WIRTSCHAFTSUNIVERSITÄT WIEN
BAKKALAUREATSARBEIT

Titel der Bakkalaureatsarbeit:
BSF4Rexx: Creating TestUnits

Englischer Titel der Bakkalaureatsarbeit:
BSF4Rexx: Creating TestUnits

Verfasserin/Verfasser:	**Michael Lex**
Matrikel-Nr.:	**0552475**
Studienrichtung:	**J033 526 Bakkalaureat Wirtschaftsinformatik**
Kurs:	**0780 SBWL Kurs V - Management Information Systems**
Textsprache:	**Englisch**
Betreuerin/Betreuer:	**ao.Univ.Prof. Dr. Rony G. Flatscher**

Ich versichere:

dass ich die Bakkalaureatsarbeit selbstständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfe bedient habe.

dass ich die Ausarbeitung zu dem obigen Thema bisher weder im In- noch im Ausland (einer Beurteilerin/ einem Beurteiler zur Begutachtung) in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

dass diese Arbeit mit der vom Betreuer beurteilten Arbeit übereinstimmt.

Datum _____ Unterschrift _____

Table of contents

Table of figures

Table of code

1 Introduction

1.1 Initial words

This paper deals with the use of unit tests and especially focuses on unit tests under Rexx for BSF4Rexx. Starting an overview about the used technologies and how they work together. In addition this paper contains how to get a running system and how to do some tests on your own computer. Examples show how to run the unit tests and the related file listings are attached in the appendix.

Additionally there are some general information about unit tests for those who are not familiar with this kind of approach. Furthermore there is a chapter about unit tests in general explains what unit tests can be used for and why people should use unit tests and opportunities of unit tests.

The last part of this paper is about unit tests for BSF4Rexx. There you can find explanations about the author's work and findings.

1.2 Abstract

1.2.1 English

This paper is a result from the fact that there were no unit tests available for the Bean Scripting Framework for Rexx. It refers to the ability of unit testing and why they are so important for programs, especially for frameworks. Furthermore it holds a description how unit tests should be set up, practical examples and explanations for Unit Test for BSF4Rexx. The aim was to find out if there are difficulties on the current version of BSF4Rexx and so to help developers to test future versions accurate and quick.

Keywords: ooRexx, BSF4Rexx, unit tests, extreme testing

1.2.2 German

Dieses Paper ist entstanden, weil es zum Bean Scripting Framework für Rexx noch keine Unit Tests gibt. Es wird auf die Möglichkeiten von Unit Tests eingegangen und warum diese wichtig sind für Programme, speziell für Frameworks. Des Weiteren ist der Weg beschrieben, wie vernünftige Unit Tests aussehen sollten. Darüber hinaus gibt es und konkrete Beispiele und Erläuterungen für Unit Test für BSF4Rexx. Das Ziel hat darin bestanden bestehende Probleme und Fehler im Framework aufzudecken und aufzuzeigen.

Keywords: ooRexx, BSF4Rexx, Unit Tests, Extreme Testing

2 System environment

This chapter provides various information about the system environment that are used to create unit tests for BSF4Rexx.

2.1 Bean Scripting Framework

2.1.1 Overview

A Bean Scripting Framework (BSF) is a connecting link between Java and a scripting language. The use of BSF allows to access and to use Java objects and methods from the scripting language. Furthermore it is possible that Java uses the functionality of the scripting language and runs code from the scripting language. (Apache Software Foundation 2009)

So far the following scripting languages are supported by Jakarta BSF version 2.4 (Apache Software Foundation 2009):

BSF currently supports several scripting languages:

- Javascript (using Rhino ECMAScript, from the Mozilla project)
- NetRexx (an extension of the IBM REXX scripting language in Java)
- Commons JEXL
- Python (using Jython)
- Tcl (using Jacl)
- XSLT Stylesheets (as a component of Apache XML project's Xalan and Xerces)

In addition, the following languages are supported with their own BSF engines:

- Java (using BeanShell, from the BeanShell project)
- Groovy

- Groovy Monkey
- JLog (PROLOG implemented in Java)
- JRuby
- JudoScript
- ObjectScript
- ooRexx (Open Object Rexx), using BSF4Rexx.

2.1.2 Architecture

The Bean Scripting Framework basically holds two parts. There is the BSFManager and the second part is the BSFEngine. By creating an instance of the BSFManager class Java gains access to the functions of the scripting language. Furthermore the BSFManager manages the object registry and the access to Java objects from the scripting language. The BSFEngine is an interface that allows general script runs and objects registration for the scripting language.

2.2 ooRexx

Open Object Rexx (ooRexx) is an object orientated programming language or you might as well say a scripting language that works platform independent. It is based on Object Rexx which was released by IBM in the year of 2004.

2.2.1 History

Rexx stands for Restructured Extended Executor. Rexx served ooRexx and was developed between 1979 and 1982 by IBM. In 1990 the Rexx Language Association was founded to spread and promote Rexx. The foundation took place during the first Rexx symposium which took place in Stanford, Ca. Since then there has been an annual Rexx symposium where developers from all over the world meet and exchange their findings and experiences about Rexx. IBM published the source code

of Object Rexx in 2004 under the Common Public Licence (CPL). With this step IBM paved the way for the start of the development of Open Object Rexx (ooRexx). The first public release of ooRexx was in February of 2005. Right now the Rexx Language Association is negotiating with IBM to turn NetRexx into an open source project. (ooRexx 2009)

2.2.2 Overview

The idea behind ooRexx is to provide a platform independent scripting language that is easy to understand and to work with. For example, to do some output on the screen the only thing to do is to write the command SAY “Hello World” in the source file, and the text “Hello World” immediately appears on the screen. The reason why it is readable on the screen is that the intention of the programmer is that everyone can learn it quick. Another point that makes it very easy to learn writing Rexx scripts is that there are no complex things like data types compared to other programming languages. Rexx interprets every information as a string and there is no need to set definitions for variables. In sum there are only a few rules the user has to follow to get Rexx code working. In comparison to Java, Rexx is a programming language has an interpreter and not a compiler like Java. When Rexx code gets executed, every line of code is read after another and this makes it very fast. Rexx can also be used to automate and to script Windows applications. Rexx is an easy to learn programming language and it is very powerful although at first view it does not look that capable.

2.3 Java

2.3.1 History

The Java language project started in 1991 under the codename “Oak”. In 1993 when the first mosaic browser (the first browser which was able to display text and pictures at the same time without requesting the picture additionally) was launched Sun commissioned a team to do further developments on Oak. A new web browser called HotJava that was developed by Sun in Java was launched in 1994. The main idea behind this browser were Java Applets, distributed programming, machine independent and downloadable. After that in 1995 the Oak project has been renamed to its actual name Java. Furthermore in 1995 Sun made the first public release of Java with the slogan "Write once, run anywhere" which was created by Sun Microsystems to display the advantage of platform independent applications.

