# 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 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 |

**NOTE:** the :body, :body\_text, and :body\_json parameters are not designed to be used in the *same* Resource Handler. Review the following sections and decide which of these three bind parameters best fits your use case.

### 3.1.1 Automatic binding

ORDS also supports, under various conditions, automatic binding of the following:

* Query parameters (*all conditions*)
* Form data
* JSON objects

When query parameters are provided, they are always automatically bound by Resource Handlers. Whereas automatic binding behavior of form data and JSON objects are dependent on the following two factors:

1. Where and how the :body, :body\_text, and :body\_json implicit parameters are used, *and*
2. The media- or MIME type used:
   * application/x-www-form-urlencoded
   * application/json
   * multipart/form-data
     + *with a single file*
     + *with multiple files*

**NOTE:** Sections **3.1.1 About the :body parameter**, **3.1.2 About the :body\_text parameter**, and **3.1.3 About the :body\_json parameter** will cover in detail automatic binding behavior under various conditions.

#### Examples

##### Query Parameters

ORDS supports automatic binding of query parameters for POST requests with all Content Types (i.e., application/x-www-form-urlencoded, application/json, multipart/form-data - *with a single file,* multipart/form-data - *with multiple files*).

An HTTP request is issued:

https://localhost:8443/ords/my\_schema/demo/etc?shape=triangle

The value, triangle, would be accessible in an ORDS handler with the automatic bind :shape. As can be seen in the example PL/SQL Handler code:

Begin  
 HTP.p('RESULT: ' || :shape);  
End;  
  
RESULT: triangle

##### Form Data

ORDS supports automatic binding of POST request body form data under various conditions. For detailed guidance, refer to the :body, :body\_text, and :body\_json sections of this document. For illustrative purposes, the following example assumes a POST request is being issued to an ORDS Resource Handler with *none* of the previously mentioned “:body\_” implicit parameters.

An HTTP request is issued (in the form of a curl command):

curl 'https://localhost:8443/ords/my\_schema/demo/etc'  
 --header 'Content-Type: application/x-www-form-urlencoded'  
 --data-url-encode 'last\_name=Ever'  
 --data-url-encode 'first\_name=Greatest'

The values for last\_name and first\_name would be accessible in an ORDS handler with the automatic binds :last\_name and :first\_name. As can be seen in the example PL/SQL Handler code:

Begin  
 HTP.p('Hello: ' || :first\_name || :last\_name);  
End;  
  
Hello: Greatest Ever

**NOTE:** Refer to the :body, :body\_text, and :body\_json sections of this document for detailed guidance on when automatic binding of form data can be utilized.

##### JSON items

ORDS supports automatic binding of a JSON object in POST requests when the following conditions have been met:

1. The Content-Type is of application/json, *and*
2. None of the following implicit bind parameters are used in the Resource Handler: :body, :body\_text, :body\_json.

An HTTP request is issued (in the form of a curl command):

curl 'https://localhost:8443/ords/my\_schema/demo/etc'  
 --header 'Content-Type: application/json'  
 --data '{username: "clark", "password: "superman1234"}'

The values for username and password would be accessible in this ORDS handler with the automatic binds :username and :password. As can be seen in the example PL/SQL Handler code:

Begin  
 HTP.p('Hello: ' || :username);  
 Htp.p('Your password: ' || :password);  
End;  
  
Hello: clark  
Your password: superman1234

### 3.1.3 About the :body\_json parameter

The :body\_json implicit parameter can be used with POST Resource Handlers to receive the contents of a request body as a JSON object for application/x-www-form-urlencoded, application/json, and multipart/form-data Content types. This allows Resource Handlers to directly reference JSON properties (i.e., {"key": "value"} pairs).[[1]](#footnote-25)

Additionally, the :body\_json implicit parameter can be used in multipart/form-data POST requests that *may* include one or more files as well as an accompanying JSON object (e.g., as is the case with HTML form data). Form data, bound to the :body\_json implicit parameter, continues to be received as a JSON object while files can be processed with the ORDS.BODY\_FILE\_COUNT LOOP function and the ORDS.GET\_BODY\_FILE procedure.

