SRP_List

SRP Lists are an alternative replacement to BASIC+ lists and can greatly enhance performance for large processes.

Method	Description	Added
Add	Adds an element to the end of an SRP List.	
Clear	Removes all values from an SRP List.	2.1.10
Count	Gets the number of elements in the list.	
Create	Creates an SRP List.	
CreateFromFastArray	Creates an SRP List initialized to a list within the given SRP Fast Array.	
GetAt	Gets the element from an SRP List at the given index.	
GetVariable	Converts an SRP List back into a BASIC+ variable.	
InsertAt	Inserts an element into an SRP List at the given index position.	
Locate	Locates a value in an SRP List.	
Match	Finds the index of the first element from the starting index that matches the given string.	
Reduce	Creates a new list containing only those elements that match the given string.	
Release	Releases the handle to an SRP List.	
RemoveAt	Removes an element from an SRP List at the given index position.	
SetAt	Sets an element into an SRP List at the given index position.	

List Defined

In this documentation, the term List refers to a single-dimensional dynamic array. For multi-dimensional array support, see SRP FastArray.

BASIC+ Lists

Easily one of the best features of the BASIC+ language is its array manipulation. It's easy, intuitive, and flexible: the trifecta of efficient programming. No need to predetermine the size of your array. No need to worry about going out of bounds. Just set a position and you're all set.

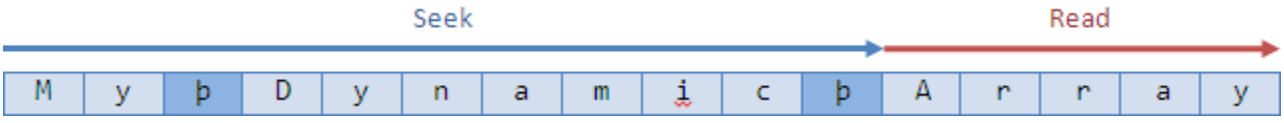
```
List<2> = MyValue
```

However, efficient programming comes at a cost. In this case, that cost is performance. To understand why, we have to remember that a dynamic array in BASIC+ is just a single string.

```
List = "My":@FM:"Dynamic":@FM:"Array"
```

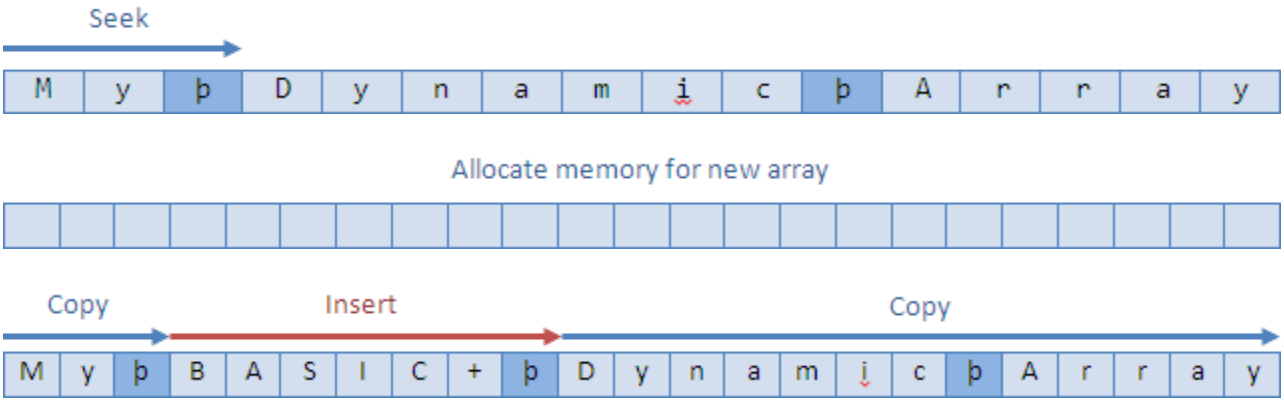
Whenever you extract a value from the array, BASIC+ scans the entire string counting delimiters until it finds what you are looking for:

```
Value = List<3>
```



Inserts, Replaces, and Deletes involve three steps. First, it scans the array to find the point of insertion, deletion, or replacing. Next, a new block of memory is allocated. Finally, the new array is created by copying elements from the old array as well as copying in any new values.

```
List = Insert(List, 2, 0, 0, "BASIC+")
```



If you have a For loop that is inserting thousands of elements into a dynamic array, then BASIC+ is reallocating memory and rewriting your array thousands of times also. This kind of operation is expensive performance wise.