Sun released the first Java Developer Kit (JDK) 1.0 in 1996 with the AWT (Abstract Window Toolkit) and the JDBC (Java Database Connectivity) added. Only one year after the release of JDK 1.0 JDK 1.1 was released in 1997 which allowed commercial usage and Sun added new security features and access to local resources like the file system or printers. The next release was JDK 1.2 which is better known as Java 2 Platform in 1998. The J2ME (Java 2, Micro Edition) got released and allowed to write programs for embedded devices. Another edition that came up was the J2SE (Java 2, Standard Edition) where the Java foundation classes were available which included the drag&drop functionality, Swing, and the accessibility API for disabled persons. The third edition J2EE (Java 2, Enterprise Edition) includes JNDI (Java Naming and Directory Interface) to act with an authentication server LDAP (Lightweight Directory Access Protocol Server). Also JTI (Java Transaction Interface) was added to guarantee that the source code is executed completely or not at all which contributes to raise the transaction security. Also the Enterprise Java Beans were added which is a Java based standard for the component model (cooperation of distributed systems, transaction security...).

In the year 2000 JDK 1.3 was released with JPDA (Java Platform Debugger Architecture) and HotSpot added. JDK 1.4 which was release in the year 2002 supported IPv6, regular expressions, more integrated security and cryptography

extensions, a XML parser and some other major changes to go with the time of development. JDK version 5 was released in the year 2004 which was a big step because many significant new language features were added and the class library was enhanced.

A very important step took place in May 2007 because most of the Java technology was published under the GNU General Public License version 2 (GPLv2).

The version 5 is right now still supported by Sun but Sun announced to stop the support for it by the end of October 2009. The latest available version of Java is version 6 which was released in the year 2006 where for example the support of old Window 9x systems was dropped. The development of version 7 started in the year 2006 and the release is scheduled for 2010.

Through the ability to run Java applets within web pages Java soon became very popular and still enjoys great popularity. (Wikimedia Foundation 2009, Java version history)

2.3.2 Overview

Java is an object orientated programming language and the programmer has to take the object orientated approach. In ooRexx the programmer can decide whether to use the object orientated technology or to use write procedural program code. There are three different platforms of Java available which are the Standard Edition, Enterprise Edition and Micro Edition and each of the companies offers a very huge class library which is harmonized with each platform. Java, same as Rexx a portable language which means that it runs under different operating systems. To achieve that Java uses the Java Virtual Machine that and the fact that Java can be embed in websites is why it is also called as language of the World Wide Web. Java also brought up some new concepts like garbage collection, exceptions and hierarchic modules (packages). Moreover Java commands allow multi threading, graphic programming and applets.

There are two different types of java programs. There are Java applications that are stand alone software packages and there are Java applets that are Java programs that can only be executed with a Java capable browser or a Java applet viewer.

Many people who do not have a consolidated knowledge on computers might think that JavaScripts are also Java programs but JavaScript is another programming language that mostly is embedded in HTML documents.

Another big difference to ooRexx is that the Java source code has to be compiled. To run Java source code the user has to compile the source code first. Instead of an exe file (executable file under Windows) the compiler translates the source code to a class file that contains the byte code of the Java source file. After that the Java program can be executed by via the Java Virtual Machine. The Java Virtual Machine is available for different platforms like Windows, Linux, Mac OSX, Solaris and so on this makes Java platform independent. (Wikimedia Foundation 2009, Java (programming language))

2.4 BSF4Rexx

Bean Scripting Framework for Rexx (BSF4Rexx) adds the possibility to execute Java commands from Rexx and vice versa.

2.4.1 History

The roots of BSF4Rexx go back to the year 2000 when BSF4Rexx was developed as a proof of concept at the University of Essen (Germany). In the years 2002-2003 the version which was developed in Essen was displaced by the Augsburg version which was developed in Augsburg (Germany). This version is a revision of the version from Essen and both versions are compatible to each other. This new version allowed the programmer for the first time to start Java out of Rexx code, therefore there was no need for the user to have fundamental knowledge in Java programming. After a development time of three years in 2006 the Vienna version was published. Since this version there is no need to do type indications any more because Rexx does that all automatically. In the case that anyone needs to do type indication (for example if there is more than one constructor where the only difference is in the data types) there is the possibility to specify the data types explicit.

2.4.2 Overview

Through to the framework it is possible that a Rexx program uses Java classes and methods. Moreover it is possible that the programmer can for example deal with Java arrays in Rexx like the Java arrays were Rexx arrays. Due to the framework and the fact that Rexx is an easy to learn scripting language the user gains the possibility to use the whole functionality and power of Java without detailed knowledge in Java. The user just has to be able to handle the Java API[1] documentation.

To get the BSF support in the Rexx scripts it is necessary to include BSF.cls in the source code of the script. To include it just use the following line of code:

```
::requires BSF.cls
```

A small example for a Rexx script which is using Java is the following one:

```
call BSF.CLS
say .bsf4rexx~system.class~getProperty("java.version")
```

Code 1: Rexx script which gets Java version

This code produces an output like this:

```
1.6.0_11
```

The only thing this code makes is to get the Java system property which stores the version and outputs it.

[1] Application Programming Interface

2.4.3 Architecture

The following picture shows how the interaction between Rexx and Java is done.

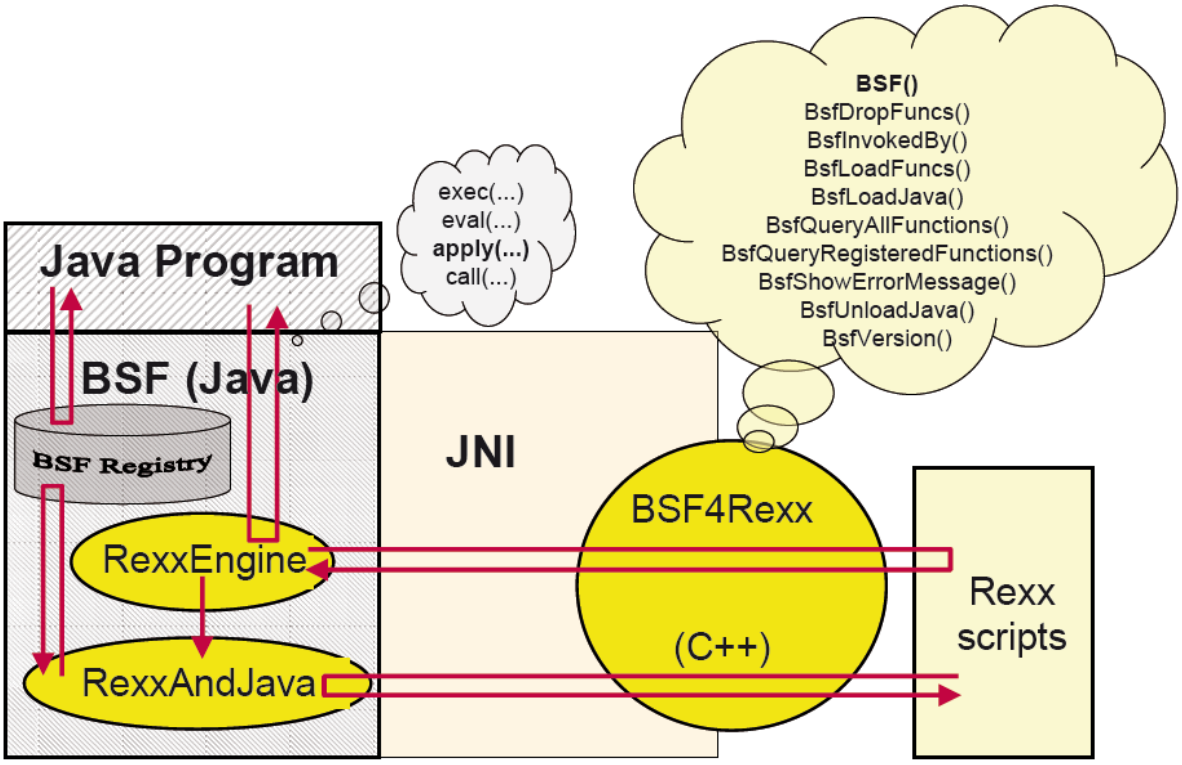

Figure 1: Architecture of BSF4Rexx

(Flatscher, Rony G., 2006, 17th Rexx symposium, Austin, USA)

There are two main fields, the “RexxEngine” and the “RexxAndJava”. The communication between Rexx and Java (for example to exchange objects) is done via the BSFManager.