Similar to the :body and :body\_text implicit parameters, when the :body\_json implicit parameter is included in a Resource Handler, **it must be invoked and dereferenced**, in order to be used. The :body\_json parameter can be invoked in various ways. Some examples:

* 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;

#### Example

A table (BODY\_JSON\_DEMO\_TABLE) has been created with the following attributes:

![BODY\_JSON\_DEMO\_TABLE columns image.](./images/3.1-demo-table-in-sql-worksheet-image-1.png ” “)

CREATE TABLE BODY\_JSON\_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,  
 SHAPE VARCHAR2(20)  
);

**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) with the following considerations:

* 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 of the POST request.
* The ORDS.GET\_BODY\_FILE procedure will be used to temporarily store (in the current database session’s memory) file names, details, and contents. Which allows the ORDS Resource Handler to “handle” multiple files in a single POST request.

![The body-json PLSQL Handler code.](./images/3.1-body-json-demo-plsql-handler-image-2.png ” “)

The following Resource Handler code example then performs an INSERT on the BODY\_JSON\_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\_BODY\_JSON CLOB;  
BEGIN  
 L\_BODY\_JSON := :BODY\_JSON;  
 HTP.PARAGRAPH;  
 HTP.PRINT('Submitted by: ' || JSON\_VALUE(L\_BODY\_JSON, '$.submitted\_by'));  
 HTP.BR;  
 HTP.PARAGRAPH;  
 HTP.PRINT('File visibility status: ' || JSON\_VALUE(L\_BODY\_JSON, '$.file\_visibility'));  
 HTP.BR;  
 HTP.PARAGRAPH;  
 HTP.PRINT('Shape selected: ' || :shape);  
 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  
 );  
 HTP.PARAGRAPH;  
 HTP.PRINT('Inserted file #' || i || ': ' || L\_FILE\_NAME);  
 HTP.BR;  
 INSERT INTO BODY\_JSON\_DEMO\_TABLE (  
 FILE\_NAME,  
 FILE\_BODY,  
 CONTENT\_TYPE,  
 FILE\_VISIBILITY,  
 SUBMITTED\_BY,  
 SHAPE  
 ) VALUES ( L\_FILE\_NAME,  
 L\_FILE\_BODY,  
 L\_CONTENT\_TYPE,  
 JSON\_VALUE(L\_BODY\_JSON, '$.submitted\_by'),  
 JSON\_VALUE(L\_BODY\_JSON, '$.file\_visibility'),  
 :shape );  
 END LOOP;  
END;

Click to reveal the complete Resource Module.

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

**NOTE:** You may have observed the included query parameter in the above POST request. In this example, we illustrate how automatic binding of query parameters (e.g., shape=triangle can be used in ORDS POST Resource Handlers).

curl --location 'https://localhost:8443/ords/ordsdocs/binds/body\_json\_demo?shape=triangle' \  
--form 'files=@"demo-3.sql"' \  
--form 'files=@"demo-2.sql"' \  
--form 'submitted\_by="chris"' \  
--form 'file\_visibility="public"'

Accordingly, a client may respond with the following:

<p>  
Submitted By: chris  
<br />  
<p>  
File visibility status: public  
<br />  
<p>  
Shape: triangle  
<p>  
Inserted File: demo-3.sql  
<br />  
<p>  
Inserted File: demo-2.sql  
<br />

![Example curl command output.](./images/3.1-body-json-curl-command-example-image-3.png ” “)

An API testing tool, such as Postman can be used as well:

![A Postman example.](./images/3.1-body-json-postman-example-image-4.png ” “)

Querying the target database reveals the updates from these tests:

![Test results in the BODY\_JSON\_DEMO\_TABLE](./images/3.1-table-results-from-body-json-tests-image-5.png ” “)

1. In a scenario such as this, the form data in the POST body is sent as a JSON object and then handled as a CLOB data type in the Oracle database. While *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-25)