SRP Lists

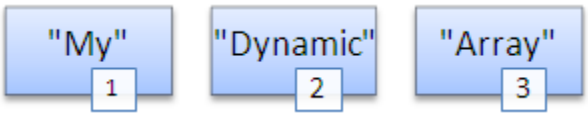
SRP Lists seek to offer an alternative to BASIC+ lists that behave exactly the same way yet offer much more performance. This performance is achieved in two ways. First, it is written in optimized C++. Second, it never uses a single string to hold the entire list. You start by creating an SRP List. You can create an empty SRP List like this:

```
ListHandle = SRP_List("Create")
```

Or you can create an SRP List initialized to a BASIC+ list, like this:

```
ListHandle = SRP_List("Create", "My":@FM:"Dynamic":@FM:"Array")
```

Here's what your list looks like in memory.

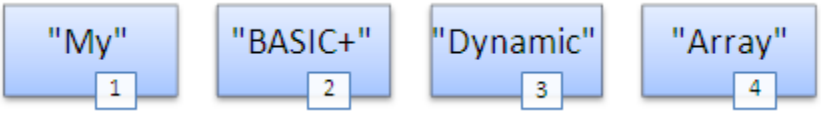


Notice how no delimiters are tracked and each element is stored in its own block of memory, although their order is maintained through a numeric index. Getting an element is always fast because there is no need to scan a string.

```
Value = SRP_List("GetAt", ListHandle, 3)
```

SRP_List simply grabs element three and returns it. Inserts, replaces, and deletes are all equally fast for similar reasons. SRP_List never needs to recopy its elements. An insert simply places the new value as a new string right where it needs to go.

```
SRP_List("InsertAt", ListHandle, 2, "BASIC+")
```



You can imagine how fast this is for lists with thousands of elements being inserted, deleted, and replaced. Of course, you will eventually need your SRP List to become a BASIC+ list again, and this is how you do it:

```
List = SRP_List("GetVariable", ListHandle, @FM)
```

This function does allocate memory for the entire list and copies each element—with delimiters—into it. Obviously, if you call this method thousands of times, then you're no better off than when you were using a BASIC+ list. The idea is to manipulate your list using SRP List and then write it out when you are done.

There is always one more step when using SRP Lists. You have to release the handle.

```
SRP_List("Release", ListHandle)
```

Sure, it's an extra step, but the performance gains are usually worth it. If you forget to do this, SRP Utilities will clean up all unreleased handles when OpenInsight closes, but if you plan on your application running for hours at a time, you still want to avoid memory getting used up and never freed.

Locates and SRP Lists

In BASIC+, we use the Locate statement to search for elements in our list.

```
Locate Target in List using @FM setting Pos then
end
```

Just like all other list manipulation, BASIC+ scans the string from front to back until it finds the target. As usual, if you are doing this hundreds or thousands of times, you can experience some serious slowdowns.

SRP_List, however, has a special trick up its sleeve. The entire list is indexed! So, whenever you do a locate against it, it's instant. It doesn't matter if you have ten elements or a million. Moreover, the syntax is easy.

```
Pos = SRP_List("Locate", ListHandle, Target)
```

Locates are case sensitive. If you need a more flexible albeit slower means of searching, check out the Match service. And if you want to filter your list into a smaller one based on a search string, see the Reduce sevice.

Lists Sizes

One thing we often do is count the number of elements in an list. In BASIC+ we typically do this with the following code:

```
ListCount = Count(List, @FM) + (List NE "")
```

It should be obvious that the Count method scans the entire string counting delimiters, and then we have to add 1 to the final count if the list is not empty. To count the number of elements in an SRP List, just do this:

```
ListCount = SRP_List("Count", Handle)
```

Counting elements in an SRP List is instantaneous because the list count is always cached.

Delimiter Agnostic

SRP Lists do not care if your original list is field-mark delimited or comma delimited, because internally the list never uses delimiters. Instead, you specify a delimiter when you initialize the SRP List and you specify the delimiter when you get the final list. You don't even have to use the same delimiter each time.

```
// Init with comma as the delimiter
ListHandle = SRP_List("Create", "FirstDay,Sunday,Tuesday,Wednesday,Thoosday,Friday", ",")

// Do some inserts, replaces, and additions
SRP_List("RemoveAt", ListHandle, 1)
SRP_List("InsertAt", ListHandle, 2, "Monday")
SRP_List("SetAt", ListHandle, 5, "Thursday")
SRP_List("Add", ListHandle, "Saturday")

// Return the list, but make it value-mark delimited
List = SRP_List("GetVariable", ListHandle, @VM)

// Clean up
SRP_Release("Release", ListHandle)
```

When to Use SRP_List

As with everything, use the right tool for the right job. There are two factors to consider: the size of the list and the number of operations you intend to perform upon it. The smaller the list, the less amount of time BASIC+ spends scanning, allocating, and copying. Also, even if you have a list that takes up several megabytes in memory, if you are only doing two extracts, SRP List won't make that much of a difference.