2.5 ooRexxUnit

ooRexxUnit is a framework to make unit tests under Rexx. As for many other unit test frameworks JUnit was best practice example for the implementation of ooRexxUnit and therefore ooRexxUnit provides nearly the same functionality like JUnit. In the rough it was implemented to test the Rexx interpreter but also to provide Rexx programmers a framework to test their own applications. ooRexxUnit was implemented by Mark Miesfeld. The latest available snapshot is snapshot03, starting August 2008. People who want to support the development of the Rexx interpreter get all the available test units in a package included to have a look at them and to add more tests.

3 Installation

In this chapter is about how the reader becomes a running system from the scratch of all the software in use. All the software works on different platforms so it can be used under Windows, as well as under Linux and many other operating systems. The aim is that the reader has got a running system within minutes and does not have to read the whole documentation for the software packages. This manual focuses on computers running under Windows, but in most cases there is only a slight difference on for example Linux based computers. Therefore there are some remarks for Linux users so they also can use this manual. At the end of this chapter the reader should have a system where all the different components work together correctly. To get further information on installing the different software packages just take a look at the documentation provided on the websites of the software producer.

3.1 Java

Java is very common and used by millions of people. So before installing it be sure to check if it had not been installed before. The easiest and fastest way to check whether java has been installed or not is to open a shell and to run the following command:

```
java -version
```

If Java is already installed there will be an output like this:

```
java version "1.6.0_11"
Java(TM) SE Runtime Environment (build 1.6.0_11-b03)
Java HotSpot(TM) Client VM (build 11.0-b16, mixed mode,
sharing)
```

The Java version must not be lower than 1.4 because to run BSF4Rexx scripts it is necessary to have Java 1.4 or higher installed. If there is an error like command not found then there is no java installed. To install or update Java go to http://java.sun.com and download JDK or JRE to get BSF4Rexx scripts to run.

3.2 ooRexx

The easiest way to get ooRexx installed is to download one of the binary packages from http://www.oorexx.org/download.html. There are binary packages available for Windows, Linux, Mac OSX and Solaris. To get them installed run the setup under Windows or use the package manager of your operating system. Please bear in mind to uninstall any previous installed versions of ooRexx before you install a new version. There is no need to do some special settings to get ooRexx to run.

3.3 BSF4Rexx

To get the current version of BSF4Rexx go to http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/current/ and download BSF4Rexx_install.zip. In this directory listing there is also a lot of useful documentation for BSF4Rexx available.

Again, if there is a previous version of BSF4Rexx installed on the computer you have to uninstall it. First of all unzip the archive, open a shell and change to the BSF4Rexx directory. Now run

```
rexx setupBSF.rex
```

to get your install scripts. Under Windows run

```
installBSF.cmd
```

or under Linux run

```
installBSF.sh
```

Additional step for Linux users: copy the statement shown to your bash resource script to get BSF4Rexx to work (this change will take effect when you open a new shell session). To check if the installation works run:

```
infoBSF.rex
```

The output should look like the following:

```
Rexx interpreter:  [REXX-ooRexx_3.2.0(MT) 6.02 30 Oct 2007]
BSFManager         [244.20080704]
BSF4Rexx (DLL/so): [280.20080903
org/rexxla/bsf/engines/rexx]
Java Rexx engine:  [283.20080909 org.rexxla.bsf.engines.rexx
(org.apache.bsf)]
Java version:      [1.6.0_11]
Java home dir:     [C:\Programme\Java\jre6]
... cut here ...
```

To get rid of any errors or to get more detailed information read readmeBSF4Rexx.txt which is included in the archive which was downloaded before.

3.4 ooRexxUnit

The last software is ooRexxUnit. To download ooRexxUnit, which is hosted on SourceForge, go to http://sourceforge.net/project/showfiles.php?group_id=119701&package_id=251951 to download the zip archive. After unzipping run (under Windows)

```
setTestEnv.bat
```

or under Linux

```
setTestEnv.sh
```

to set the environment for the ooRexxUnit package. After this is done it is possible to run ooRexx unit tests. To test the installation run:

```
rexx testOORexx.rex
```

After a couple of minutes there should be an output that shows a statistic about the executed tests. This output should look similar to this one:

```
ooTest Framework - Automated Test of the ooRexx Interpreter


Interpreter: REXX-ooRexx_3.2.0(MT) 6.02 30 Oct 2007
ooRexxUnit:  2.0.0_3.2.0    ooTest: 1.0.0_3.2.0

Tests ran:           16616
Assertions:          537831
Failures:            33
Errors:              0
Skipped files:       0
```

3.5 Vim

Vim is just an editor, so there is no need to install Vim to work with the environment which was installed but Vim offers syntax highlighting for Rexx code. Vim is also an open source product and it is also available for different operating systems. To get Vim or gVim go to http://www.vim.org/ and download it.

4 Unit tests

This chapter provides an overview about unit tests and gives an answer to the following questions:

- What are unit tests?
- Why should anyone use unit tests?
- What are the different approaches on unit tests?

4.1 General information about software tests

Software testing is a very important part of software development because if the developed software is not tested there can be a lot of errors left since it is impossible to create faultless codes especially for bigger projects. Software projects had an experienced an enormous growth of size and complexity over the past few years. Therefore software testing is not as easy as someone might think, like just doing some clicks in the program to proof that it is working correctly. One of the approaches on software testing is to create unit tests which will be handled in more detail later on.

Nowadays more and more of a project's budget is used to finance the software testing. It is obvious that there is a higher effort on testing for software which is used in a very critical environment for instance where humans could suffer damages. As already mentioned software testing costs a lot of money but on the other hand software failures cost even more hence software testing is a value based principle. The main objective for software testing is to identify and weep out as many errors as possible and so to enhance the quality of the software. Moreover software tests do validation and verification of software. In other words it ensures that the software is processing what the user expects from the software and that the software sticks to its specification.

Due to the complexity and the high costs of software testing it can be seen as a project of the project. The first step is to evaluate a suitable test strategy. After the right strategy has been found the tests have to be planned which leads to the next step the design and the implementation of the tests. The last step is to run the tests

and to analyze them to finally create a report to document the status. There are many different types to measure the tests for example the number of defects per line of code or per function. (TU Wien Software Testing 2007)

4.2 General information about unit tests

Unit tests are a method to do software testing. Software testing is done to proof the functionality of software, to measure the quality and to find errors in the implementation. Unit tests is are an automated way to do software tests. They are small independent programs which are developed separately from the main program. Due to the fact that unit tests are small programs the tests are running automatically, therefore it is possible to run a huge pack of tests in almost no time wherefore a developer would need a lot of more time and cost a lot of money. The use of unit tests is very efficient for rewritten or updated programs. The name expresses that the tests are split up in small parts – so called units. Such a unit is the smallest testable part of a software implementation; for example single functions or classes. There are only two different types of results for a unit test. A unit test can whether be successful or can fail. There are different kinds of tests like to check if a result equals the expected result. Sometimes errors are provoked to tests and to proof that the error handling is correct and the tested function reacts and as a result it should display the error.

