JSON Object

DireXions 2025 – Pathways to Progress

Agenda

Overview

Methods

Loading and Saving Methods

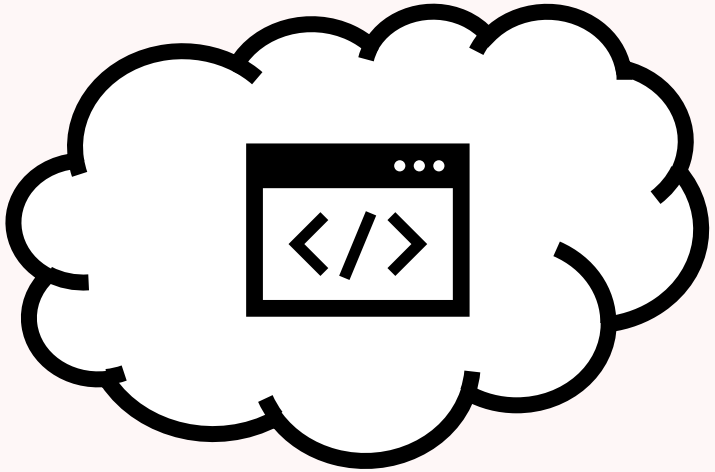
Coming in PxPlus 2025 Update 1

XML Enhancements

Performance

Demo

```
{
  "Customer Name":"John Doe",
  "Date":"01/01/2025",
  "Location":{
    "City":"Toronto"
  },
  "Order":{
    "Amount":10,
    "ID":12345,
    "Name":"Apple",
    "Unit Price":1.85
  },
  "Total":18.5
}
```

