PL/JSON Reference Guide (version 0.8.0)

For Oracle 10g and 11g

Jonas Krogsbøll

PURPOSE

The goal of PL/JSON is to create a correct implementation of JSON to use in a PL/SQL environments. The oracle object syntax is used because we want to be able to store JSON-objects in tables, without using character lobs. Parsing JSON-text and manual construction of JSON-objects should be easy as well as fetching data from the constructed objects should be. PL/JSON is delivered AS IS and we make no garantee for anything (although we think that the product is safe to use).

DESCRIPTION

The parser and pretty-printer is implemented in two packages, but you'll never need to access these directly. The essential objects are linked to the relevant functions in the packages (in constructors and output methods). The two objects you should use are JSON and JSON_LIST. You can consider the JSON object to be a mapping of strings to values and JSON_LIST to be a list of values. The values that can be stored are the four primitives: JSON_BOOL, JSON_NULL, VARCHAR2 and NUMBER, plus JSON_LIST and JSON as nested structures (see figure 1).

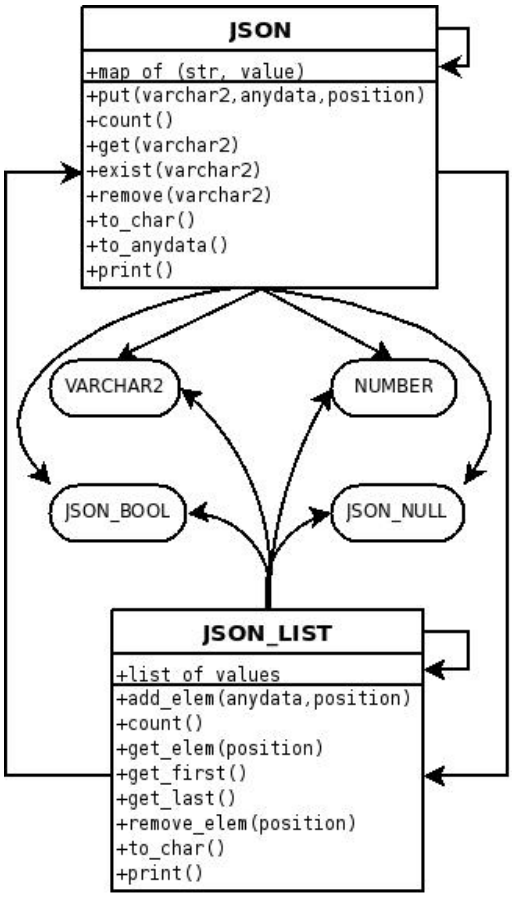

Figure 1: Essential Objects in PL/JSON

IN THE RELEASE

- Install script
- Uninstall script.
- 8 new oracle types ready to use in your database.
- 3 packages (parser, printer and extension)
- A few examples files - does not modify the database.
- Some testing script - creates and delete a table (JSON_TESTSUITE)

Going from 0.6.2 to 0.8.0?

Where did 0.7 go? Well, almost the entire code was rewritten. The API is very different now, so you cannot perform a simple upgrade from 0.6.2 to 0.8.0. Now there is no "to be implemented" methods. Additional methods (toClob, toRAW, etc.) will be added when they are ready and properly tested. The author of JSON thinks thats the way to go too:

> We are finding that people like products that just work. It turns out that designs that just work are much harder to produce than designs that assemble long lists of features.
> - Douglas Crockford (JavaScript: The Good Parts page 99)

KNOWN LIMITATIONS

People hate limitations, but frustration kicks in when the limitations are unknown or undescribed.

- The maximum input size the parser can accept is 32000 characters.
- The pretty-printer limits it's output to 32676 characters (extra space added for line breaks and spaces).
- Each string (pair-name or value) is limited to 4000 characters.
- The number parsing assumes that oracles number type can contain the input (in most cases it can).