There are different approaches how to realize software projects. One of them is the method of extreme programming (XP). There the developers break down the whole software project to small parts, so called units, and implement them one after another. With every implemented part of the whole breakdown they bring the project a little bit closer to the goal. This approach stays in sync with the so called extreme testing approach for software testing. In comparison to extreme programming the focus for extreme testing is more on the testing aspect. For example it is allowed to leave some trivial tests out in extreme programming but in extreme testing there are more test cases. The programmer decides how detailed and how many tests there are for each test scenario. Doing extreme testing means that at every time the success rate of unit tests is 100%. It is not allowed that at any stage one single unit test fails. So to say testing takes the center stage of the project and not the

programming any more. The test cases are defined before the software implementation starts to ensure that the software provides the results which it has to provide.

To avoid any misunderstandings I would like to point out that there is no need to use one of these approaches (extreme programming or extreme testing) to do unit tests for testing the developed software. These approaches just integrate unit tests in their concepts.

Unit tests should not be seen as the solver of every problem because unit tests do not fit to every case. For example for some projects it is not reasonable to implement unit tests because it would take too much time to write the tests.

For almost every programming language there is a framework available to implement unit tests. The most famous product of a unit test framework is indeed JUnit, which is a framework to do unit tests in Java. The guide for most of the unit test frameworks which are available nowadays was JUnit. Therefore many frameworks use the same syntax as JUnit. (Wikimedia Foundation 2009, Unit testing)

4.3 How unit tests should look like

First of all it is obvious that there are tests that proof whether a result is the same as expected. This for example is a test of a calculator where test unit checks if “3+2” equals 5. Certainly this sounds trivial but unit tests cannot be trivial enough. In most cases there will not be enough time to write trivial tests but to ensure that a unit test suite is more or less complete there should be also trivial tests.

It is also very important to write test units where the limit values are tested and also a little bit lower values than the limit value and of course also with values that are a little bit higher.

Supplemental there should be unit tests implemented which knowingly provoke an error to test the error handling of the function. This functionality for example was not included at the beginning of the ooRexxUnit test framework but the developers recognized very fast that also this kind of test is very important.

As seen there are many different types of tests to proof that one function works correctly so the workload can be very high if every aspect is considered and it could happen that there are more lines of code for the test purposes than for the function which is actually tested.

4.4 Reasons to use unit tests

There are many different reasons why it absolutely makes sense to choose unit tests. When unit tests are in use while the project is implemented the developers are able to find mistakes very fast before the projects grows bigger and bigger and debugging gets harder and harder. Especially when there are many interlaced classes and methods it is easier to find errors provided that there are unit tests available for every case which could appear. In this case test units for single functions are written to make the testing more effective, in the correct work of the software as well as financial aspects. In addition there are test units for the functions which call the functions. So in the beginning all the parts are tested. In the next step the working together of all single parts in a body are tested. It is a kind of bottom-up strategy.

A good example where unit tests make sense is an interpreter or a framework like BSF4Rexx. Such programs will be enhanced from time to time and they need to reach equivalent results in a new version. When there are unit tests available for all these updated functions it is easy for the developer to assure that no errors are left because after the code has changed the developer simply runs the unit tests and can proof the code if the functionality changed after the latest changes of the source code. If the result of the unit tests is not the same any more then there is an error in the new implementation and as a result that programs which use the previous version of the tested software would not work correctly any more. The programmer can see exactly which test failed and therefore identify the error very fast and fix it. At this time when enough unit tests are available for a project the developers do not have to worry a lot about the testing because unit tests run automatically and the developers only have to analyze the results which is usually done within some minutes.

For unexperienced developers who start to participate in a programming project unit test could also be some kind of documentation because when they look at the unit

tests they can see what results the program should come up with. In addition also user of an open source project where unit tests are available can get some additional help out of the unit tests if there is for example some information in the documentation missing. But I would advise against this type of incomplete documentation because it is hard for the user to learn the rules of the unit tests so it would be better to provide some kind of well known documentation.

4.5 Negative aspects on unit testing

On the one hand it is easy to find errors which are a result of a wrong allocation but on the other hand it is hard to find logical errors because these cannot be detected automatically. Beside of that there is no guarantee that there are no errors in the software by using unit tests because only things get tested where there is an existing unit test. If there is no unit test for a case which could appear then it is not tested. If there is an error in one of the test units there would be the risk that the software operates wrong because the developer stuck to the test and the function has got a return value of 0 instead of 1.

For huge projects to run the risk of investing too much time in implementing test units is very high because there are usually more lines of code for the test unit to test one single line of code in the software.

The automated testing with unit tests is not the only thing how testing should be done and no one should only rely on this type of software testing. A good mix between the automated unit tests and manual testing is an excellent way to test the developed software.

A major problem for extreme testing driven projects is that if the customer wants to have a minor change after project has reached a certain progress it is very hard to adapt all the existing test units. The development has to go all the way back and do the steps again progressive to get to the same functionality it already had before.

5 BSF4Rexx test units

This chapter deals with the ooRexxUnit framework to create unit tests for Rexx applications and especially unit test for the BSF4Rexx framework. There will be explanations how the test framework works and what to bear in mind when creating unit tests and of course some examples of the tests I wrote to test the Bean Scripting Framework. Moreover the issues I discovered when I did the tests for BSF4Rexx will be discussed and analyzed.

For the tests which are described in the following the following versions of the tools were used:

- Java

```
java version "1.6.0_11"
Java(TM) SE Runtime Environment (build 1.6.0_11-b03)
Java HotSpot(TM) Client VM (build 11.0-b16, mixed mode,
sharing)
```

- ooRexx

```
REXX-ooRexx_3.2.0(MT) 6.02 30 Oct 2007
```

- BSF4Rexx

```
Arraywrapper:  org.rexxla.bsf.engines.rexx.ArrayWrapper
103.20060101
BSF manager:  org.apache.bsf.BSFManager 244.20080704
EnumerationWrapper:
org.rexxla.bsf.engines.rexx.EnumerationWrapper 102.20060101
Java4Rexx:  org.rexxla.bsf.engines.rexx.Java4Rexx
205.20060101
REXXAndJava:  org.rexxla.bsf.engines.rexx.RexxAndJava
283.20080909
REXXEngine:  org.rexxla.bsf.engines.rexx.RexxEngine
212.20080814
Supllier:  org.rexxla.bsf.engines.rexx.Supplier 103.20070707
```

- ooRexxUnit

```
ooRexxUnit:  2.0.0_3.2.0
ooTest: 1.0.0_3.2.0
```

For the tests for the new version of BSF4Rexx other versions were used which are described in detail in the corresponding chapter.

5.1 How ooRexxUnit works

First of all I would like to explain how the ooRexxUnit framework is build up and what its outputs mean.

The test framework basically consists of four classes. The TestResult, TestSuite, TestCase and the Assert class. The class TestResult is the class which holds the information for the statistic which can be seen after a test suite ran. All the logging for the tests are done by this class.

