# 3 Implicit Parameters

This chapter describes the implicit parameters used in REST service handlers that are not explicitly declared. Oracle REST Data Services (ORDS) adds these parameters automatically to the resource handlers.

## 3.1 List of Implicit Parameters

The following table lists the implicit parameters:

Note:Parameter names are case sensitive. For example, :CURRENT\_USER is not a valid implicit parameter.

Table 3-1 List of Implicit Parameters

| Name | Type | Access Mode | HTTP Header | Description | Introduced |
| --- | --- | --- | --- | --- | --- |
| :body | BLOB | IN | N/A | Specifies the body of the request as a temporary BLOB. | 2.0 |
| :body\_text | CLOB | IN | N/A | Specifies the body of the request as a temporary CLOB. | 18.3 |
| :body\_json | CLOB | IN | N/A | Specifies the body of the request as a temporary CLOB in JSON format. | 24.1 |
| :content\_type | VARCHAR | IN | Content-Type | Specifies the MIME type of the request body, as indicated by the Content-Type request header. | 2.0 |
| :current\_user | VARCHAR | IN | N/A | Specifies the authenticated user for the request. If no user is authenticated, then the value is set to null. | 2.0 |
| :forward\_location | VARCHAR | OUT | X-ORDS-FORWARD-LOCATION | Specifies the location where Oracle REST Data Services must forward a GET request to produce the response for this request. | 18.3 |
| :fetch\_offset | NUMBER | IN | N/A | Specifies the zero-based offset of the first row to be displayed on a page. | 18.3 |
| :fetch\_size | NUMBER | IN | N/A | Specifies the maximum number of rows to be retrieved on a page. | 18.3 |
| :page\_offset | NUMBER | IN | N/A | Specifies the zero based page offset in a paginated request. Note: The :page\_offset parameter is deprecated. Use :row\_offset parameter instead. | 2.0 |
| :page\_size | NUMBER | IN | N/A | Specifies the maximum number of rows to be retrieved on a page. Note: The :page\_size parameter is deprecated. Use :fetch\_size parameter instead. | 2.0 |
| :row\_offset | NUMBER | IN | N/A | Specifies the one-based index of the first row to be displayed in a paginated request. | 3.0 |
| :row\_count | NUMBER | IN | N/A | Specifies the one-based index of the last row to be displayed in a paginated request. | 3.0 |
| :status\_code | NUMBER | OUT | X-ORDS-STATUS-CODE | Specifies the HTTP status code for the request. | 18.3 |

### 3.1.1 About the :body parameter

The :body implicit parameter is used in the resource handlers to receive the contents of the request body as a temporary BLOB.

Note:Only POST or PUT requests can have a request body. The HTTP specification does not permit request bodies on GET or DELETE requests.

Example 3-1 Example The following example illustrates a PL/SQL block that stores the request body in a database table:

begin insert into tab (content) values (:body); end;

Note:

The :body implicit parameter must be dereferenced exactly once in a PL/SQL block. If it is dereferenced more than once, then the second and subsequent dereferences will appear to be empty. This is because the client sends the request body only once. If you need this value more than once, then assign it to a local variable, and dereference the local variable instead.

You can use either one of the implicit parameters :body or :body\_text. Otherwise, the PL/SQL block displays an error message “Duplicate steam parameter’’.

If you use either :body or :body\_text, then you cannot use :bind notation to read attributes of the JSON payload of the request.

The following example will not work as intended because it dereferences the :body parameter twice:

begin insert into tab1(content) values (:body); – request body will be inserted insert into tab2(content) values (:body); – an empty blob will be inserted end;

To avoid this limitation, the :body parameter value must be assigned to a local PL/SQL variable before it is used. This enables the local variable to be dereferenced more than once:

declare l\_content blob := :body; begin insert into tabl(content) values(l\_content); insert into tab2(content) values(l\_content); end;

### 3.1.2 About the :body\_text parameter

The :body\_text implicit parameter is used in the resource handlers to receive the contents of the request body as a temporary CLOB. Typically, the content of the request body is textual (for example JSON or HTML content) and so, receiving the request body as a CLOB saves the resource handler author from the effort of converting the :body BLOB parameter to a CLOB instance.

Note::body\_text implicit parameter must only be dereferenced once inside the entire PL/SQL block. If you need this value more than once, assign it to a local variable, and dereference the local variable instead.

You can use either one of the implicit parameters :body or :body\_text. Otherwise, the PL/SQL block displays an error message “Duplicate steam parameter’’.

It is recommended to use :body\_text ( a character representation ) rather than :body ( a binary representation ) particularly where the PL/SQL block uses JSON functions to process the request body efficiently.