Perhaps the best rule of thumb is this: if you have a process that takes a long time to run, look to see how much list manipulation it is doing. If there appears to be a great deal of it, then try out SRP List and see if you get an improvement. Benchmark the differences to see if the increase in code complexity (SRP List methods are, after all, not as pretty as angle bracket operators) seems worth the performance gain.

When Not to Use SRP_List

There is one situation in which SRP List does not seem to offer any significant performance increase: appending values. If your routine is only appending values at the end of your single-dimensional list, then BASIC+ is plenty fast for you. Essentially, BASIC+ dynamic arrays begin to suffer in performance when you perform random access, and merely appending values in a single-dimensioned array is not random.

Sample Benchmark Code

Here is a routine you can copy into OpenInsight and run (after installing SRP Utilities 1.5.7 or greater). It inserts and extracts elements into each type of list at random and times it. It also benchmarks repeated locates. Notice that 1000 iterations or less offer pretty negligible gains, but 10,000 or more iterations really start to show the speed difference. Your mileage will vary depending upon your PC specifications.

```
Compile function Test_List(VOID)

$Insert SRPLIST
// OI routines
Declare function GetTickCount
Declare subroutine Msg

Iterations = 10000                              ; // How many operations to perform
MaxPos = 10000                                  ; // The maximum field, value, or subvalue position
InsertText = "SRP Computer Solutions, Inc."     ; // The text to insert

InitRnd Time():Date()

// Insert the text at random field positions
StartTime = GetTickCount()
TestOI = ""
For i = 1 to Iterations
   TestOI = Insert(TestOI, Rnd(MaxPos), 0, 0, InsertText)
Next i
InsertTimeOI = GetTickCount() - StartTime

// Insert the text at random field positions using SRP List
StartTime = GetTickCount()
Handle = SRP_List("Create")
For i = 1 to Iterations
   SRP_List("InsertAt", Handle, Rnd(MaxPos), InsertText)
Next i
TestSRP = SRP_List("GetVariable", Handle)
InsertTimeSRP = GetTickCount() - StartTime

// Extract values at random field positions
StartTime = GetTickCount()
For i = 1 to Iterations
   A = Extract(TestOI, Rnd(MaxPos), 0, 0)
Next i
ExtractTimeOI = GetTickCount() - StartTime

// Extract values at random field positions user SRP List
StartTime = GetTickCount()
For i = 1 to Iterations
   A = SRP_List("GetAt", Handle, Rnd(MaxPos))
Next i
ExtractTimeSRP = GetTickCount() - StartTime

// Insert a unique string into the center of the lists (to simulate average locate times)
Target = "LOOK AT ME!"
InsertPos = Int(MaxPos / 2)
```

```
TestOI = Insert(TestOI, InsertPos, 0, 0, Target)
SRP_List("InsertAt", Handle, InsertPos, Target)

// Locate the target over and over
StartTime = GetTickCount()
For i = 1 to Iterations
   Locate Target in TestOI using @FM setting Pos then null
Next i
LocateTimeOI = GetTickCount() - StartTime

// Locate the target over and over using SRP List
StartTime = GetTickCount()
For i = 1 to Iterations
   Pos = SRP_List("Locate", Handle, Target)
Next i
LocateTimeSRP = GetTickCount() - StartTime

// Clean up the memory
SRP_List("Release", Handle)

Msg(@Window, "OI Insert Time: ":InsertTimeOI:"ms|SRP Insert Time: ":InsertTimeSRP:"ms||OI Extract Time: ":
ExtractTimeOI:"ms|SRP Extract Time: ":ExtractTimeSRP:"ms||OI Locate Time: ":LocateTimeOI:"ms|SRP Locate Time: ":
LocateTimeSRP:"ms")

Return 1
```

Note that the first benchmark will always be a little slower than subsequent benchmarks because upon the first run, the C++ code has to be loaded into memory.

Initialize from SRP Fast Arrays

SRP Lists can be initialized from SRP Fast Arrays using a special function. You simply provide the SRP Fast Array handle and the list from within the SRP Fast Array you want to use. If you want to initialize it from all the fields in the array, you pass zeroes in both the field and value parameters.

```
ListHandle = SRP_List("CreateFromFastArray", ArrayHandle, 0, 0)
```

If you want to initialize the SRP List to a list of values, then only set the value parameter to zero.

```
ListHandle = SRP_List("CreateFromFastArray", ArrayHandle, 7, 0)
```

And, of course, if you want to initialize the list to a list of subvalues, then set both the field and the value parameters.

```
ListHandle = SRP_List("CreateFromFastArray", ArrayHandle, 2, 3)
```

Points to Remember

Don't forget to release your SRP List handles. Always.

Note that one major difference between the BASIC+ Insert, Delete, and Replace routines and the SRP List equivalent routines is that the BASIC+ routines always creates a new list and returns it. SRP List does not do this since creating copies is the performance bottle neck SRP List is working to avoid.

Oh yeah, one more thing: **don't forget to release your SRP List handles.**