The TestSuite class brings the test cases together and combines them into one test suite. The test cases which are created for testing are a subclass of TestCase. Finally the Assert class provides all the functions which are usually used to for the implementation of test units.

5.1.1 Output of ooRexxUnit

It is easier to understand the following examples when you are familiar with the output of ooRexxUnit. Therefore you can find a description of the output in the following:

After the automated test suite is completed, the output looks like this:

```
ooTest Framework - Automated Test of the ooRexx Interpreter


Interpreter: REXX-ooRexx_3.2.0(MT) 6.02 30 Oct 2007
ooRexxUnit:  2.0.0_3.2.0    ooTest: 1.0.0_3.2.0

Tests ran:           16616
Assertions:          537831
Failures:            33
Errors:              0
Skipped files:       0
```

This means that there were in total 16.616 tests with 537.831 assertions and 33 failures occurred. These failures occur if a test does not pass. In other words the expected result is not the result what it was supposed to be. An error is displayed when there was something the interpreter has problems with a line of code. This is like if there would be an application which cannot be executed without errors. There is another more detailed output per failed test which looks like this:

```
[failure] [20090610 04:01:39.328000]
  Test:   TEST_DEMONSTRATION_FAIL
  Class:  demonstration.testGroup
  File:   C:\rexx\bsftest\playground\demonstration.testGroup
  Line:   26
  Failed: assertEquals
    Expected: [[0], identityHash="535806324"]
    Actual:   [[1], identityHash="535807656"]
```

The label “Test” shows the name of the test which failed. In the next line the class which threw the error is indicated and immediately under it the output shows the path and filename of the file where source code of the test is inside. Afterwards the line and the function (in this case assertEquals) is displayed. The last two lines are more

important because they show which value was expected and which value the result actually was.

5.1.2 Functions

The following functions can be used to create test units. There some more which are documented in ooRexxUnit documentation but this chapter should only provide a quick overview moreover there are also examples provided to get very fast an idea how the functions can be used. These nutshell examples are examples of the implementation which is added in the appendix of this paper.

assertEquals assertNotEquals

assertEquals provides two different possibilities to use it. The programmer can choose whether to provide an additional fail message or not. The call looks like this:

assertEquals(value1, value2)

assertEquals(failure_message, value1, value2)

assertNotEquals(value1, value2)

assertNotEquals(failure_message, value1, value2)

The function compares the two values provided as argument using the = operator. Where value1 represents the expected value and value2 represents the actual value. If the additional failure message is provided then the message would be a prefix of the expected value.

```
::method "test_3"
        n=BSF("new", "jad", "java.awt.Dimension")
        f=BSF("setFieldValue", "jad", "width", "10")
        f=BSF("getFieldValue", "jad", "width")
        self~assertEquals(10, f)
```

Code 2: assertEquals example

In this example the value of width is compared with 10 to proof if the setter and getter worked properly. The call would look like this if a failure message is provided:

```
assertEquals("this is an additional failure msg", 10, f)
```

The counterpart of assertEquals is assertNotEquals that works the same way like assertEquals does but value1 and value2 need to be different to pass the test.

assertFalse and assertTrue

assertFalse checks if the value of the provided Boolean is false. The counterpart is assertTrue which checks if the provided Boolean is true. Also here an additional failure message can be added. The syntax looks like this:

assertFalse(value)

assertFalse(failure_message, value)

assertTrue(value)

assertTrue(failure_message, value)

```
::method "test_1"
        call TIME "R"
        ret=BSF("sleep", 0.5)
        time_elapsed = TIME("R")
        condition=0
        IF (time_elapsed < 0.6) & (time_elapsed > 0.4)
THEN
        DO
                condition=1
        END
        self~assertTrue(condition)
```

Code 3: assertTrue example

This nutshell checks if the sleep function is activated as long as it should be. To test this the run time is determined and checked if it is within a certain range.

assertNull and assertNotNull

assertNull checks whether the provided argument is null[2] or not. To check if the provided argument is not null the function assertNotNull is provided. Again it is possible to add a failure message. The syntax looks like this:

assertNull(value)

assertNull(failure_message, value)

assertNotNull(value)

assertNotNull(failure_message, value)

```
::method "test_assertNotNull"
        arr=bsf.createArray("int.class", 5)
        r=BSF("arrayPut", arr, 12345, 2)
        self~assertNotNull(arr[3])
```

Code 4: assertNotNull example

This test proofs that the value which was saved in the array is not null. If it would be null there would be a failure and then the test would fail.

expectSyntax

After the execution of the expectSyntax method the framework expects that a syntax error is thrown from the interpreter. The method expectSyntax has to be executed before the function which provokes the error is executed. Please keep in mind that there were minor changes in the error codes in BSF4Rexx 4.0. The syntax looks like the following:

expectSyntax(error_code)

[2] In Rexx a null value is represented by a .nil object

```
::method "test_2"
        n=BSF("new", "jad", "java.awt.Dimension")
        errorCode=40.1
        self~expectSyntax(errorCode)
        f=BSF("getFieldValue", "jad", "foo")
```

Code 5: expectSyntax example

This test throws an error because the field “foo” is not known in the java.awt.Dimension class.

5.1.3 Writing test units

There are only a few simple rules to follow to get some working unit tests. First of all there has to be a certain layout for the file which looks like this one:

```
#!/usr/bin/rexx

  parse source . . fileSpec;

  group = .TestGroup~new(fileSpec)
  group~add(.template.testGroup)

  if group~isAutomatedTest then return group

  testResult = group~suite~execute~~print

return testResult
-- End of entry point.

::requires "ooTest.frm"     -- load the ooRexxUnit classes
::requires "BSF.cls"

::class "template.testGroup" subclass ooTestCase public

[... insert test methods here ...]

return
```

Code 6: Layout of a unit test file

To use this template just edit the two following lines:

```
group~add(.template.testGroup)

::class "template.testGroup" subclass ooTestCase public
```

Change the word template to the name of your test class. Users who do not test BSF4Rexx code can remove the ::requires “BSF.cls” line. There is a template provided with the ooRexxUnit package which is located under misc\Simplest.testGroup. In this template there are also a lot of comments that describe which line stands for what. Finally the unit tests can be implemented through inserting the code at area where the [… insert test methods here …] tag is.

The design always looks like the following example:

```
::method "test_example"
        [... evaluation ...]
        self~assertEquals(1, 1)
```

Code 7: Layout of a test unit

First of all a new method has to be created. The method name has to have the prefix “test”. In this example the name of the method is “test_example”. There are two restrictions for the method’s name. As already mentioned it has to start with “test” and the second one is that it has to be unique within a testGroup. If there are more than one test methods with the same name the following framework exception would be thrown.

```
[Framework exception] [20090610 14:16:32.296000]
  File: C:\rexx\blubb\version.testGroup
  Line: 1768
  Type: Trap Severity: Fatal
  Duplicate ::METHOD directive instruction
    29 *-* ::method "test_same"
  1768 *-* call (file)
```

At the end of the test unit usually the assertion is done. In the example Code 7: Layout of a test unit it is self~assertTrue(1, 1). The prefix self~ is necessary in the assertion call. Now the test case is completed and can be executed.

