


## Since & Until Values

 Many API endpoints support the `skip` (and `first`) parameters to implement data paging. This type of paging is great when filter fields don't change often.

One situation where these paging parameters do not work well is when syncing data. Syncing usually needs page results by modification timestamp. Problems can occur however, if the underlying data changes while a caller is paging through results. The chance of this increases when dealing with large data sets, such as sales history, inventory or customer records.

Furthermore – depending on the data type – a modification timestamp may not have sufficient resolution to be the sole sort criteria used for sorting modified results. Additional data has to be added to the criteria used to sort results. API callers should not need to bother with determining these criteria.

To get around these problems, several API endpoints also support a `since` parameter (and `until` result) that should be used for paging based on modification date. To use them, you:

- Call the API endpoint with a blank – *not null* – `since` parameter.
  - A blank value indicates that since paging should be started from the first set of results.
  - A null value indicates that no 'since' paging should be used.
- You receive the endpoint results, making note of the `until` value that is returned.
- To get the next 'page' of data, you call the API again, passing the `until` value as the new `since` parameter value.

### Format

You should treat `Since` and `Until` parameters as opaque values, without attempting to parse them for meaningful information. POS will generate the `since` and `until` values depending on the type of data it is paging.

POS may change the internal format it uses for `since` and `until` values. For this reason, do not parse the values or rely upon their internal format.

## Storage

You may store an `until` value (i.e. the `since` value for the next API call) if you are implementing a data synchronization process that runs at timed intervals. The value should be stored as a raw string value. While there is no technical limit to the `until` value size, they are typically short. If storing in a database it's still good practice to store them as `VARCHAR(MAX)` or equivalent.

In major version changes - or critical bug fixes - POS may change the format of an `until` value. If you have stored an `until` value with an older format, you may receive an `INVALID_SINCE_FORMAT` error in your next API call. It's good practice to detect and handle this error.

## Example

### Initial Query

This example uses a simple call to the `sale` endpoint, but the same approach can be applied to any endpoint that supports `since` and `until` parameters.

For the first call to the endpoint, ensure the `since` value is a blank string:

```
query MyQuery {
  sale(since: "", first: 2) {
    until
    results {
      id
    }
  }
}
```

We are also supplying a `first` parameter for the purposes of this example - to limit the number of results returned - but POS will supply a default `first` value if you do not specify your own.

The endpoint will return a response similar to:

```
{
  "data": {
```

```

"sale": {
  "until": "1554830160000;4499096027743125506;",
  "results": [
    {
      "id": "4499096027743125505"
    },
    {
      "id": "4499096027743125506"
    }
  ]
}
}
}

```

Note the `until` value - this is what you will pass in the next call.

### Next Query

For the next call to the endpoint, pass the last `until` value as the `since` parameter value:

```

query MyQuery {
  sale(since: "1554830160000;4499096027743125506;", first: 2) {
    until
    results {
      id
    }
  }
}

```

You'll receive a response similar to:

```

{
  "data": {
    "sale": {
      "until": "1554830880000;4499096027743125508;",
      "results": [
        {
          "id": "4499096027743125507"
        },
        {
          "id": "4499096027743125508"
        }
      ]
    }
  }
}

```



```
    }  
  ]  
}  
}  
}
```

## Final Query

You can continue the process above to iterate through all pages of modified data. You will know that there are no longer any data modifications when the `until` value returned is `null`:

```
{  
  "data": {  
    "sale": {  
      "until": null,  
      "results": []  
    }  
  }  
}
```

## Throttling

POS currently does not enforce automated throttling of API calls when executing multiple calls with `since` and `until` parameters.

It is good practice - particularly when calling the API of POS Cloud servers - to ensure that you add a small time delay between API calls.