SRP JsonX

SRP Json Express (JsonX) is a new and improved API for building and parsing json. Gone are the days of handles and memory management. SRP JsonX was built with workflow in mind.

> Available in SRP Utilities 2.2 or later.

Stateful

To avoid dealing with handles, SRP JsonX relies on state. Under the hood, SRP JsonX maintains a stack of JsonX documents. Whenever you start a JsonX document, that document is placed at the top of the stack and becomes the current document. Every document has a current path, which defaults to the root of the document. These two states are important to navigating, updating, and building json documents with SRP JsonX. When you are done with a document, you call SRP_JsonX_End, which removes the current document from memory and makes the previous document on the stack the current one.

Creating Json

There are two routines for creating new json documents: SRP_JsonX_Begin and SRP_JsonX_BeginString. SRP_JsonX_Begin creates a new json data structure that can be built, navigated, and modified at will. SRP_JsonX_BeginString creates a new json string that can only be built top down.

Example 1

```
$insert SRPJSONX

// Create a new document with a json object as the root. All documents are named for debugging purposes only
SRP_JsonX_Begin('MyDocument', '{')

        // Since we are in an object, the first parameter should be a member name followed by a value
        // In this case, the value is "[", which starts a new array and makes that array the current element
    SRP_JsonX('employees', '[')

                // Since we are in an array, the first parameter should be a value. In this case, we are
starting a new object
                SRP_JsonX('{')
                        SRP_JsonX('firstname', 'John')        ; // Since we're in an object, the first
parameter is a member name and the second is its value
                        SRP_JsonX('lastname', 'Doe')
                        SRP_JsonX('age', 21)
                SRP_JsonX('}')

                // Note on the previous line that passing "}" closes an object and moves the current element
back to the parent, so when we call this next
                // line, we are back in the array. Once again, we'll add an employee object
                SRP_JsonX('{')
                    SRP_JsonX('firstname', 'Anna')
                    SRP_JsonX('lastname', 'Smith')
                    SRP_JsonX('age', 32)
                SRP_JsonX('}')

                // We can also pass json strings as a value, which get fully parsed and added to the current
element, which is the employees array
                SRP_JsonX('{"firstname":"Peter", "lastname":"Jones", "age":43}')

        // This line closes the array and sets the current element back to the root object
    SRP_JsonX(']')

        // When you pass numbers as values, JsonX assumes you want that value to be unquoted in the final json
         SRP_JsonX('count', 4)

        // To pass a boolean value, we need to add a hint parameter. Hints always come after values
         SRP_JsonX('active', 1, 'Bool')

        // To set something to null, omit the value. If you cannot omit the value, you can set the hint to
'Null'
         SRP_JsonX('alwaysnull')

        // If you want a number to be quoted in the json, use the 'String' hint
         SRP_JsonX('alwaysstring', 4.321, 'String')

// We're all done, so let's turn the document into pretty formatted json and end it at the same time
Json = SRP_JsonX_End('Pretty')
```

In this example, we start a new document called MyDocument. The document name is for debugging purposes only, so use whatever you want. The second parameter defines the new document's root. It must be "{" or "[". Note that this new document is now the active one and it's current element is the root of the document.

SRP_JsonX is a special routine whose parameters are interpreted according to the current state. The comments above help explain how SRP_JsonX makes its decisions.

To get the final json output, we call SRP_JsonX_End. This routine does two things. It optionally returns the current document as a json string, then it removes the current document from memory and makes the previous document on the stack the current one.

Example 2

```
SRP_JsonX_BeginString('MyDocument', '{', JsonxPretty$)
    SRP_JsonX('employees', '[')
        SRP_JsonX('{')
            SRP_JsonX('firstname', 'John')
            SRP_JsonX('lastname', 'Doe')
            SRP_JsonX('age', 21)
        SRP_JsonX('}')
        SRP_JsonX('{')
            SRP_JsonX('firstname', 'Anna')
            SRP_JsonX('lastname', 'Smith')
            SRP_JsonX('age', 32)
        SRP_JsonX('}')
        SRP_JsonX('{"firstname":"Peter", "lastname":"Jones", "age":43}')
    SRP_JsonX(']')
    SRP_JsonX('count', 4)
    SRP_JsonX('active', 1, 'Bool')
    SRP_JsonX('alwaysnull')
    SRP_JsonX('alwaysstring', 4.321, 'String')
Json = SRP_JsonX_End()
```

Example 2 makes the same json as Example 1, but this time we started with SRP_JsonX_BeginString. Note how the calls to SRP_JsonX don't behave any differently. The only difference is that behind the scenes, SRP JsonX is producing pretty json text directly. If you know you are making a json string and you plan to build it from the top down, then SRP_JsonX_BeginString will be faster.