5.1.4 Run unit tests

To execute the written tests run

```
rexx testOORexx.rex
```

If the test framework should search for tests in the current working directory and its subfolders. To specify a folder where the tests are located or to run only tests from a certain subfolder the command looks like this:

```
rexx testOORexx.rex –R path\to\test\units
```

After that the output is displayed and it can be evaluated.

5.2 Attached tests

To execute the examples a working installation of the system environment which is described in chapter 3 is required. In the following paragraphs I will give an overview how to understand the way I built up the tests, which structure they have and general useful informations.

All tests are consecutively numbered to find them quick if the framework displays an error. Furthermore in some cases I combined some functions that belong together in one testGroup file. When I did this I tried to make it as self explanatory as possible. For example the two functions setPropertyValue and getPropertyValue are combined in the file propertyValue.testGroup. Moreover the tests are split up in the versions of BSF4Rexx and in procedural and object orientated. The directory structure looks like this:

```
bsftest
 |
 \--- 2.8
 |     |
 |     \--- oo[3]
 |     |
 |     \--- procedural
 |
 |
 \--- 4.0
       |
       \--- oo
       |
       \--- procedural
```

Due to the circumstance that the error codes were change from BSF4Rexx version 2.8 to version 4.0 the following local variable indicates the current used version. This local variable is defined at the beginning of every testGroup file.

For version 4.0:

```
.local~ver = 4
```

For version 2.8:

```
.local~ver = 2
```

[3] oo stands for object orientated

5.3 Discovered issues

In this chapter I would like to explain the errors and unexpected behaviors I have discovered using the implemented test units.

The following output was produced by the framework. More details about the failed tests are provided in the correlating sub chapters.

```
C:\rexx>testOORexx.rex -R bsftest\2.8
Searching for test containers....
Executing automated test suite........

ooTest Framework - Automated Test of the ooRexx Interpreter

   980 *-* return BSF("unregisterBean",
self~objectname)~substr(4)
Error 40:  Incorrect call to routine

Interpreter: REXX-ooRexx_3.2.0(MT) 6.02 30 Oct 200Error
40.1:  External routine "BSF" failed7

ooRexxUnit:  2.0.0_3.2.0        ooTest: 1.0.0_3.2.0

Tests ran:           757
Assertions:          719
Failures:            13
Errors:              0
Skipped files:       0

[failure] [20090619 01:25:17.015000]
  Test:   TEST_10
  Class:  eventText.testGroup
  File:   C:\rexx\bsftest\2.8\oo\eventText.testGroup
  Line:   1305
  Failed: expectSyntax
    Expected: SYNTAX 40.1
    Actual:   Not raised

[failure] [20090619 01:25:17.015000]
  Test:   TEST_11
  Class:  eventText.testGroup
  File:   C:\rexx\bsftest\2.8\oo\eventText.testGroup
  Line:   1305
  Failed: expectSyntax
    Expected: SYNTAX 40.1
    Actual:   Not raised

[failure] [20090619 01:25:17.031000]
```

```
  Test:   TEST_12
  Class:  eventText.testGroup
  File:   C:\rexx\bsftest\2.8\oo\eventText.testGroup
  Line:   1305
  Failed: expectSyntax
    Expected: SYNTAX 40.1
    Actual:   Not raised

[failure] [20090619 01:25:17.031000]
  Test:   TEST_13
  Class:  eventText.testGroup
  File:   C:\rexx\bsftest\2.8\oo\eventText.testGroup
  Line:   1305
  Failed: expectSyntax
    Expected: SYNTAX 40.1
    Actual:   Not raised

[failure] [20090619 01:25:17.031000]
  Test:   TEST_9
  Class:  eventText.testGroup
  File:   C:\rexx\bsftest\2.8\oo\eventText.testGroup
  Line:   1305
  Failed: expectSyntax
    Expected: SYNTAX 40.1
    Actual:   Not raised

[failure] [20090619 01:25:17.031000]
  Test:   TEST_12
  Class:  FieldValue.testGroup
  File:   C:\rexx\bsftest\2.8\oo\FieldValue.testGroup
  Line:   1305
  Failed: expectSyntax
    Expected: SYNTAX 40.1
    Actual:   Not raised

[failure] [20090619 01:25:17.046000]
  Test:   TEST_2
  Class:  setRexxNullString.testGroup
  File:   C:\rexx\bsftest\2.8\oo\setRexxNullString.testGroup
  Line:   1305
  Failed: expectSyntax
    Expected: SYNTAX 40.1
    Actual:   Not raised

[failure] [20090619 01:25:19.656000]
  Test:   TEST_5
  Class:  sleep.testGroup
  File:   C:\rexx\bsftest\2.8\oo\sleep.testGroup
  Line:   1305
```

```
  Failed: expectSyntax
    Expected: SYNTAX 40.1
    Actual:   Not raised

[failure] [20090619 01:25:20.281000]
  Test:   TEST_2
  Class:  BsfInvokedBy.testGroup
  File:
C:\rexx\bsftest\2.8\procedural\BsfInvokedBy.testGroup
  Line:   1305
  Failed: expectSyntax
    Expected: SYNTAX 40.1
    Actual:   Not raised

[failure] [20090619 01:25:20.359000]
  Test:   TEST_2
  Class:  BSFVersion.testGroup
  File:
C:\rexx\bsftest\2.8\procedural\BSFVersion.testGroup
  Line:   1305
  Failed: expectSyntax
    Expected: SYNTAX 40.1
    Actual:   Not raised

[failure] [20090619 01:25:20.359000]
  Test:   TEST_10
  Class:  eventText.testGroup
  File:   C:\rexx\bsftest\2.8\procedural\eventText.testGroup
  Line:   1305
  Failed: expectSyntax
    Expected: SYNTAX 40.1
    Actual:   Not raised

[failure] [20090619 01:25:20.359000]
  Test:   TEST_13
  Class:  eventText.testGroup
  File:   C:\rexx\bsftest\2.8\procedural\eventText.testGroup
  Line:   1305
  Failed: expectSyntax
    Expected: SYNTAX 40.1
    Actual:   Not raised

[failure] [20090619 01:25:20.359000]
  Test:   TEST_9
  Class:  eventText.testGroup
  File:   C:\rexx\bsftest\2.8\procedural\eventText.testGroup
  Line:   1305
  Failed: expectSyntax
    Expected: SYNTAX 40.1
```

```
    Actual:   Not raised


Interpreter: REXX-ooRexx_3.2.0(MT) 6.02 30 Oct 2007
ooRexxUnit:  2.0.0_3.2.0        ooTest: 1.0.0_3.2.0

Tests ran:           757
Assertions:          719
Failures:            13
Errors:              0
Skipped files:       0

File search:        00:00:02.282000
Suite construction: 00:00:00.031000
Test execution:     00:00:06.453000
Total time:         00:00:09.110000
```

5.3.1 unregisterBean error

This error was very tricky and really time consuming because it did not show up every time. The following error showed up:

```
980 *-* return BSF("unregisterBean",
self~objectname)~substr(4)
Error 40:  Incorrect call to routine
Error 40.1:  External routine "BSF" failed
```

At the beginning I thought that there is one specific test that causes this error and started to put one test after another within comment tags. Since the error disappeared it seemed to be that the test which was responsible for the error was identified. However at that time I did not know that the error appears and disappears per random. Due to the behavior that it did not show up every time I wrote a small batch script which ran the tests 50 times succession but I had to experience that sometimes the error did not show up until the 54th cycle. Later on it was quite plain to me that the error only showed up when there were many tests executed. As a result of these hard to handle circumstances it was not possible to figure out what exactly caused this error. Another approach which ran into no substance was to kill the rxapi.exe to have a clean new instance of ooRexx API service running.