Example to workaround the pretty-printer size limit

Suppose we want to build a large json-list and print it with dbms_output.put_line:

```
declare
  obj json := json('{
    "test": true,
    "array": [1,2,3]}');
  thelist json_list := json_list();
begin
  for i in 1 .. 300 loop
    obj.put('Loop counter', i);
    thelist.add_elem(obj.to_anydata);
  end loop;

  --printing starts
  dbms_output.put('[');
  for i in 1 .. thelist.count loop
    obj := json.to_json(thelist.get_elem(i));
    dbms_output.put(obj.to_char);
    if(i != thelist.count) then
      dbms_output.put(', ');
    end if;
  end loop;
  dbms_output.put_line(']');
end;
```

Now the limitation will be the size of the dbms_output buffer.

GETTING STARTED

To get started on using this product, you should first install the product with the install script and then take a look at the examples in the *examples* folder. The content of each example file is:

- **ex1.sql**: Simple creation of a JSON object.
- **ex2.sql**: Simple creation of a JSON list.
- **ex3.sql**: Working with parser exceptions.
- **ex4.sql**: Building an object with the API.
- **ex5.sql**: Building a list with the API.
- **ex6.sql**: Working with variables as copies.
- **ex7.sql**: Using the extension package.
- **sqlplus_run.sql**: Displaying examples from www.json.org/example.html

EXTENSION PACKAGE

The extension package makes it easier to work with the values in a JSON object or a JSON_LIST. The values are all stored as ANYDATA, but can be converted back to a useful type with the static functions in the JSON object type. To know what conversion function you should use, you can test the value with the type testing functions in the JSON_EXT package. The extension package also delivers a way of using dates in JSON, but you might ask why that functionality isn't a part of the object types.
The core implementation strictly follows the specification (RFC 4627 - http://www.ietf.org/rfc/rfc4627.txt?number=4627) on JSON. Furthermore the design goals with JSON were to, quote: "be minimal, portable, textual and a subset of JavaScript. The less we need to agree on in order to interoperate, the more easily we can interoperate" - D.Crockford. I, for instance, like dates/timestamp to be represented as milliseconds since 1st of January 1970 at 00:00:00 UTC. That is compact, but in a oracle context that isn't exactly the ideal choice.

TESTSUITE

Any proper product is tested for correctness. So should PL/JSON be with a testsuite that can be executed without installing any additional software on your database. You properly don't need the testsuite, but if you modify the implementation, adds more features, tests will be needed. Also if you discover a bug, you could report the bug by writing a relevant testcase.

CONTRIBUTING

Write to us in the forums of sourceforge.net. We will be happy to hear from you.

Q: "I've added a lot of code, please merge my changes"
A: Hmmm - it's not that we don't appriciate your work, but we would really prefer that you wrote tests and documentation to each feature - otherwise new code could easily break functionality.

Q: "I've added some changes and I might contribute them to the project, but what's in it for me?"
A: This is not GPL, so you can keep your changes if you want to. When you are contributing then more eyes will look at your code. Possible errors might get detected and corrected and new features may arrise from your features - making it a better product you can use.

FUTURE PLANS

Maybe an XPATH way of accessing and storing values will be implemented.
Additional json-list or json operations such at concat, substract, join, union, etc.
Enhancing the parser and printer to handle larger strings than 32K.
Test/example of storing json in a table.

TYPES

Example of usage can be found in the *example* folder. The important object types installed are JSON, JSON_List, JSON_BOOL and JSON_NULL.