### 3.1.2 About the :body\_json parameter

The :body\_json implicit parameter can be used in Resource Handlers to receive multipart/form-data POST requests where multiple files are expected to be processed *and* form data should be formatted as a JSON string (*as a CLOB data type*)[[1]](#footnote-22).

When the:body\_json implicit parameter is included for Resource Handlers that process multiple files, the :body\_json implicit parameter must be invoked. The :body\_json parameter can be invoked in various ways, such as:

* The DBMS\_OUTPUT package such as dbms\_output.put\_line(:body\_json);
* The hypertext procedures (htp) and functions (htf) packages, such as in htp.print(:body\_json);
* Assigning the :body\_json implicit parameter as variable, e.g. l\_body\_json := :body\_json;

**NOTE:** There is no specific requirement to assign :body\_json to a local variable. Similarly, there is no requirement to re-use the local variable, should you choose to assign one.

#### Example

A table (DEMO\_TABLE) has been created with the following attributes:

CREATE TABLE ORDSDEMO.DEMO\_TABLE   
 (   
 ID NUMBER (\*,0) GENERATED BY DEFAULT AS IDENTITY   
 ( START WITH 1 CACHE 20 ) NOT NULL ,   
 FILE\_NAME VARCHAR2 (200) ,   
 FILE\_BODY BLOB ,   
 CONTENT\_TYPE VARCHAR2 (200) ,   
 FILE\_VISIBILITY VARCHAR2 (10) ,   
 SUBMITTED\_BY VARCHAR2 (200) ,   
 SUBMITTED\_ON TIMESTAMP DEFAULT systimestamp   
 )   
 TABLESPACE DATA   
 LOGGING   
;

**NOTE:** Columns such as FILE\_VISIBILITY, SUBMITTED\_BY, and SUBMITTED\_ON are for *demonstration purposes only*. They are not required.

An ORDS Endpoint has been created (with the below Resource Handler code).

* The endpoint expects multiple files and form data *in a JSON format* (i.e., the use of the :body\_json implicit parameter).
* The ORDS.BODY\_FILE\_COUNT function will be used to count the total files in the POST request.
* The ORDS.GET\_BODY\_FILE procedure will be used to store, in session, the files.

The following code example then performs an INSERT on the DEMO\_TABLE and relies upon various HTP procedures to “print” the results to a user, client, or application.

DECLARE  
 L\_PARAMETER\_NAME VARCHAR2(4000);  
 L\_FILE\_NAME VARCHAR2(4000);  
 L\_CONTENT\_TYPE VARCHAR2(200);  
 L\_FILE\_BODY BLOB;  
 L\_FILE\_VISIBILITY CLOB;  
 L\_SUBMITTED\_BY CLOB;  
BEGIN  
 L\_SUBMITTED\_BY := JSON\_VALUE(:BODY\_JSON, '$.submitted\_by');  
 L\_FILE\_VISIBILITY := JSON\_VALUE(:BODY\_JSON, '$.file\_visibility');  
 HTP.PARAGRAPH;  
 HTP.PRINT('Submitted By: ' || L\_SUBMITTED\_BY);  
 HTP.BR;  
 HTP.PARAGRAPH;  
 HTP.PRINT('File visibility status: ' || L\_FILE\_VISIBILITY);  
 HTP.BR;  
 FOR I IN 1..ORDS.BODY\_FILE\_COUNT LOOP  
 ORDS.GET\_BODY\_FILE(  
 P\_FILE\_INDEX => I,  
 P\_PARAMETER\_NAME => L\_PARAMETER\_NAME,  
 P\_FILE\_NAME => L\_FILE\_NAME,  
 P\_CONTENT\_TYPE => L\_CONTENT\_TYPE,  
 P\_FILE\_BLOB => L\_FILE\_BODY  
 );  
 INSERT INTO DEMO\_TABLE (  
 FILE\_NAME,  
 FILE\_BODY,  
 CONTENT\_TYPE,  
 FILE\_VISIBILITY,  
 SUBMITTED\_BY  
 ) VALUES ( L\_FILE\_NAME,  
 L\_FILE\_BODY,  
 L\_CONTENT\_TYPE,  
 L\_FILE\_VISIBILITY,  
 L\_SUBMITTED\_BY );  
  
 HTP.PARAGRAPH;  
 HTP.PRINT('Inserted File: ' || L\_FILE\_NAME);  
 END LOOP;  
END;

To test this :body\_json implicit parameter a curl command such as the one below may be used:

curl --location 'https://localhost:8080/ords/ordsdemo/demo\_api/demo' \  
--form 'file\_one=@"/Users/file\_one.txt"' \  
--form 'file\_two=@"/Users/ile\_two.txt"' \  
--form 'submitted\_by="chris"' \  
--form 'file\_visibility="public"'

### 3.1.3 About the :content\_type Parameter

The :content\_type implicit parameter provides the value of the Content-Type request header supplied with the request. If no Content-Type header is present in the request, then a null value is returned.