As an explanation this error occurs because BSF wants to unregister an object which is not present any more although it should be present. In all likelihood the error is present because the ooRexx interpreter does not work properly. Before writing about anything else it has to be mentioned that this error does not occur any more in ooRexx 4.0 and BSF4Rexx 4.0.

5.3.2 pollEventText

When I wrote and executed the tests I discovered that there is an unexpected behavior in the pollEventText function. According to the BSF4Rexx refcard[4] the function has to be used in this way: (Flatscher 2008, Reference Card)

```
return_value = BSF("pollEventText" [, timeout_in_msec])
```

Code 8: pollEventText usage

The problem is that the function accepts unlimited arguments without notifying the developer that he or she used more arguments than arguments are effective. Beside of that the function works fine. The following test should throw the error code 40.1 but it does not.

```
::method "test_13"
        errorCode=40.1
        self~expectSyntax(errorCode)
        e=BSF("pollEventText", 1, 1, 1, 1, 1, 1, "foo")
```

Code 9: pollEventText test example 1

As shown here it is possible to provide as many arguments of any type as wanted with the only exception that the second argument has to be a number. If the second argument is not a number the error code 40.1 would be thrown.

Moreover it is possible to provide a negative number as timeout which should not be possible either. The correlating test looks like this one:

[4] BSF4Rexx reference card which is included in the BSF4Rexx package (refcardBSF4Rexx.pdf)

```
::method "test_10"
        e=BSF("postEventText", "eventtest")
        errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("pollEventText", -500)
```

Code 10: pollEventText test example 2

Both of the tests should fail but they do not.

5.3.3 getFieldValue

This test should raise an error but it does not. For getFieldValueStrict there is an invalid type indicator supplied. There was no “foo” type registered therefore the developer should be informed that the argument he or she supplied is invalid. This method can be found in the file bsftest\2.8\oo\fieldValue.testGroup.

```
::method "test_12"
        show_msg=BsfShowErrorMessage(0)
        .bsf~bsf.import("java.awt.Dimension", "jad")
        ftf=.jad~new
        IF .ver = 4 THEN errorCode=40.900; ELSE
errorCode=40.1
        self~expectSyntax(errorCode)
        f=ftf~bsf.getFieldValueStrict("width", "foo")
```

Code 11: OO getFieldValue

5.3.4 setRexxNullString

Again this issue has to do something with arguments. According to the reference card there should be only one argument supplied namely the value of the new null string but there is no error raised if there is more than one argument supplied. This test is located in the file bsftest\2.8\oo\setRexxNullString.testGroup.

```
::method "test_2"
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        nns=.bsf~bsf.setRexxNullString("newnullstring", "foo")
```

Code 12: oo setRexxNullString

5.3.5 Sleep & BsfInvokedBy & BSFVersion

These tests should also provoke an error. It is the same issue like the setRexxNullString issue. The functions BsfInvokedBy and BSFVersion should not accept any arguments. These tests are located in the files bsftest\2.8\oo\sleep.testGroup, bsftest\2.8\proc\BsfInvokedBy.testGroup and bsftest\2.8\proc\BSFVersion.testGroup

```
::method "test_5"
        show_msg=BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        ret=.bsf~bsf.sleep(0,1)
```

Code 13: oo sleep

```
::method "test_2"
        IF .ver = 4 THEN errorCode=88.922; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BsfInvokedBy("foo")
```

Code 14: BsfInvokedBy

```
::method "test_2"
        IF .ver = 4 THEN errorCode=88.922; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSFVersion("foo")
```

Code 15: BSFVersion

5.4 BSF4Rexx 4.0

For the BSF4Rexx 4.0 test I used Windows XP with SP3 virtualized with VM Ware. Since BSF4Rexx 4.0 is under development right now (June 2009) the following tests were not run with the final version.

BSF4Rexx version:

```
Rexx interpreter:  [REXX-ooRexx_4.0.0(MT) 6.03 15 Jun 2009]
BSFManager         [244.20080704]
BSF4Rexx (DLL/so): [400.20090608
org/rexxla/bsf/engines/rexx]
Java Rexx engine:  [400.20090608 org.rexxla.bsf.engines.rexx
(org.apache.bsf)]
```

The following output was produced by the framework when I ran the attached unit tests:

```
C:\Rexx>testOORexx.rex -R bsftest
Searching for test containers/// ===> ===> ===> in
bsfLoader(), tid=[912], c_rii_ID=[7EEE5458] ...
..
Executing automated test suite.........

ooTest Framework - Automated Test of the ooRexx Interpreter


Interpreter: REXX-ooRexx_4.0.0(MT) 6.03 15 Jun 2009
ooRexxUnit:  2.0.0_3.2.0        ooTest: 1.0.0_3.2.0

Tests ran:           757
Assertions:          726
Failures:            5
Errors:              0
Skipped files:       0

[failure] [20090619 01:14:55.249000]
  Test:   TEST_11
  Class:  eventText.testGroup
  File:   C:\Rexx\bsftest\4.0\oo\eventText.testGroup
  Line:   1305
  Failed: expectSyntax
    Expected: SYNTAX 40.900
    Actual:   Not raised
```

```
[failure] [20090619 01:14:55.249000]
  Test:   TEST_13
  Class:  eventText.testGroup
  File:   C:\Rexx\bsftest\4.0\oo\eventText.testGroup
  Line:   1305
  Failed: expectSyntax
    Expected: SYNTAX 40.900
    Actual:   Not raised

[failure] [20090619 01:14:55.249000]
  Test:   TEST_12
  Class:  FieldValue.testGroup
  File:   C:\Rexx\bsftest\4.0\oo\FieldValue.testGroup
  Line:   1305
  Failed: expectSyntax
    Expected: SYNTAX 40.900
    Actual:   Not raised

[failure] [20090619 01:14:55.342000]
  Test:   TEST_2
  Class:  setRexxNullString.testGroup
  File:   C:\Rexx\bsftest\4.0\oo\setRexxNullString.testGroup
  Line:   1305
  Failed: expectSyntax
    Expected: SYNTAX 40.900
    Actual:   Not raised

[failure] [20090619 01:14:57.952000]
  Test:   TEST_5
  Class:  sleep.testGroup
  File:   C:\Rexx\bsftest\4.0\oo\sleep.testGroup
  Line:   1305
  Failed: expectSyntax
    Expected: SYNTAX 40.900
    Actual:   Not raised


Interpreter: REXX-ooRexx_4.0.0(MT) 6.03 15 Jun 2009
ooRexxUnit:  2.0.0_3.2.0        ooTest: 1.0.0_3.2.0

Tests ran:           757
Assertions:          726
Failures:            5
Errors:              0
Skipped files:       0

File search:        00:00:00.360000
Suite construction: 00:00:00.031000
Test execution:     00:00:07.438000
```

```
Total time:         00:00:08.063000

\\\  <=== <=== <=== in bsfUnloader() tid=[912],
c_rii_ID=[7F503CE8] ...
/// ===> ===> ===> in bsfLoader(), tid=[912],
c_rii_ID=[7F503CE8] ...
```

On the whole there were no critical errors. The issue unregister bean error which is described in chapter 5.3.1 is gone now which leads to the speculation that this error was an error in the ooRexx interpreter. 3.2.0. To get further information about the failed tests check the previous chapter since they are already described there.