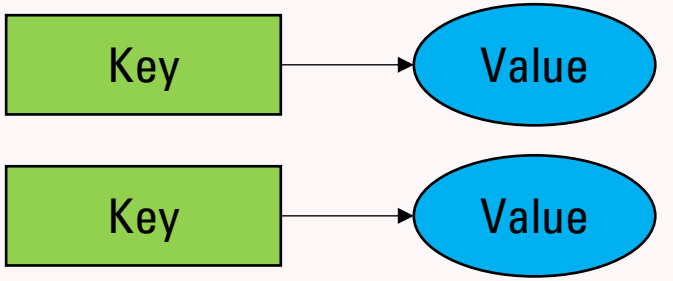


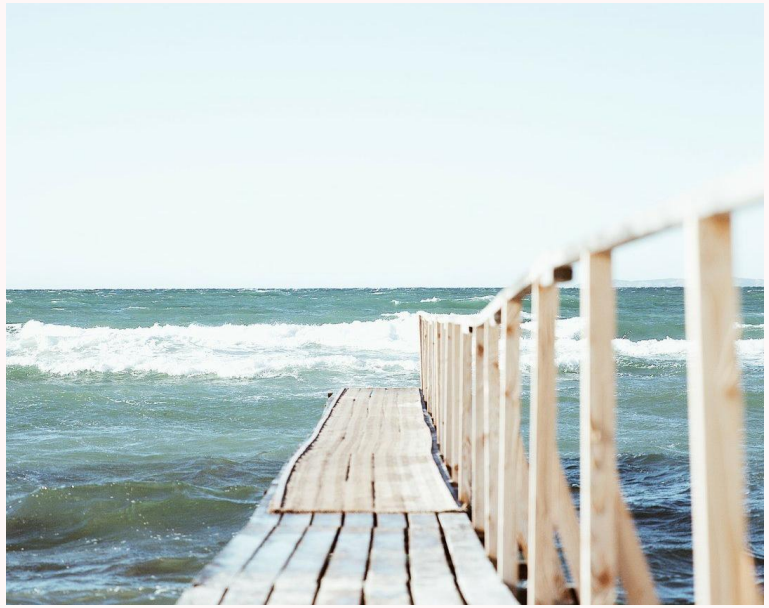
JSON

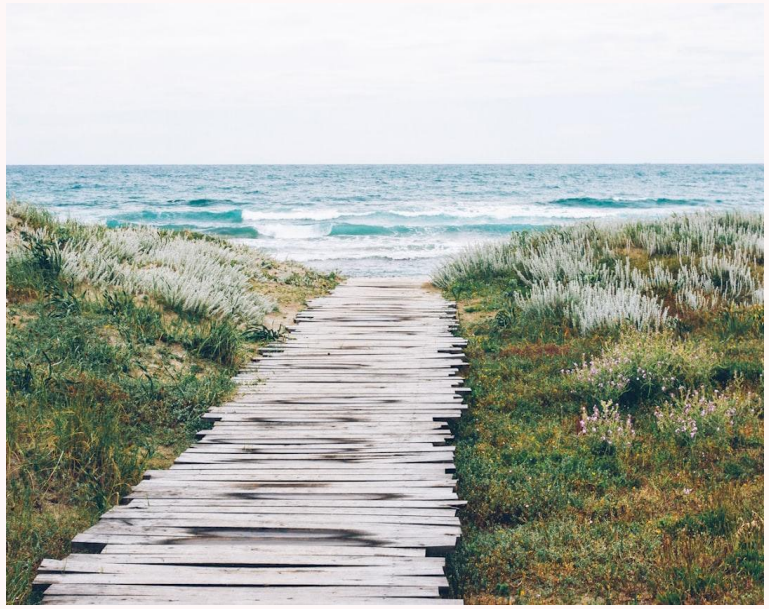
JavaScript Object Notation

Commonly used in web applications and APIs

Key-Value pairs







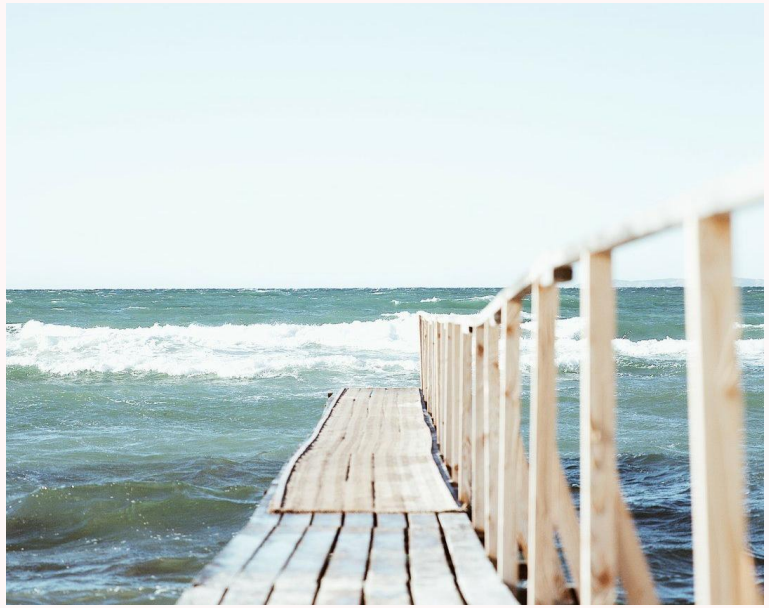
JSON in PxPlus

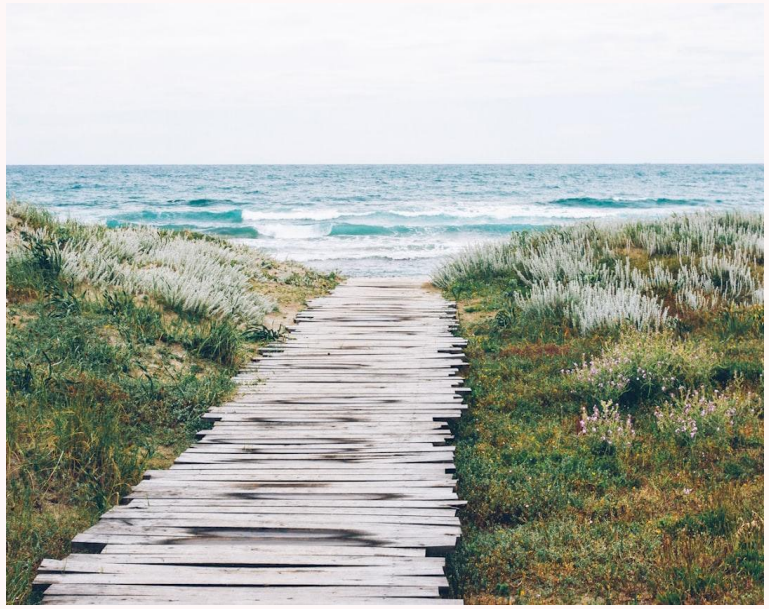
Associative Array Example:

```
dim my_array$

my_array$["Person.FirstName"] = "Jane"
my_array$["Person.LastName"] = "Smith"
my_array$["StreetNum"] = 456
my_array$["StreetName"] = "Ontario Street"

Print dim(list edit my_array${ALL})
```





JSON Object Overview

The new JSON object is easy-to-use and enables you to intuitively work with JSON data.

The JSON object behaves and functions similarly to the associative array but also comes with many functions to make working with your data easier.

