# Type Checking With "bbjcpl"

***By David Wallwork***

The traditional BBx® language has only three variable types; string (**A$**), number (**A**) and integer (**A%**). Since the introduction of embedded Java, BBj® gave the BBx developer the ability to work with Java objects as well as these three native BBx types. In the 6.0 release of BBj, the available variable types are even broader with the introduction of custom objects; objects defined completely in BBj program files.

While using Java objects and BBj custom objects allows BBj developers to write sophisticated object-oriented applications, these same object-oriented applications have the potential for hard-to-find type check errors. BBj 6.0 provides an easy-to-use tool for discovering these errors – new type checking options in the **bbjcpl** executable that compiles ASCII program files into binary tokenized programs. In this article, we will examine a small program that demonstrates some of the errors that may occur when using object-oriented code, and then show how to discover these errors using **bbjcpl** with these new options.

Consider the program in **Figure 1**. This program contains problems that will generate runtime errors when executed.

```
0010  rem  demo1.bbj

0200  A! = "abc"
0210  B! = 5
0220  print A! + B! ; rem runtime error

0300  C! = "abc"
0310  D! = 5
0320  print C! + D! ;  rem runtime error

0410  x! = bbjapi()
0420  x!.getExistingNamespace( 44 ) ;  rem runtime error
```

**Figure 1.** Code sample that produces runtime errors

**Problem 1**

At lines 200-220, this program assigns a string value to **A!** and a numeric value to **B!**, and then attempts to add **A!** to **B!**. Line 220 will generate a runtime error because BBj does not support adding a numeric value to a string value.

**Problem 2**

The same problem occurs again at lines 300-320. The problem appears twice for demonstration purposes. Below we will resolve the problem at line 200-220 differently from the resolution offered to the problem at line 300-320.

**Problem 3**

At line 420 the program attempts to pass a numeric value to the method **getExistingNamespace()**. Because that method accepts a string parameter rather than a numeric parameter, the program will generate a runtime error when line 420 is executed.

The entire program is syntactically correct. This means it will load and list correctly and none of the lines will be marked as syntax errors. And yet, we can see from looking at the program that these statements will cause runtime errors. One might ask, "Why is the compiler not able to recognize these problems and mark them as errors during compilation?"

***David Wallwork***
*Senior Architect*

The reason that a human reading the program can recognize the errors is that the reader has additional information about the variables. This additional information is *type information*. For example, the user understands from looking at line 200-210 that variable **A!** is of type "string" and the variable **B!** is of type "number." And the user can recognize that at line 220 the program is attempting to add a number to a string. Similarly, at line 420 the reader knows (or can discover) that the method **getExistingNamespace()** requires a string but that what is being passed is a number.

*continued...*

If there were a way to provide type information to the compiler, then the compiler would also be able to recognize these problems. In BBj 6.0, **declare** statements provide this type information. The program in **Figure 2** uses **declare** statements to provide this information to the compiler.

```
0010  rem demo2.bbj

0100  declare BBjNumber A!
0110  declare BBjNumber B!

0200  A! = "abc"
0210  B! = 5
0220  print A! + B! ; rem runtime error

0300  C! = "abc"
0310  D! = 5
0320  print C! + D! ;  rem runtime error

0400  declare BBjAPI  X!
0410  X! = bbjapi()
0420  X!.getExistingNamespace( 44 ) ;  rem runtime error
```

**Figure 2.** Declare statement code sample

By adding the **declare** statements at line 100, 110, and 400, we are telling the compiler what type we want variables **A!**, **B!** and **X!** to represent in the program. This allows the compiler to recognize some of the problems that are in our program. Because we have not declared the type for the variables **C!** and **D!**, the compiler still cannot recognize the problem at line 300-320. If we run the compiler on this program using the **-t** option that enables type checking, we receive the output shown in **Figure 3**.

```
> bbjcpl -t demo2.bbj

demo2.bbj: type check error [Incompatible assignment from <BBjString> to
<BBjNumber>] at line 200 (6): A! = "abc"

demo2.bbj: type check error [No match for method
com.basis.bbj.proxies.BBjAPI.getExistingNamespace(<BBjInt>)] at line 420 (16):
x!.getExistingNamespace( 44 ) ;   rem runtime error
```

**Figure 3.** Type checking output using the -t option

This output tells us that at line 200 we have mistakenly assigned a numeric value to a variable that is meant to be a string variable and that at line 420, we have attempted to invoke a method that does not exist.

Also notice this output still does not give any information about our problem at line 300-320 because the compiler has no type information about the variables **C!** and **D!**. If we run the compiler with the **-t -W** options that enable type checking and related warning messages, it will list all the lines that have type check errors. In addition, it will also list all the lines that may have errors but could not be checked because there is no type information for their variables. Using the **-t -W** options, we receive the output displayed in **Figure 4**.

*continued...*

```
> bbjcpl -t -W demo2.bbj

demo2.bbj: type check error [Incompatible assignment from <BBjString> to
<BBjNumber>] at line 200 (6): A! = "abc"

demo2.bbj: type check warning [Undeclared variable: C!] at line 300 (10): C! ="abc"

demo2.bbj: type check warning [Undeclared variable: D!] at line 310 (11): D! = 5

demo2.bbj: type check warning [Undeclared variable: C!; Undeclared variable:
D!] at line 320 (12): print C! + D! ;  rem runtime error

demo2.bbj: type check error [No match for method
com.basis.bbj.proxies.BBjAPI.getExistingNamespace(<BBjInt>)] at line 420 (16):
x!.getExistingNamespace( 44 ) ;  rem runtime error
```

**Figure 4.** Type checking output using the -t -W options

This output tells us about the errors on line 200 and line 420, and warns us that lines 300-320 use variables that have not been declared and so could not be checked for errors.

## Summary

Using the **`declare`** verb along with the type check options of BBj 6.0 allows the developer to discover many programming problems at compile time which, if left undiscovered, would become runtime errors. Compile time errors are vastly preferred since they appear during development and provide an opportunity for correction before deploying the application. Runtime errors, on the other hand, occur during execution of the program – oftentimes in a production environment – and therefore, tend to be more costly and difficult to track down.

For a complete discussion of type checking errors and the new options for bbjcpl, refer to the *Type Checker Overview* at www.basis.com/solutions/TypeChecker.pdf

For more information about BBj custom objects see *A Primer for Using BBj Custom Objects* article on page 11.