Parsing Json

SRP JsonX is the fastest parser available to OI developers, and now it's easier to navigate, modify, and extract json documents thanks to the elimination of cumbersome handles. Let's parse the following json:

```
{
    "employees": [
        {
            "firstname": "John",
            "lastname": "Doe",
            "age": 21
        },
        {
            "firstname": "Anna",
            "lastname": "Smith",
            "age": 32
        },
        {
            "firstname": "Peter",
            "lastname": "Jones",
            "age": 43
        },
    ],
        "nums": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    "alwaysbool": true,
    "alwaysnull": null,
    "alwaysstring": "4.321"
}
```

Now lets parse it and learn how to navigate it, extract values, and even modify it.

Example 3

```
$insert SRPJSONX

// Create a new document by parsing the json text
SRP_JsonX_Parse('MyParsedDoc', Json)

        // Get the first employee's last name using SRP JsonX path syntax
        LastName = SRP_JsonX_Get('employees[1].lastname')

        // If our path points to an object or an array, we'll get the entire object or array as json text
        Employee = SRP_JsonX_Get('employees[1]')

        // SRP_JsonX_Get doesn't change the current element. If we want to change it, we use the Go routine.
        // Let's make the second employee the current element.
        SRP_JsonX_Go('employees[2]')

        // Now, if I want to get the last name, I must use a path relative to the current element
        LastName = SRP_JsonX_Get('lastname')

        // If I try to use a full path, I'll get null because all paths are relative to the current element
        // In this case, 'employees' doesn't exist inside 'employees[2]'
        LastName = SRP_JsonX_Get('employees[3].lastname')

        // We can go back up the hierarchy one level at a time. This call makes 'employees' the current element
        SRP_JsonX_GoBack()

        // Now I can get the third employee's last name
        LastName = SRP_JsonX_Get('[3].lastname')

        // We can make the root the current element at any time
        SRP_JsonX_GoRoot()

        // When a value is null or not found, '' is returned by default. If you want a different value,
        // then set the DefaultIfNull parameter to your desired default
        Value = SRP_JsonX_Get('alwaysnull', 'This is null!!!')

        // We can also query the document for information (remember, we are currently at the root)
        NumEmployees = SRP_JsonX_Count('employees')
        MemberExists = SRP_JsonX_Has('employees[2].age')
        MemberNames  = SRP_JsonX_Members('employees[3]')
        Type         = SRP_JsonX_Type('alwaysbool')
        Numbers      = SRP_JsonX_Values('nums', ',')

        // We can also modify the document
        SRP_JsonX_Clear('nums')
        SRP_JsonX_Delete('employees[2]')
        SRP_JsonX_Sort()

// When we're done, we end it. Note that we can call this as a subroutine when we don't want json returned
SRP_JsonX_End()
```

See Paths for more information.

Debugging

Given that SRP JsonX relies so heavily on state, it's important to give you the tools you need to troubleshoot issues. If you want to view the current state of SRP JsonX, call SRP_JsonX_State or SRP_JsonX_Trace. Both routines give you the stack and the current elements for each document on the stack. The difference between the two is that SRP_JsonX_Trace displays a message box whereas SRP_JsonX_State returns the stack as an @FM delimited array suitable for viewing in the debugger.

When an SRP JsonX method fails, you can call SRP_JsonX_Error immediately afterwards to get a detailed error message.

Reference

Here are links to all the SRP JsonX functions.

Routine	Description
SRP_JsonX	Adds elements to the current document based on the current path.
SRP_JsonX_Begin	Creates a new document.
SRP_JsonX_BeginString	Creates a new document that must be built in order.
SRP_JsonX_Clear	Deletes all elements in an object or array.
SRP_JsonX_Count	Gets the number of elements in an object or array.
SRP_JsonX_Delete	Deletes an element.
SRP_JsonX_End	Ends the current document, optionally returning json.
SRP_JsonX_Error	Gets the last known error.
SRP_JsonX_Get	Gets an element's value.
SRP_JsonX_Go	Makes an element the new current element.
SRP_JsonX_GoBack	Makes the parent of the current element the new current element.
SRP_JsonX_GoRoot	Makes the document's root the current element.
SRP_JsonX_Has	Determines if an element exists.
SRP_JsonX_Members	Gets all an object's member names.
SRP_JsonX_Parse	Parses json into a new document.
SRP_JsonX_Set	Sets a value or json.
SRP_JsonX_Sort	Sorts an object's members by member name.
SRP_JsonX_State	Gets the state of the stack and all its documents.
SRP_JsonX_Trace	Shows the state of the stack and all its documents.
SRP_JsonX_Type	Gets an element's type.
SRP_JsonX_Values	Gets all an object's or array's values.