Invocation:

```
Json_obj = NEW("*obj/json")
```

Methods (Appends)

Method	Description
Append(*key$, value***)** **Append(***key$, value$***)**	Appends a key with a numeric or string value to the JSON object, if the given key does not exist.
Append_to(*key$, key2$, value***)** **Append_to(***key$, key2$, value$***)**	Appends *key2* as a numeric or string value to the given key of the JSON object, if key.key2 does not exist.
Append_json(*jsonObj***)**	Appends all the elements from the given JSON object.
Append_json_to(*key$, jsonObj***)**	Appends the elements of the given JSON object to a key, if the key does not exist.

Append Example

Example:

```
Employee_obj = NEW("*obj/json")

Employee_obj'append("Name", "John Smith")

Employee_obj'append_to("Address", "Street", "32 Avenue")
```

JSON data is automatically sorted since the system parameter JV = 2 (Default)

JSON Data:

```
{
  "Address": {
    "Street": "32 Avenue"
  },
  "Name": "John Smith"
}
```

Methods (Gets)

Method	Description
Get_str$(***key$*)** **Get_str$(***key$, no_validation***)**	Returns a string from the key. Returns err 11 if key not found. *no_validation*=1 will ignore validation.
Get_num(*key$***)** **Get_num(***key$, no_validation***)**	Returns a numeric from the key. Returns err 11 if key not found. *no_validation*=1 will ignore validation.
Get_json(*key$***)** **Get_json(***key$, full_key_flag***)**	Returns a JSON object of the given key. Keys will use the full key name when *full_key_flag*=1.
Get_key(*index***)**	Returns the key at an index in the object.

Get Example

Example:

```
Print Employee_obj'get_str$("Name")
```

Output:

```
John Smith
```

Example:

```
New_obj = Employee_obj'get_json("Address")
```

New_obj:

```
{
  "Street": "32 Avenue"
}
```

Methods (Sets)

Method	Description
Set(*key$, value***)** **Set(***key$, value$***)**	Sets the key with a numeric or string value.
Set_json(*key$, jsonObj***)**	Sets the key to the elements in the given JSON object. **Warning! Self-referencing will result in an Error #91: Class/Object in use.**
Force_set(*key$, value***)** **Force_set(***key$, value$***)** **Force_set(***key$, value, ignore_remove***)** **Force_set(***key$, value$, ignore_remove***)**	Sets value without key validation. *ignore_remove*=1 will ignore removing old values.

Set Example

Example:

```
Employee_obj'set("Name", "Jane Doe")
```

JSON Data:

```
{
  "Address": {
    "Street": "32 Avenue"
  },
  "Name": "Jane Doe"
}
```

Set Example

Example:

```
Employee_obj'set("Address", "32 Avenue")
```

JSON Data:

```
{
  "Address": "32 Avenue",
  "Name": "Jane Doe"
}
```

(Bad) Set Example

Example:

```
Employee_obj'force_set("Address", "32 Avenue", 1)
```

JSON Tags:

```
[
  "Address.Street",
  "Address",
  "Name"
]
```

JSON Data:

```
{
  "Address": {
    "Street": "32 Avenue"
  },
  "Address": "32 Avenue",
  "Name": "Jane Doe"
}
```

Methods (Strings)

Method	Description
String$()	Returns a string of the JSON data.
String$(***key$*)**	Returns a string of the JSON data at the given key.
String$(*sort_flag***)**	Returns a string of the sorted JSON object when *sort_flag*=1.
String$(***key$, sort_flag*)**	Returns a string of the sorted JSON object from the given key when *sort_flag*=1.

Note: This string is automatically sorted when the system parameter JV=2 (Default)

String Example

Example:

```
print Employee_obj'string$()
```

Output:

```
{
  "Address": {
    "Street": "32 Avenue"
  },
  "Name": "Jane Doe"
}
```

Methods (Prints)

Method	Description
Print()	Prints the JSON string.
Print(*key$***)**	Prints the JSON string at the given key.
Print(*sort_flag***)**	Prints the sorted JSON string when *sort_flag*=1.
Print(*key$, sort_flag***)**	Prints the sorted JSON string at the given key when *sort_flag*=1.

Print Example

Example:

```
Employee_obj'print()
```

Output:

```
{
  "Address": {
    "Street": "32 Avenue"
  },
  "Name": "Jane Doe"
}
```

Methods (Finds)

Method	Description
Find(*regex$***)**	Returns a JSON list of keys that match the given *regex$* (where *regex$* is a string of a regular expression). Returns an empty list if no keys are found.
Find(*regex$, key$***)**	Returns a JSON list of keys that match the given *regex$* within the given sub-key. Returns an empty list if no keys are found.
Find_any(*regex$***)**	Returns 1 if any matching keys are found. Returns 0 if none are found.
Find_any(*regex$, key$***)**	Returns 1 if any matching keys in the given sub-key are found. Returns 0 if none are found.

