShortName": "BLACK",
  "GroupCode": "RTL",
  "NationalAccount": "",
  "Status": "Active",
  "InactiveDate": null,
  "DateLastMaintained": "2010-08-18T00:00:00",
  "OnHold": "No",
  "AccountSet": "USA",
  "AccountType": "BalanceForward",
  "Terms": "DUETBL",
  "TaxGroup": "CALIF",
  "CustomerOptionalFieldValues": [
    {
      "CustomerNumber": "1200",
      "OptionalField": "CREDTWARNING",
      "Value": "0",
      "CustomerOptionalFieldValueType": "YesNo",
    },
    {
      "CustomerNumber": "1200",
      "OptionalField": "UPSZONE",
      "CustomerOptionalFieldValueType": "Text",
      "Value": "RED",
    }
  ]
}
```

2.3.1 Ordering of properties

When constructing a Sage 300 Web API request, it is important to consider the order of the properties within a JSON payload carefully because the order in which the properties appear will be reflected in the order in which they are applied to the system.

For those experienced with Sage 300 Views, this means that the corresponding View fields will be Put to in the order that they appear in the payload. Thus, the ordering of properties can cause very different results for some resources.

2.3.2 Partial payloads

Sage 300 Web API resources supports the use of partial payloads for all HTTP verbs. More specifically, properties can be left out of the JSON payload where the corresponding default value is sufficient.

In addition to simplifying the construction of a Sage 300 Web API request, using partial payloads has the added benefit of improving performance dramatically. As such, Sage recommends the use of partial payloads in most cases.

3. Discovering resources

Example: `GET http://localhost/Sage300WebApi/v1.0/-/SAMLTD/AR/$metadata`

The `$metadata` resource is used for discovering the list of resources available through the Sage 300 Web API and learning about the properties these resources support. A GET request made on the `$metadata` resource returns an Atom (XML) based EDMX document containing a complete listing of the feeds, types, properties and relationships that are exposed.

Since the `$metadata` resource is company and application specific, both the company org ID as well as the application module ID must be specified within the GET request. The format for this request is described in the following:

{protocol}://{host-application-path}/{api-version}/-/{Company}/{app-module}/$metadata

The following is an extract of a sample response you can expect from the `$metadata` request on SAMLTD and AR:

```
<?xml version="1.0" encoding="utf-8"?>

<edmx:Edmx Version="4.0" xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx">

    <edmx:DataServices>

        <Schema Namespace="Sage.CA.SBS.ERP.Sage300.AR.WebApi.Models" xmlns="
http://docs.oasis-open.org/odata/ns/edm">

            <EntityType Name="Customer">

                <Key>

                    <PropertyRef Name="CustomerNumber" />

                </Key>

                <Property Name="CustomerNumber" Type="Edm.String"
Nullable="false" />

                <Property Name="ShortName" Type="Edm.String" />

                <Property Name="GroupCode" Type="Edm.String" />

                <Property Name="NationalAccount" Type="Edm.String" />

                <Property Name="Status" Type="
Sage.CA.SBS.ERP.Sage300.AR.WebApi.Models.StatusEnum" Nullable="false" />

                <Property Name="InactiveDate" Type="Edm.DateTimeOffset" />

                <Property Name="DateLastMaintained" Type="Edm.DateTimeOffset" />

                <Property Name="OnHold" Type="
Sage.CA.SBS.ERP.Sage300.AR.WebApi.Models.OnHoldEnum" Nullable="false" />

                <Property Name="AccountSet" Type="Edm.String" />
                <Property Name="AccountType" Type="
Sage.CA.SBS.ERP.Sage300.AR.WebApi.Models.AccountTypeEnum" Nullable="false" />

                <Property Name="Terms" Type="Edm.String" />

                <Property Name="TaxGroup" Type="Edm.String" />

                <Property Name="CustomerOptionalFieldValues"
Type="Collection(Sage.CA.SBS.ERP.Sage300.AR.WebApi.Models.CustomerOptionalFieldV
alue)" />

                ...