TYPE NAME	JSON
CONSTRUCTOR	json()
Returns an empty JSON-object	
CONSTRUCTOR	json(varchar2)
Parses an json-text and return a JSON-object. Throws ORA-20101 and ORA-20100 in case of an faulty json-text.	
MEMBER PROCEDURE	put(varchar2, anydata, position) put(varchar2, varchar2, position) put(varchar2, number, position) put(varchar2, json_bool, position) put(varchar2, json_null, position) put(varchar2, json_list, position) put(varchar2, json, position)
Adds a pair to the JSON-object. If the pair-name is allready present, then the value is replaced. If position is null (default) or larger than the number of pairs, the pair is inserted as a new last pair. If position ¡ 2, the pair is inserted as a new first pair. Otherwise the pair is inserted at the specified position (list is 1-indexed).	
MEMBER FUNCTION	count
Returns the amount of pairs in the JSON-object (not counting nested object pairs).	
MEMBER FUNCTION	get(varchar2)
Returns the pair-value as ANYDATA if the pair exists (NULL otherwise). Convert ANYDATA to a useable object with the static functions.	
MEMBER FUNCTION	exist(varchar2)
Returns TRUE if the pair exists, FALSE otherwise	
MEMBER PROCEDURE	remove(varchar2)
Removes a pair from the JSON-object	
MEMBER FUNCTION	to_char
Pretty-prints the JSON-object into a VARCHAR2	
MEMBER PROCEDURE	print
Outputs to_char with dbms_output.put_line.	
MEMBER FUNCTION	to_anydata
Convert the JSON-object into an ANYDATA value. Useful when inserting a JSON-object in a list.	
STATIC FUNCTION	to_json
ANYDATA to JSON	
STATIC FUNCTION	to_json_list
ANYDATA to JSON_LIST	
STATIC FUNCTION	to_json_bool
ANYDATA to JSON_BOOL	
STATIC FUNCTION	to_json_null
ANYDATA to JSON_NULL	
STATIC FUNCTION	to_number
ANYDATA to NUMBER	
STATIC FUNCTION	to_varchar2

ANYDATA to VARCHAR2

TYPE NAME	JSON_LIST
CONSTRUCTOR	json_list()
Returns an empty JSON_LIST-object	
CONSTRUCTOR	json_list(varchar2)
Parses an json-text and return a JSON_LIST-object. Throws ORA-20101 and ORA-20100 in case of an faulty json-text.	
MEMBER PROCEDURE	add_elem(anydata, position) add_elem(varchar2, position) add_elem(number, position) add_elem(json_bool, position) add_elem(json_null, position) add_elem(json_list, position) add_elem(json, position)
Adds a value to the JSON_LIST-object. If position is null (default) or larger than the length of the list, the value is inserted as a new last. If position ¡ 2, the value is inserted as a new first. Otherwise the value is inserted at the specified position (note the the list is 1-indexed).	
MEMBER FUNCTION	count
Returns the amount of values in the JSON_LIST-object (not counting nested list).	
MEMBER FUNCTION	get_elem(position)
Returns the as ANYDATA if the value exists (NULL otherwise). Convert ANYDATA to a useable object with the JSON static functions.	
MEMBER FUNCTION	get_first
= list.get_elem(1)	
MEMBER FUNCTION	get_last
= list.get_elem(list.count)	
MEMBER FUNCTION	remove_elem(position)
Remove the entry if it exists (assures that there are no ̈holesïn the list).	
MEMBER FUNCTION	remove_first
= list.remove_elem(1)	
MEMBER FUNCTION	remove_last
= list.remove_elem(list.count)	
MEMBER FUNCTION	to_char
Pretty-prints the JSON_LIST-object into a VARCHAR2	
MEMBER PROCEDURE	print
Outputs to_char with dbms_output.put_line.	

TYPE NAME	JSON_BOOL
CONSTRUCTOR	json_bool(boolean)
Returns an JSON_BOOL-object (TRUE or FALSE)	
MEMBER FUNCTION	to_char
'true' or 'false'	
MEMBER FUNCTION	is_true
TRUE or FALSE	
MEMBER FUNCTION	is_false

FALSE or TRUE	
STATIC FUNCTION	maketrue
json_bool(TRUE)	
STATIC FUNCTION	makefalse
json_bool(FALSE)	

TYPE NAME	**JSON_NULL**
CONSTRUCTOR	json_bool(boolean)
Returns an JSON_BOOL-object (TRUE or FALSE)	
MEMBER FUNCTION	to_char
'true' or 'false'	
MEMBER FUNCTION	is_true
TRUE or FALSE	
MEMBER FUNCTION	is_false
FALSE or TRUE	
STATIC FUNCTION	maketrue
json_bool(TRUE)	
STATIC FUNCTION	makefalse
json_bool(FALSE)	

Test environments

Tested on Oracle Database 10g Express Edition 10.2.0.1.0 @ Ubuntu 9.04 (x86)
Tested on Oracle Database 10gR2 Enterprise Edition 10.2.0.2.0 @ Windows Server 2003 (x64)