### 3.1.4 About the :current\_user parameter

The :current\_user implicit parameter provides the identity of the user authenticated for the request.

Note:In a scenario, where the user is not authenticated, the value is set to null. For example, if the request is for a public resource, then the value will be set to null.

### 3.1.5 About the :status\_code parameter

The :status\_code implicit parameter enables a resource handler to indicate the HTTP status code value to include in a response. The value must be one of the numeric values defined in the HTTP Specification document.

### 3.1.6 About the :forward\_location parameter

The :forward\_location implicit parameter provides a mechanism for PL/SQL based resource handlers to produce a response for a request.

Consider a POST request that results in the creation of a new resource. Typically, the response of a POST request for REST APIs contains the location of the newly created resource (in the Location response header) along with the representation of the new resource. The presence of the Location header in the response indicates that there must be a GET resource handler that can produce a response for the specified location.

Instead of applying logic to the POST resource handler to render the representation of the new resource in the response, the resource handler can delegate that task to the existing GET Resource Handler.

The following resource handler defines a POST handler that delegates the generation of the response to a GET resource handler:

ords.define\_handler( p\_module\_name => ‘tickets.collection’, p\_pattern => ‘.’,  
p\_method => ‘POST’, p\_mimes\_allowed => ‘application/json’, p\_source\_type => ords.source\_type\_plsql, p\_source => ’ declare l\_owner varchar2(255); l\_payload clob; l\_id number; begin l\_payload := :body\_text; l\_owner := :current\_user; l\_id := ticket\_api.create\_ticket( p\_json\_entity => l\_payload, p\_author => l\_owner ); :forward\_location := ’‘./’’ || l\_id; :status\_code := 201; end; ’ );

Where:

The ords.define\_handler API is used to add a POST handler to an existing resource module named tickets.collection.  
  
The p\_pattern with value '.' indicates that the POST handler should be bound to the root resource of the resource module. If the base path of the tickets.collection' is /tickets/, then the POST handler is bound to the /tickets/ URL path.  
  
The p\_mimes\_allowed value indicates that the POST request must have a Content-Type header value of application/json'.  
  
The p\_source\_type value indicates that the source of the POST handler is a PL/SQL block.  
  
The p\_source value contains the source of the PL/SQL block:  
  
Where:  
  
Note:The :body\_text implicit parameter is assigned to a local variable, so that it can be dereferenced more than once.  
  
 The identity of the user, making the POST request, is determined from the :current\_user implicit parameter.  
  
 The PL/SQL block, delegates the task of storing the request payload to a PL/SQL package level function. The PL/SQL block should only contain logic to bridge from the HTTP request to the PL/SQL package invocation.  
  
 Note:When all the data modification operations are wrapped in a PL/SQL API, the PL/SQL block can be independently unit tested. Long and complicated PL/SQL blocks are an anti-pattern indicative of code that is difficult to test and maintain.  
  
 The PL/SQL package level function returns the ID of the newly created resource.  
  
 The :forward\_location implicit parameter is assigned the value of './' || l\_id. For example, if the value of l\_id is 4256, then the value of :forward\_location is /tickets/4256 .  
  
 When ORDS evaluates the preceding PL/SQL block and checks the value assigned to the :forward\_location implicit parameter, it initiates a GET request against the specified location (for example, /tickets/4256) and return the response generated by the GET request as the response of the POST request. In addition, ORDS includes a location response header with the fully resolved URL of the :forward\_location value.  
  
 The :status\_code implicit parameter is assigned the HTTP response status code value. The 201 (Created) status code indicates that a new resource is created. This value will override the status code generated by the GET request.

### 3.1.7 About the pagination implicit parameters

The following table lists the pagination implicit parameters:

Note:Oracle REST Data Services reserves the use of the query parameters, page, offset, and limit. It is not permitted to define REST services that use named bind parameters with any of the preceding query parameter names. Alternatively, REST services must use the appropriate pagination implicit parameters defined in the following table:

Table 3-2 Pagination Implicit Parameters Name Description Status

:page\_offset

Specifies the zero based page offset in a pagination request.

Deprecated

:page\_size

Specifies the maximum number of rows to be retrieved on a page.

Deprecated

:row\_offset

Specifies the index of the first row to be displayed in a pagination request.

Not Recommended

:row\_count

Specifies the index of the last row to displayed in a pagination request.

