SmartArrays® and APL: Frequently Asked Questions

Because SmartArrays, like APL, is based on arrays, and because it was developed by veterans of the APL industry, a number of questions have been asked about how SmartArrays compares to APL, J, and other array languages. The short answer is that SmartArrays was inspired by APL and our desire to share the power of array-oriented programming with a larger audience. In order to serve the needs of that audience as well as possible, we have given greater weight to making SmartArrays work smoothly within object-oriented languages like C++ and Java than we have to matching all aspects of APL’s behavior.

Is programming with SmartArrays similar to programming in APL?

Writing code to use SmartArrays will feel very familiar to someone who has prior experience with APL. Even though the syntax used is different, the way one thinks about a computation as a series of array operations is the same. Consider the following APL statement that selects all the positive numbers from an array named A:

\[ b ← (a > 0) / a \]

With SmartArrays, an array is an object, and it has built-in array methods. The equivalent expression in Java or C++ is:

\[ \text{SmArray} \ b = \text{a.gt}(0).\text{compress}(\text{a}); \]

The expression reads from left to right, because that's the way the syntax of these languages works. The first part \[ a.\text{gt}(0) \] takes an SmArray object named \[ a \] and invokes its method \[ \text{gt}() \], which produces a new array object as the result (a Boolean vector). Then that new object’s \[ \text{compress}() \] method is invoked, which in turn produces a new array that is assigned to the variable \[ b \].

There are also ways in which working with SmartArrays is very different from working with APL. In APL, everything must be done with an array because arrays are the only form of data the language provides. This is both good and bad. Although arrays have great power to simplify and shorten the logic of data manipulations and calculations, APL forces the programmer to do everything with arrays, even in places where array operations seem more of a hindrance. The result can be code that appears bizarre or silly to programmers who are familiar with standard compiled languages. Such code can also be very inefficient.

Consider for example the following APL statement for conditionally executing some code if a scalar number is negative:

\[ →(x > 0) / \text{POSITIVE} \]
\[ \cdots (\text{code to run if } x \text{ is negative}) \]

Evaluating this statement requires an APL interpreter to:
1. allocate and initialize a scalar array and set its value to zero
2. apply the > primitive between array x and that array, creating a new array.
3. allocate and initialize a scalar array containing the numeric line number of the label POSITIVE.
4. apply the / (replicate) primitive to this scalar and the result of >, producing a new vector.
5. examine this array to determine if it is empty or, if not, which line number to branch to.

Even a highly refined APL interpreter requires hundreds or thousands of machine instructions to do this, while the equivalent C++ code:

```
if ( x <= 0 ) { ... }
```

compiles into only two or three machine instructions. In such a case, APL is much slower and clumsier than C++ with SmartArrays.

The goal of SmartArrays is to provide the benefits of array-oriented programming to standard languages, while preserving the things that those languages already do well, including their efficiency and all the benefits of object-oriented programming. A C++ program written using SmartArrays can use C++ in a natural way, and then apply arrays in places where they are needed. This will tend to produce programs that are readily understood by other programmers. In practice, a C++ or Java programmer is likely to develop a suite of building-block classes that use SmartArrays and encapsulate the array logic in ways that make it easy for others to understand and maintain the code.

To summarize, the overall approach to programming in SmartArrays is different from APL, but the array-oriented logic -- creating arrays and applying primitive functions to them -- is the same.

**Does SmartArrays have the same functions as APL?**

SmartArrays has nearly all the APL and APL2 primitives, with the same or nearly the same definitions. We have felt free to leave some out and add new ones. We have also changed the behavior of some primitives in a few areas where we felt the APL behavior would not make sense to a C++ or Java programmer. A few examples:

- All indexing operations are origin-0 only; there is no origin 1.
- Relational functions like gt() or eq() do not use a comparison tolerance by default, although they accept a comparison tolerance as an optional parameter.
- Because SmartArrays primitives are called with a parameter list, we are able to add additional arguments in ways that give the programmer greater control over what the primitive does. APL’s syntax limitation of no more than 2 arguments does not apply.
- SmartArrays has an atomic string datatype (a sequence of Unicode characters that is treated as a simple scalar), a feature not found in any existing APL interpreter.
- Reduction and scan by default work from left-to-right (like C++ and Java) instead of right-to-left. This rarely makes a difference unless one of the more obscure cases of reduction or scan is being used.
- Edge conditions with prototypes and fill items in empty arrays that plagued second-generation APL designs have been dealt more simply in SmartArrays.

But the bulk of the primitives work the same way. Common ideas like reduction, or inner product, or grade, or “dyadic iota” are the same in SmartArrays as they are in APL.

**Will knowing APL help me use SmartArrays?**

Yes! Any programmer who has worked with APL or one of its derivative languages has a major head start in learning SmartArrays. An intuitive understanding of the behavior of APL primitives will be directly transferable onto SmartArrays, while a C++ programmer with no array oriented experience will need time to become familiar with these ideas.
Should I replace APL with SmartArrays?

If you are happy with APL or J, and it is meeting your needs, there is no reason to switch to SmartArrays. But SmartArrays works very well in situations where APL is problematic, such as when corporate policy dictates that a “mainstream” language must be used, or when code must be deployed in contexts that are awkward for APL, such as server-side applications with web front ends. In such situations, SmartArrays can be a valuable alternative or complement to APL.

If an APL application needs to be rewritten, SmartArrays can allow you to keep the APL data model, and the efficient array algorithms that work on it, while putting those ideas to work in new forms. In fact, some developers like to prototype algorithms first in the interactive APL environment before writing them in a compiled language with SmartArrays.

Will SmartArrays let me automatically translate APL to Java or C++?

SmartArrays Inc. has developed a translator tool that converts APL functions into languages with C++ or Java syntax. This tool will be released as a commercial product in early 2004.

The SmartArrays Translator works on a single APL function at a time and produces the computationally equivalent function in the target language. It is useful for converting legacy APL applications to a SmartArrays base. Contact SmartArrays for more detailed information on the Translator.

Can I use SmartArrays with APL?

It’s possible to mix APL and SmartArrays in a number of interesting ways. For example, SmartArrays provides the methods toWrapl() and fromWrapl(), that work with the serialized form of APL+Win (APL2000) arrays. This means that any APL+Win array, no matter how large or deeply nested, can be easily and efficiently converted into a SmartArrays array and back. For example, one might use this facility to build an interface that integrates APL+Win with ADO.NET data sources. We have also built C++ and Java classes that know how to read APL+Win component files. The classes are freely available to licensed SmartArrays users.

We have also seen a demonstration of calling SmartArrays from Dyalog APL for .NET. Because SmartArrays for .NET implements a standard Common Language Runtime API, Dyalog APL is able to call the SmartArrays classes like any other .NET class.

If you have technical questions on any of these topics, please feel free to contact SmartArrays.