            </EntityType>

            <ComplexType Name="CustomerOptionalFieldValue">

                <Property Name="CustomerNumber" Type="Edm.String" />

                <Property Name="OptionalField" Type="Edm.String" />

                <Property Name="Value" Type="Edm.String" />

                <Property Name="CustomerOptionalFieldValueType" Type="
Sage.CA.SBS.ERP.Sage300.AR.WebApi.Models.CustomerOptionalFieldValueTypeEnum"
Nullable="false" />

                ...

            </ComplexType>

           ...

        </Schema>

        ...

        <Schema Namespace="Default" xmlns="http://docs.oasis-
open.org/odata/ns/edm">

            <EntityContainer Name="Container">

                <EntitySet Name="ARCustomers"
EntityType="Sage.CA.SBS.ERP.Sage300.AR.WebApi.Models.Customer" />

                ...

            </EntityContainer>

        </Schema>

    </edmx:DataServices>

</edmx:Edmx>
```

Examining this Atom feed, we see that all the available resources are listed near the bottom inside a Schema node with the `Default` Namespace. Here we discover a resource named `Customers` with an EntityType of `Sage.CA.SBS.ERP.Sage300.AR.WebApi.Models.Customer`. Thus, specifics about the AR Customer model can be found inside the Schema node with a Namespace of `Sage.CA.SBS.ERP.Sage300.AR.WebApi.Models`. Examining the top of the feed, we see the properties within the AR Customer model:

```
<Key>

    <PropertyRef Name="CustomerNumber" />
```

```
</Key>

<Property Name="CustomerNumber" Type="Edm.String" Nullable="false" />

<Property Name="ShortName" Type="Edm.String" />

...

<Property Name="CustomerOptionalFieldValues"
Type="Collection(Sage.CA.SBS.ERP.Sage300.AR.WebApi.Models.CustomerOptionalFieldV
alue)" />
```

By examining the Key node, we discover that Customer has a key with a single property called `CustomerNumber`. Note that resources with a composite key will have multiple children within this node.

We also see the properties of a Customer listed. For example, we see a property named `ShortName` of type string. The `CustomerOptionalFieldValues` is a special property because its type is a collection of `CustomerOptionalFieldValue`. This is an example of a detail relationship where there is a one-to-many correspondence.

In the current version of the Sage 300 Web API, details are exposed as `ComplexType` nodes. As you can see below, in the Customer `EntityType` node, the properties of `CustomerOptionalFieldValues` are listed in the `ComplexType` node with the Name `CustomerOptionalFieldValues`.

```
<ComplexType Name="CustomerOptionalFieldValues">

    <Property Name="CustomerNumber" Type="Edm.String" />

    <Property Name="OptionalField" Type="Edm.String" />

    <Property Name="Value" Type="Edm.String" />

    <Property Name="CustomerOptionalFieldValueType" Type="
Sage.CA.SBS.ERP.Sage300.AR.WebApi.Models.CustomerOptionalFieldValueTypeEnum"
Nullable="false" />

    ...

</ComplexType>
```

The same method of discovery can be made for all other resources in the system.

4. Requesting a resource feed (HTTP GET)

4.1 Basic

Example: `GET http://localhost/Sage300WebApi/v1.0/-/SAMLTD/AR/ARCustomers`

GET {protocol}://{host-application-path}/{api-version}/-/{company}/{app-module}/{resource}

The simplest way to retrieve records of a particular resource is to call a GET directly on the resource. The result is a list of records presented as an OData feed.

Currently the maximum number of records that can be returned for a single GET request is 100. In order to retrieve records beyond the first 100 or find records based on a given set of criteria, the use of query options is required.

To reduce the number of requests necessary for transferring large datasets, the maximum page size can be adjusted from the default value of 100 records per request.

4.2 Query options

{protocol}://{host-application-path}/{api-version}/-/{company}/{app-module}/{resource}?{query-options}

Component	Description	Examples

{query-options}		`$skip=100`
		`$filter=CustomerNumber eq 'BARMART'`

Query options allow the caller to specify a subset of records to be returned in the response feed. The query options portion of a Sage 300 Web API request appears at the end of the request URL and begins with a question mark symbol (`?`). Each type of query option is prefixed with a dollar sign character (`$`) as listed in the following subsections.

4.2.1 $skip

Example: `GET http://localhost/Sage300WebApi/v1.0/-/SAMLTD/AR/ARCustomers?$skip=100`

The `$skip` query option must be a positive integer N that specifies the records beyond which the response feed should start with, effectively skipping N records. This is typically used to retrieve records beyond the top 100, the default maximum limit of a single response page.

4.2.2 $top

Example: `GET http://localhost/Sage300WebApi/v1.0/-/SAMLTD/AR/ARCustomers?$top=5`

The `$top` query option must a positive integer N that specifies the maximum number of records the response feed could contain, effectively selecting the first N records.

4.2.3 $count

Example:
`GET http://localhost/Sage300WebApi/v1.0/-/SAMLTD/AR/ARCustomers?$count=true`

The `$count` query option specifies whether a count of all records will be returned as part of the response feed, regardless of how many records are actually returned in the response.

The following is an excerpt of a sample response from querying AR Customers:

```
{
  "odata.context": "http://localhost/Sage300WebApi/v1.0/-
/SAMLTD/AR/$metadata#ARCustomers",
  "odata.count": "625",
  "value": [
    {
      "CustomerNumber": "1100",
       ...
   },
    {
      "CustomerNumber": "1105",
       ...
   },
   ...
           ]
}
```

Note that even though the page only contains 100 customers, `odata.count` is 625, indicating there are a total of 625 customers in the company.

4.2.4 $filter

Example: `GET http://localhost/Sage300WebApi/v1.0/-/SAMLTD/AR/ARCustomers?$filter=ShortName eq 'BLACK'`

The `$filter` query option specifies a set of criteria that records must satisfy before being returned, effectively allowing the caller to retrieve a subset of the resource collection based on a specified filter. This filter is constructed using the OData filter expression language and is very flexible.

Expressions can reference properties as well as literals. Literal values can be strings enclosed in single quotes, dates, numbers, and boolean values.

The following is a list of the operators that are supported in the Sage 300 Web API:

Operator	Description	Example
eq	Equals	`/ARCustomers?$filter=CustomerName eq ‘1200’`
ne	not equals	`/ARCustomers?$filter=AccountType ne 'BalanceForward'`
gt	greater than	`/ARCustomers?$filter=NumberOfDaysToPay gt 30m`
lt	less than	`/ARCustomers?$filter=AmountPastDue lt 99.99m`
le	less than or equal to	`/ARCustomers?$filter=DateLastMaintained le datetime'2015-12-31T12:00'`
and	logical and	`/ARCustomers?$filter= GroupCode eq 'WHL' and OnHold eq ‘No’`
or	logical or	`/ARCustomers?$filter=City eq 'Los Angeles' or City eq 'San Francisco'`

The following is a list of functions that are supported:

Operator	Example
bool substringof(string po, string p1)	`/ARCustomers?$filter=substringof('BAR', CustomerNumber)`
bool endswith(string p0, string p1)	`/ARCustomers?$filter=endswith(CustomerNumber, 'MART')`
bool startswith(string p0, string p1)	`/ARCustomers?$filter=startswith(CustomerNumber, 'BAR')`
int length(string p0)	`/ARCustomers?$filter=length(CustomerName) gt 10`

4.2.5 Format of literals within filter parameter

When using literal values within a filter parameter, they must be formatted properly for the value to be recognized. Here is a list of literal types that have special formatting.

Literal Type	Example
String	`‘1200’`
Decimal	`99.99m`
DateTime	`datetime'2015-12-31T12:00'`

4.2.6 Combining query options

Example: `GET http://localhost/Sage300WebApi/v1.0/-/SAMLTD/AR/ARCustomers?$filter=CustomerNumber gt '1100'&$skip=5&top=2&$count=true`

Query options can be combined using the ampersand (`&`) symbol; however, each type of query option should only appear once per request.

5. Requesting a resource entry (HTTP GET)

5.1 Basic

Example: `GET http://localhost/Sage300WebApi/v1.0/-/SAMLTD/AR/ARCustomers('1200')`

GET

{protocol}://{host-application-path}/{api-version}/-/{company}/{app-module}/{resource}({entity-key})

Component	Description	Examples
{entity-key}		`'1200', CustomerNumber = 'BARMART'`

Requesting for an entry returns a response with a single record. If the key value of the sought after record is known ahead of time, a request for an entry is much more performant. The response payload for an entry is also more streamlined since no OData feed container is returned.

In order to request for an entity, an entity key must be supplied after the resource name of the request URL. The entity key must be enclosed within parenthesis characters `()`.

5.1.1 Entity key without property name

The simplest way to construct an entity key is to just use the value of the key.

Strings values should be enclosed in single quote characters (for example: `'BARMART'`) but this is not a requirement when no space characters appear within the value.

Numerical values should only contain numeric characters, without group separators. (for example: `25`)

5.1.2 Entity Key with property name

To make the entity key property explicit, a property name can be used identify the key property followed be an equals character (=) and the value as described in the previous section. (for example: BatchNumber = 65).

The use of a property names is more important when the resource key is composite.

5.2 Composite Key

Example: `GET http://localhost/Sage300WebApi/v1.0/-/SAMLTD/AR/ARReceiptAndAdjustmentBatches(BatchRecordType = 'CA', BatchNumber = 65)`

For resources that have more than one key segment, each segment value must appear in the entity key or an error response would be returned. Key segment values must be separated by a comma character (`,`). The key segment values can be in any order if property names are provided, otherwise, they must follow the exact order as the key properties appear in the associated entity type's metadata.

For information about retrieving metadata data for a resource, refer to the "Discovering resources" section of this document.