Find Example

Example:

```
Found = Employee_obj'find(".*Street")

Found'print()
```

Output:

```
[

  "Address.Street"

]
```

JSON Data:

```
{

  "Address": {

    "Street": "32 Avenue"

  },

  "Name": "Jane Doe"

}
```

Methods (Other)

Method	Description
Clear()	Removes all values from the JSON object.
Is_list_index(*key$***)**	Returns 1 if the given key is an index of a list; otherwise, returns 0.
Is_num(*key$***)**	Returns 1 if the given value is numeric; otherwise, returns 0.
Is_str(*key$***)**	Returns 1 if the given value is a string; otherwise, returns 0.
Remove(*key$***)**	Removes key from the JSON object.
Size()	Returns the total number of elements in the JSON object.

Methods (Loading and Saving)

Method	Description
Load_array(*array${all}***)**	Load JSON from an associative array.
Load_array(*array${all}, num_flag***)**	Load JSON from an associative array as only strings if *num_flag*=0.
Load_json_file(*path$***)**	Loads the JSON object from the data in the given JSON file. File content must start with "{" or "[".
Load_string(*jsonString$***)**	Load JSON from a string.

Methods (Loading and Saving)

Method	Description
Load_xml(*xml_object***)**	Loads the data of an XML object into the JSON object. Converts strings to numeric if possible. Does not support duplicate XML tags.
Load_xml(*xml_object, num_flag***)**	Loads the data of an XML object into the JSON object as only strings if *num_flag*=0. Does not support duplicate XML tags.
Save_json_file(*path$***)**	Saves the JSON object data to a file.

Note: Load_xml() does not retain XML attribute tags

Loading Array Example

Create an associative array:

```
dim my_array$

my_array$["Person.FirstName"] = "Jane"
my_array$["Person.LastName"] = "Smith"
my_array$["StreetNum"] = 456
my_array$["StreetName"] = "Ontario Street"

print dim(list edit my_array${ALL})
```

Output:

```
{
  "Person":{
    "FirstName": "Jane",
    "LastName": "Smith"
  },
  "StreetName": "Ontario Street",
  "StreetNum": 456
}
```

Loading Array Example

JSON Data:

```
My_json = NEW("*obj/json")

My_json'load_array(my_array${all})
```

JSON Data:

```
{
  "Person": {
    "FirstName": "Jane",
    "LastName": "Smith"
  },
  "StreetName": "Ontario Street",
  "StreetNum": 456
}
```

Loading XML Example

Example:

```
New_json = NEW("*obj/json")

New_json'load_xml(xml_obj)
```

XML Data:

```
<Product>

  <Name>Laptop</Name>

  <Price currency="USD">400</Price>

  <SN>294819383847</SN>

</Product>
```

Loading XML Example

JSON Data:

```json
{
  "Product": {
    "Name": "Laptop",
    "Price": 400,
    "SN": 294819383847
  }
}
```

XML Data:

```xml
<Product>
  <Name>Laptop</Name>
  <Price currency="USD">400</Price>
  <SN>294819383847</SN>
</Product>
```

Lists

Example:

```
Json list=NEW("*obj/json")

Json_list'append("1", "First")

Json_list'append_to("2", "sub_obj", "Second")

Json_list'append_to("3", "#123", "NotAList")

Json_list'print()
```

Output:

```
[

    "First",

    { "sub_obj":   "Second" },

    { "123": "NotAList" }

]
```

Null, Booleans, and Empty Lists

Example:

```
Special_values = NEW("*obj/json")

Special_values'append("Null", "null")

Special_values'append("Boolean", "False")

Special_values'append("EmptyList", "[]")

Special_values'print()
```

Output:

```
{

    "Null": null,

    "Boolean": False,

    "EmptyList": []

}
```

Methods Coming in PxPlus 2025 Upd 1

Method	Description
Exists(*key$)*	Returns 1 if key exists, 0 if key does not exist.
Get_json(*key$, full_key_flag,* ***no_validation*****)**	Returns a JSON object of the given key. Keys will use the full key name when *full_key_flag*=1. *no_validation*=1 will ignore key validation. Err 11 if the object size is 0.
Get_keys()	Returns a JSON object of all keys in the object.
Get_num(*key$,* ***no_validation*****)**	Returns a numeric from the key. *no_validation*=1 will ignore key validation. Returns 0 if key not found.
Get_str$(***key$,* ***no_validation***)**	Returns a string from the key. *no_validation*=1 will ignore key validation. Returns 0 if key not found.

