

A Primer for Using BBJ Custom Objects

By David Wallwork

Partnership

Language/Interpreter

Database Management

Development Tools

System Administration

During the last several years, BASIS added a series of enhancements to BBJ® that provide new functionality ranging from embedded Java to transaction-based commit and rollback; from Web Services that execute BBJ applications to new development tools such as the IDE and AppBuilder; from new table or file types to data entry validation. Conspicuously absent from this list was the ability to write object-oriented code as BBJ native code without using embedded Java.

BBJ Custom Objects

BBJ 6.0 introduces a number of new verbs that allow the developer to write classes and objects within a BBJ program file, delivering fully object-oriented capability to Business BASIC. Classes defined in BBJ are referred to as custom classes and instances of a custom class are referred to as custom objects.

Custom objects in BBJ provide a full set of object-oriented features including:

- Protection levels (private, protected, public)
- Inheritance structures (extends)
- Interfaces structures (implements)
- Strongly-typed parameter lists and return values
- Multiple constructors
- Overloaded methods (accepting different parameter lists)
- Static variables
- Static methods
- Variable initialization
- Variable scope

A Custom Class

Understanding custom classes requires both a general knowledge of object-oriented programming and specific understanding of BBJ syntax. Since there is a large body of information already published and readily available about object-oriented programming, we will not discuss the general concepts (see a few of these resources listed at the end of this article). What is important for us to demonstrate in this article is the syntax used in BBJ to write an object-oriented program. The following two short examples demonstrate that syntax. A complete syntax definition may be found in the references at the end of this article.

The first example shown in **Figure 1** is a simple program that defines a custom class, instantiates an instance of that custom class (instantiates a custom object), and invokes a method on the instance. The output of this program prints **hello world** to the console.

```
declare Speaker speaker!  
speaker! = new Speaker()  
speaker!.speak()  
  
rem definition of the CustomClass Speaker  
class public Speaker  
    method public void speak()  
        print "hello world"  
    methodend  
classend
```



David Wallwork
Software Architect

Figure 1. Sample program **helloworld.src** that defines a custom class

continued...

The second example shown in **Figure 2** greets the user in Spanish, English, or French. Next, it prints information about the greeter and about the static value that holds the total number of guests greeted.

```

declare Greeter Pierre!
declare Greeter Joe!
declare Greeter Maria!
declare Greeter Host!

Pierre! = new FrenchGreeter("Pierre")
Joe! = new EnglishGreeter("Joe")
Maria! = new SpanishGreeter("Maria")
Host! = Joe!

while 1
    input "quel lingua? 0-espanol, 1-ingles, 2-frances ", language

    switch language
        case 0
            Host! = Maria!
            break
        case 1
            Host! = Joe!
            break
        case 2
            Host! = Pierre!
            break
    swend

    Host!.greet()

    print "you have been greeted by: ", Host!.tellName()
    print "you are guest number: ", Greeter.getTotalGreetingCount()
    print "number of greetings by this Greeter: ", Host!.getGreetingCount()
    print
wend

class public Greeter

    field private BBjString Greeting$
    field private BBjString Name$
    field private BBjNumber count
    field private static BBjNumber totalCount

    method public Greeter(BBjString p_greeting$, BBjString p_name$)
        #Greeting$ = p_greeting$
        #Name$ = p_name$
    methodend

    method public void greet()
        print
        print #Greeting$
        #count = #count + 1
        #totalCount = #totalCount + 1
    methodend

    method public static BBjNumber getTotalGreetingCount()
        methodret #totalCount
    methodend

    method public BBjString tellName()
        methodret #Name$
    methodend

    method public BBjNumber getGreetingCount()
        methodret #count
    methodend

classend

```

continued...

```

class public FrenchGreeter extends Greeter
    method public FrenchGreeter(BBjString p_name$)
        #super!("bon jour", p_name$)
    methodend
classend

class public EnglishGreeter extends Greeter
    method public EnglishGreeter(BBjString p_name$)
        #super!("hello ", p_name$)
    methodend
classend

class public SpanishGreeter extends Greeter
    method public SpanishGreeter(BBjString p_name$)
        #super!("buenos dias", p_name$)
    methodend
classend


```

Figure 2. Code sample `greeter.src` that greets users in multiple languages

This small program demonstrates a number of object-oriented concepts and how they are realized in BBj. The reader will notice the use of polymorphic classes (the various child classes of Greeter), static values and static methods (`totalGreetingCount`), class extension, and protection levels illustrated in Figure 3.

Summary

Though these examples assume an understanding of object-oriented programming, many readers may be unfamiliar with this concept. There is an abundance of well-written technical resources available, but BASIS recommends reviewing the sites listed below as a good starting point to learn the basic concepts.

More than any other of the changes in the recent past, the adding of object-oriented programming to BBx® technology has the greatest potential to impact the day-to-day development and the way developers build business applications. BBj custom objects can streamline the development process and promote code re-use, resulting in more robust applications getting to market faster than ever before. Truly, adding object-oriented programming takes BBx technology light-years ahead of traditional Business BASIC. 

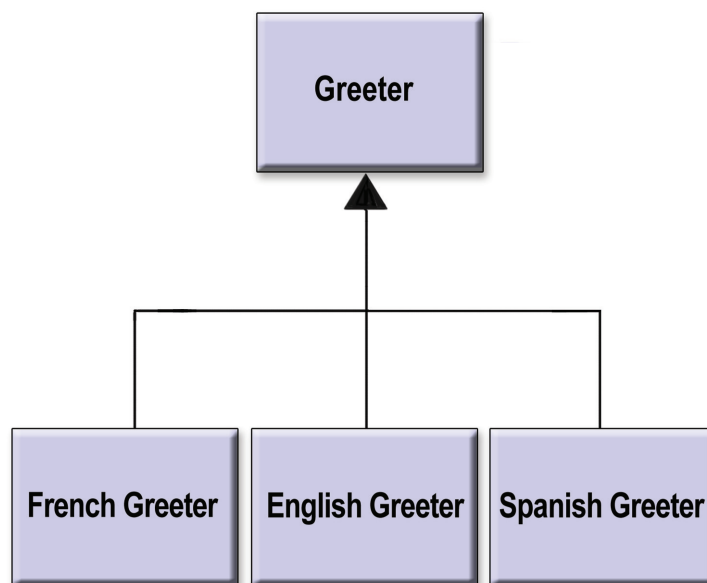


Figure 3. Using polymorphism in BBj



Download the sample programs referenced in this article from www.basis.com/advantage/mag_v10n1/usingco.zip

For an overview of how object-oriented programming techniques can help you write modules that are more robust and reusable, see java.sun.com/docs/books/tutorial/java/concepts

To learn more about custom objects and how to use them in BBj, refer to www.basis.com/solutions/BBj_CustomObjects.pdf and www.basis.com/solutions/BeginBBjObj.ppt