6. Deleting a resource entry (HTTP DELETE)

Example: `DELETE http://localhost/Sage300WebApi/v1.0/-/SAMLTD/AR/ARCustomers('1200')`

DELETE

{protocol}://{host-application-path}/{api-version}/-/{company}/{app-module}/{resource}({entity-key})

Deletion of records through the Sage 300 Web API is done through a DELETE HTTP request. Currently, Sage 300 Web API only supports deletion of one record per request. An entity key is required in order to identify the record being removed. The URL for the DELETE request is exactly the same as performing a GET of an entry, down to the format of the entity key.

7. Inserting a resource entry (HTTP POST)

Example: `POST http://localhost/Sage300WebApi/v1.0/-/SAMLTD/AR/ARCustomers`

POST

{protocol}://{host-application-path}/{api-version}/-/{company}/{app-module}/{resource}

Insertion of records through the Sage 300 Web API is done through an HTTP POST request. Currently, Sage 300 Web API only supports the insertion of one record per request. Unlike all other requests up to this point, an insertion POST request requires that an entry payload be sent as the request body.

For more information about the format of a payload, refer to the "Payload" section of this document.

The easiest way to construct a payload for insertion is to use the response from a GET request for a single entry of the same resource as a starting point. Extraneous properties can be removed from this entry payload and the values of the other properties can be changed to suit the needs of the new record. It is important to ensure that the entry payload does contain all key properties however.

For resources where the key is system-generated (for instance, the transaction resources), the new record being created will most likely have different key values than those found in the request entry payload. Because of this, the response of insertion requests will contain an entry payload of the record that was inserted into the system thus indicating the key values generated by the system.

8. Updating by replacement (HTTP PUT)

Example: `PUT http://localhost/Sage300WebApi/v1.0/-/SAMLTD/AR/ARCustomers('1200')`

PUT

{protocol}://{host-application-path}/{api-version}/-/{company}/{app-module}/{resource}({entity-key})

Updates of records in Sage 300 Web API is done through HTTP PUT requests. Currently, Sage 300 Web API only supports the update of one record per request. Similar to insertion, an update request requires an entry payload be sent as the request body.

Like all other requests, PUT update requests supports partial payloads; however, you must ensure that details payloads appear as you require them after the update. A PUT update request acts as a replacement method and will purge any details (like optional field details) that do not appear in the payload body.

9. Invoking special services (HTTP POST)

Example: `POST http://localhost/Sage300WebApi/v1.0/-/SAMLTD/AR/ARPostInvoices($process)`

POST

{protocol}://{host-application-path}/{api-version}/-/{company}/{app-module}/{resource}($process)

Sage Web API supports the invocation of special service process resources to perform such tasks as posting invoices and generating GL batches.

For a complete list of process endpoints, refer to the Sage 300 Web API Endpoint Reference document, available under "Technical Documentation" here:

http://cdn.na.sage.com/docs/en/customer/300erp/Documentation.htm

To invoke a special service, a POST request must be made on the process endpoint with an entity key of `$process`. An entry payload in the body of the request (similar to an insertion request) is necessary to specify the parameters for the service invocation. Because process resources do not support GET requests, it may be necessary to retrieve the template entry (as described in the following section) to use as a starting point.

10. Retrieving resource templates (HTTP POST)

Example: `POST http://localhost/Sage300WebApi/v1.0/-/SAMLTD/AR/ARPostInvoices($template)`

POST

{protocol}://{host-application-path}/{api-version}/-/{company}/{app-module}/{resource}($template)

Sage 300 Web API supports a way of retrieving a sample entry payload from a resource with a template request. This is especially useful for process endpoints where a GET is not supported or where no records exist to allow a GET of an entry.

In order to retrieve the resource template, an HTTP POST should be performed on the resource using the entity key `$template`. The response will be an entry payload where all properties contain the default value from the underlying business layer.

11. Errors

In addition to returning error HTTP status codes for failures to process requests, Sage 300 Web API provides more information about the nature of the problem with standard OData error payloads within the response body.

The following is an example of such a response for a failed request:

```
HTTP Status: 404

{
  "error": {
    "code": "RecordNotFound",
    "message": {
      "lang": "en-US",
      "value": "A Record cannot be found for the specified entity key"
    }
  }
}
```

11.1 Error code

The error code is a CamelCase string indicating the type of error encountered. For automated resolutions of Sage 300 Web API errors, the error code should be used for determining the nature of the issue since it is more precise than the HTTP status code. Here is a complete list of error codes and their meanings:

Error Code	Meaning
InternalServerError	An unexpected and unhandled server error
Unauthorized	Invalid credentials used in authentication header
Forbidden	The specified user does not have enough rights to process the request
TooManyRequests	Server was flooded with too many concurrent requests
InvalidParameters	The parameters used in the URL are invalid or malformed
ResourceNotFound	The resource requested does not exist
InvalidEntityKey	The entity key is malformed or is not compatible with the requested resource

RecordNotFound	No record can be found for the specified entity key
MethodNotAllowed	The request is not supported by the specified resource
InvalidPayload	The payload body is not valid for the resource specified
InvalidAction	A POST request was made with an entity key that is not `$template` or `$process`
DBLinkError	An application installation problem is preventing the request from being processed
SessionError	A system installation problem is preventing the request from being processed
RecordDuplicate	An insert request was made with a key value that already exist in the system
RecordInvalid	The payload used for the request is not valid due to business rules
General	The request failed due to an error in the business layer

11.2 Error message

The error message is a human readable phrase that describes the problem in an even more precise manner than the error code. It is intended to be displayed to end users; however, this should not be interpreted by automated systems, since the error message can vary greatly under similar situations.

12. Performance tips

It is possible to adjust the performance of Sage 300 Web API if improvements are required for integration purposes.

12.1 Increase page size for GET requests

When large amounts of records need to be retrieved through Sage 300 Web API, the page size setting can be increased to reduce the number of GET requests and thus reduce the overall time for the entire process.

In order to increase the page size setting, you can use a text editor to open the Web.config file located in the Online\WebApi folder under your Sage 300 installation folder. Search for the line in this file that contains the following text:

```
<add key="PageSize" value="100" />
```

Increase the value to the required page size. The recommended setting for large numbers of records is 1000.

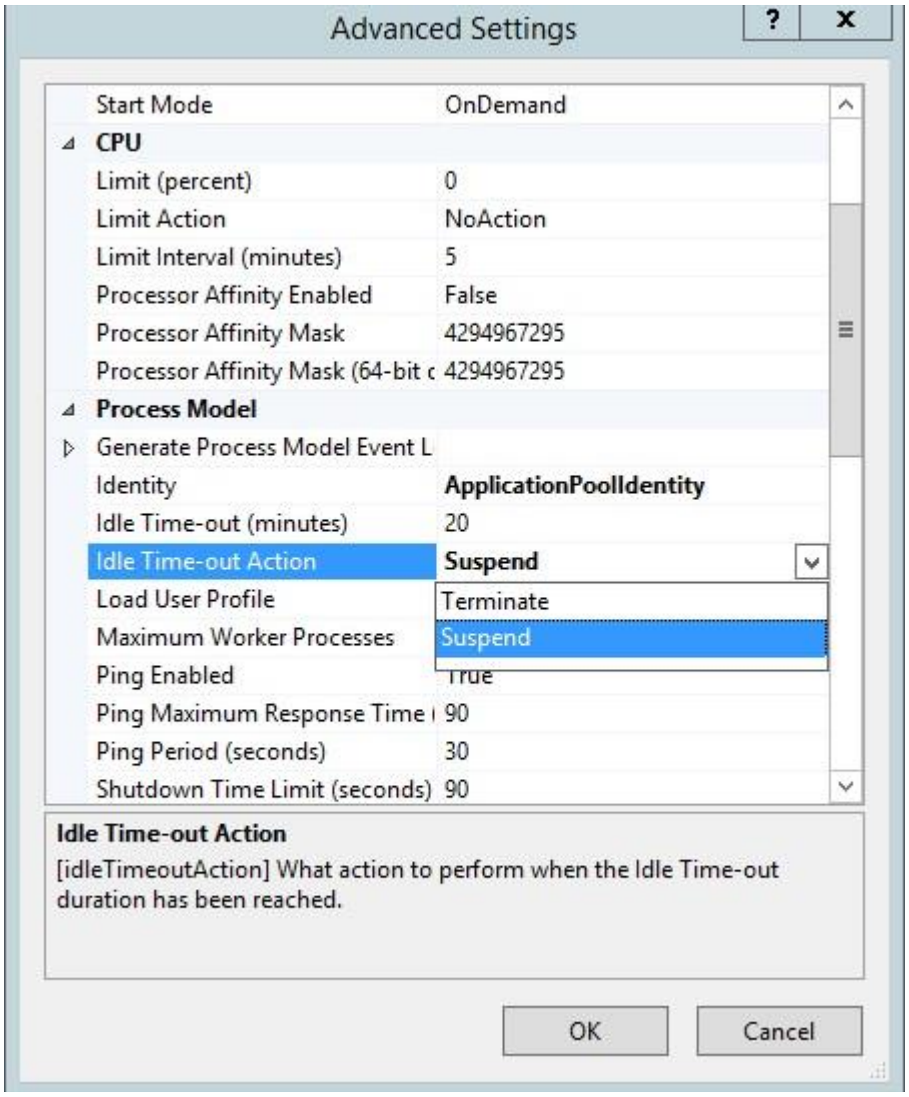
12.2 Change IIS Idle Time-out settings

By default, IIS terminates a web application after 20 minutes of inactivity. This means that if Sage 300 Web API requests are not made within 20 minutes of one another, a request will take longer than usual to process. To curb this behavior, the IIS application pool Idle Time-out setting can be adjusted.

To increase the Time-out time and Time-out Action:

1. Open IIS Manager.
2. In **Application Pools**, select **Sage 300 Pool.**
3. In the **Actions** pane, click **Advanced Settings**.
4. In the **Process Model** section, change the **Idle Time-out Action** from Terminate to Suspend.

   The **Idle Time-out** time should be increased to a more reasonable amount. 1447 minutes is the recommended value, as it is greater than a single day.



Start Mode	OnDemand
CPU	
Limit (percent)	0
Limit Action	NoAction
Limit Interval (minutes)	5
Processor Affinity Enabled	False
Processor Affinity Mask	4294967295
Processor Affinity Mask (64-bit c	4294967295
Process Model	
Generate Process Model Event L	
Identity	**ApplicationPoolIdentity**
Idle Time-out (minutes)	20
Idle Time-out Action	**Suspend**
Load User Profile	Terminate
Maximum Worker Processes	Suspend
Ping Enabled	True
Ping Maximum Response Time (	90
Ping Period (seconds)	30
Shutdown Time Limit (seconds)	90

Idle Time-out Action

[idleTimeoutAction] What action to perform when the Idle Time-out duration has been reached.