Methods Coming in PxPlus 2025 Upd 1

Method	Description
Print_keys()	Prints all keys within the object.
Remove(*key$, **no_validation***)**	Removes key from the JSON object. *no_validation*=1 will ignore key validation. Err 11 if the object size is 0.
Set_json(*key$, jsonObj, **no_validation***)**	Sets the key to key: JSON. *no_validation*=1 will ignore key validation. Returns an Err 91 if operating on itself. Returns Err 11 if the given object size is 0.
Load_xml(*xml_object, num_flag, **dupe_flag***)**	Loads the xml object data into the JSON object as only strings if *num_flag*=0. If *dupe_flag*=1 then duplicates will also be handled by condensing duplicates into a single list (Note: Loading data with very large amounts of duplicates may become very slow.) Err 11 if the object size is 0.

Methods Coming in PxPlus 2025 Upd 1

Method	Description
Find_val(*regex$, val***)** **Find_val(***regex$, val$***)**	Returns a JSON list of keys matching the given *regex$*, where *regex$* is a string of a regular expression, that equal the given value. Err 11 if no keys are found.
Find_val(*regex$, val, key$***)** **Find_val(***regex$, val$, key$***)**	Returns a JSON list of keys matching the given *regex$*, where *regex$* is a string of a regular expression, that equal the given value, within the given sub-key.
Find_any_val(*regex$, val***)** **Find_any_val(***regex$, val$***)**	Returns 1 if any matching keys that equal the given value are found, 0 if none are found.
Find_any_val(*regex$, val, key$***)** **Find_any_val(***regex$, val$, key$***)**	Returns 1 if any matching keys in the given sub-key are found that equal the given value, 0 if none are found.

Print Keys Example

Example:

```
! Prints the return of get_keys()
New_json'print_keys()
```

Output:

```
[
  "Product.Name",
  "Product.Price",
  "Product.SN"
]
```

JSON Data:

```
{
  "Product": {
    "Name": "Laptop",
    "Price": 400,
    "SN": 294819383847
  }
}
```

Loading Duplicate XML Example

Example:

```
Dupe_json = NEW("*obj/json")

Dupe_json'load_xml(xml_obj, 1, 1)

Dupe_json'print()
```

XML Data to load:

```
<Order>

  <Product>Apple</Product>

</Order>

<Order>

  <Product>Orange</Product>

</Order>
```

Loading Duplicate XML Example

JSON Data:

```
{
  "Order": [
    { "Product": "Apple" },   (Order.1.Product)
    { "Product": "Orange" }   (Order.2.Product)
  ]
}
```

XML Data to load:

```
<Order>
  <Product>Apple</Product>
</Order>
<Order>
  <Product>Orange</Product>
</Order>
```

New Method Examples

Example:

```
Regex$ = "Products\.Product.*"
Found = New_json'find_val(Regex$, 296819383847)
Found'print()
```

Output:

```
[
  "Products.Product2"
]
```

JSON Data:

```
{
  "Products": {
    "Product1": 194819383847,
    "Product2": 296819383847,
    "Product3": 394819333847
  }
}
```

Performance

27,875 data entries, no duplicates

XML	Speed (Seconds)	Memory (MB)
Load File	21.03	17.1
Save File	6.15	6.4
Load JSON	103.41	17.1
Get String	6.23	6.4

JSON	Speed (Seconds)	Memory (MB)
Load File	0.03	3.6
Save File	0.03	0.0
Load XML	13.33	6.2
Get String	0.03	1.0

Performance

27,875 data entries, no duplicates

JSON	Speed	Memory
Load File	701x faster	5x less
Save File	205x faster	∞ less
Load XML	8x faster	3x less
Get String	208x faster	6x less

Performance (Duplicates)

JsonObj'Load_xml()	Entries	Time (seconds)	Processing Rate
No duplicates:	27,875	13.33	2091 / second
All duplicates:	5,000	75	67 / second

Summary

- Great performance
- Easy to use
- Intuitive
- Variety of useful functions
- Saves development and processing time
- Load and save your data easily
- Methods for performance-minded developers

Resources

Where can I find more information about the PxPlus JSON Object?

How To Tutorials – Several How-To tutorials on working with the PxPlus JSON Object

PxPlus Video Library – 2-part video series that covers the highlights of the PxPlus JSON Object and the creation and use of the JSON Object

PxPlus Help - JSON Object – JSON Object documentation page.