Not Recommended

:fetch\_offset

Specifies the zero based index of the first row to be displayed on a page.

Recommended

:fetch\_size

Specifies the maximum number of rows to be retrieved on a page.

Recommended

#### 3.1.7.1 About the :page\_offset parameter

The :page\_offset implicit parameter is provided for backward compatibility, so it is used only with source\_type\_query source type resource handlers.

Note:

The source\_type\_query source type is deprecated, instead use the source\_type\_collection feed parameter.  
  
The :page\_offset implicit parameter is deprecated, instead use the :row\_offset implicit parameter.

#### 3.1.7.2 About the :page\_size parameter

The :page\_size implicit parameter is used to indicate the maximum number of rows to be retrieved on a page. :page\_size parameter is provided for backward compatibility. This parameter is deprecated, instead use :fetch\_size implicit parameter.

#### 3.1.7.3 About the :row\_offset parameter

The :row\_offset implicit parameter indicates the number of the first row to be displayed on a page. The :row\_offset implicit parameter is used when you are using both a wrapper pagination query and row\_number() (used in Oracle 11g and earlier releases). Starting Oracle 12c or later releases, Oracle recommends using the :fetch\_offset implicit parameter and a row limiting clause instead of the :row\_offset parameter.

#### 3.1.7.4 About the :row\_count parameter

The :row\_count implicit parameter is used to indicate the number of rows to be displayed on a page. The :row\_count value is the value of the sum of :row\_offset and the pagination size. The :row\_count implicit parameter is useful when implementing pagination using a wrapper pagination query and row\_number()method that was used in Oracle database 11g and earlier releases. Starting Oracle Database release 12c or later, Oracle recommends that you use :fetch\_size parameter and a row limiting clause instead.

#### 3.1.7.5 About the :fetch\_offset parameter

The :fetch\_offset implicit parameter is used to indicate the zero based offset of the first row to display in a given page. The :fetch\_offset implicit parameter is used when you implement pagination using a row limiting clause, which is recommended for use with Oracle 12c and later releases.

#### 3.1.7.6 About the :fetch\_size parameter

The :fetch\_size implicit parameter is used to indicate the maximum number of rows to retrieve on a page. ORDS always sets the value of :fetch\_size to the pagination size plus one. The presence or absence of the extra row helps ORDS in determining if there is a subsequent page in the results or not.

Note:The extra row that is queried is never displayed on the page.

#### 3.1.7.7 About Automatic pagination

This section describes the automatic pagination process. If a GET resource handler source type, source\_type\_collection\_feed or source\_type\_query has a non zero pagination size (p\_items\_per\_page) and the source of the GET resource handler does not dereference any of the implicit pagination parameters discussed in the preceding sections, then ORDS automatically wraps the query in a pagination clause to constrain the query results to include only the values from the requested page. With automatic pagination, the resource handler author needs to specify only the pagination size, and ORDS automatically handles the remaining effort in paginating the resource.

Note:All resource modules have a default pagination size (p\_items\_per\_page) of 25. So, by default automatic pagination is enabled.

#### 3.1.7.8 About Manual pagination

This section describes the manual pagination process. In some scenarios, a GET resource handler needs to perform pagination on its own rather than delegating the pagination process to ORDS. In such cases, the source of the GET resource handler will dereference one or more implicit pagination parameters discussed in the preceding sections.

Note:The GET resource handler must specify the desired pagination size so that ORDS can correctly calculate the required values for the implicit pagination parameters.

Examples

Manual pagination example using row limiting clause

The following example defines a REST service that uses a row limiting clause to paginate the query result set. This is the recommended way to implement manual pagination:

begin ords.define\_service( p\_module\_name => ‘example.paging’, p\_base\_path => ‘/example/’, p\_pattern => ‘/paged’, p\_items\_per\_page => 7, p\_source => ‘select \* from emp e order by empno desc offset :fetch\_offset rows fetch next :fetch\_size rows only’ ); commit; end;

Manual pagination example using row\_number() method

The following example defines a REST service that uses a wrapper query and row\_number() method. This approach is not recommended.

begin ords.define\_service( p\_module\_name => ‘example.paging’, p\_base\_path => ‘/example/’, p\_pattern => ‘/paged’, p\_items\_per\_page => 7, p\_source => ‘select \* from (select q\_.\* , row\_number() over (order by 1) rn\_\_ from (select \* from emp e order by empno desc) q\_ )where rn\_\_ between :row\_offset and :row\_count’ ); commit; end;

1. Although you can store JSON in the Oracle database as JSON, VARCHAR2, CLOB, and BLOB, ORDS uses the CLOB data type, to ensure backward compatibility with earlier releases of the Oracle database. [↑](#footnote-ref-22)