6 Conclusion

Software tests become more and more important since software projects are getting bigger and bigger. The creation of test units assures that the produced source code provides the expected results. Due to the automation of unit test driven software testing, the developer can check the written source code very fast and the results can be evaluated and statistically analyzed. This also helps to measure and improve the quality of the software. Especially developers who refactor their code can benefit from the power of unit tests. After the code is refactored the only thing the developer needs to do is to execute the written tests that checks whether the application is still providing the same results as before or not. However it is important to spend enough time in writing test units and especially to pay attention that there are no mistakes in the test units otherwise the results could be wrong without recognizing or manpower is wasted for searching a mistake which is actually not really present.

BSF4Rexx had hardly any errors and the errors which occurred were not essential because the functions provided the results which were expected therefore a normal user of BSF4Rexx would not have seen it with the upmost probability.

7 References

Apache Software Foundation (2009): Jakarta BSF - Bean Scripting Framework. http://jakarta.apache.org/bsf/index.html, retrieved on 2009-04-27.

Flatscher, Rony G. (2008): Reference Card Vienna Version of BSF4Rexx.

ooRexx (2009): Open Object Rexx. http://www.oorexx.org, retrieved on 2009-05-04.

Wikimedia Foundation (2009): Java (programming language). http://en.wikipedia.org/w/index.php?title=Java_(programming_language)&oldid=297600088, retrieved on 2009-06-10.

Wikimedia Foundation (2009): Java (version history). http://en.wikipedia.org/w/index.php?title=Java_version_history&oldid=297591678, retrieved on 2009-06-10.

TU Wien Department of Rechnergestützte Automation (2007): Software Testing. http://www.inso.tuwien.ac.at/uploads/media/Vorlesung1.pdf, retrieved on 2009-06-11.

Wikimedia Foundation (2009): Unit Testing. http://en.wikipedia.org/w/index.php?title=Unit_testing&oldid=296803728, retrieved on 2009-06-12.

8 Appendix

8.1 Project management

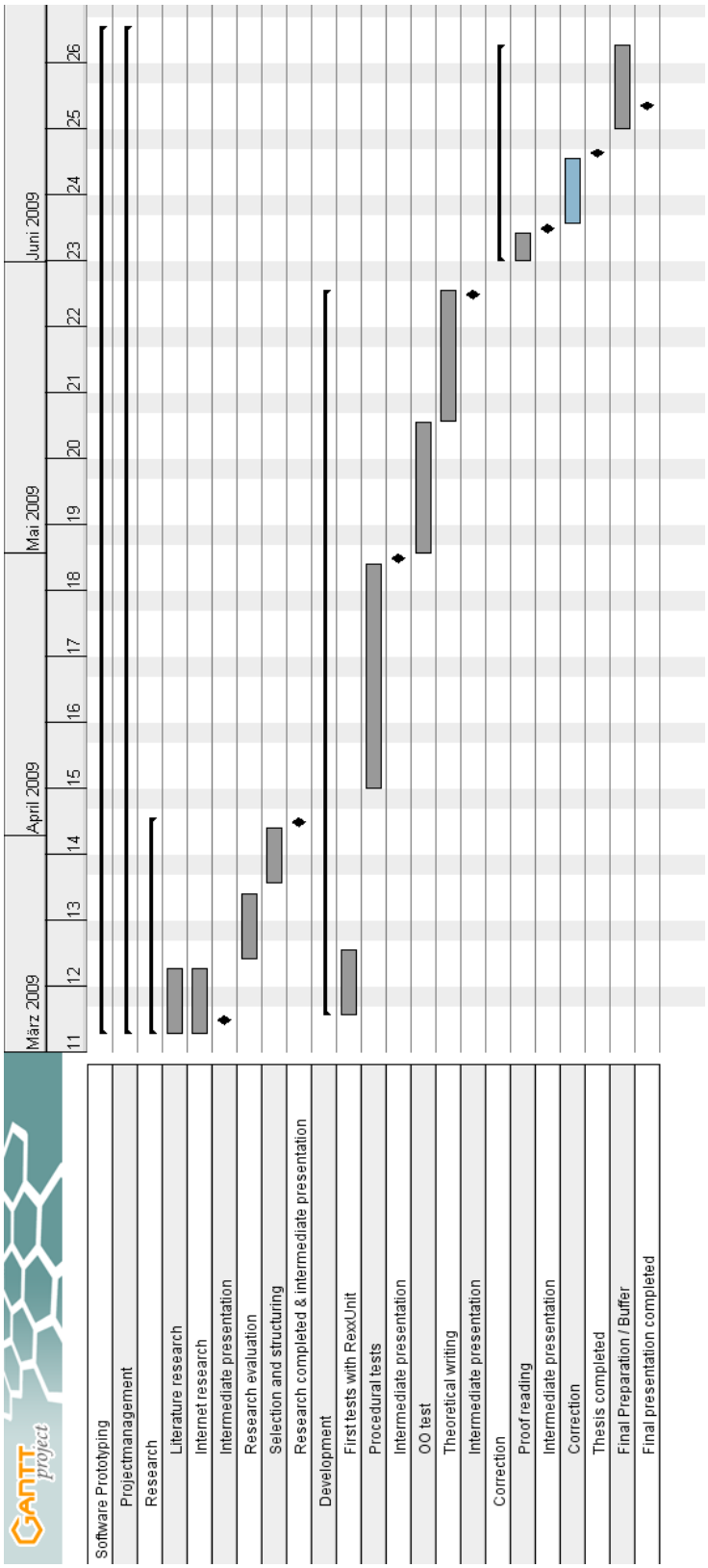

Figure 2: Gantt Chart

8.2 Source Code of procedural tests BSF4Rexx 2.8

Array.testGroup

```
#!/usr/bin/rexx
  parse source . . fileSpec;

  group = .TestGroup~new(fileSpec)
  group~add(.Array.testGroup)

  if group~isAutomatedTest then return group

  testResult = group~suite~execute~~print

return testResult
-- End of entry point.

::requires "ooTest.frm"      -- load the ooRexxUnit classes
::requires "BSF.cls"

::class "Array.testGroup" subclass ooTestCase public

::method "test_1"
        .local~ver = 2

        arr=bsf.createArray("int.class", 5)
        r=BSF("arrayPut", arr, 12345, 2)
        self~assertEquals(arr[3], 12345)

::method "test_2"
        show_msg = BsfShowErrorMessage(0)
        arr=bsf.createArray("int.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 12345, 7)

::method "test_3"
        arr=bsf.createArray("int.class", 5, 5)
        r=BSF("arrayPut", arr, 12345, 2, 2)
        self~assertEquals(arr[3,3], 12345)

::method "test_4"
        arr=bsf.createArray("int.class", 5, 5, 5)
        r=BSF("arrayPut", arr, 12345, 2, 2, 2)
        self~assertEquals(arr[3,3,3], 12345)

::method "test_5"
        arr=bsf.createArray("int.class", 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 12345, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3], 12345)

::method "test_6"
        arr=bsf.createArray("int.class", 5, 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 12345, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3], 12345)

::method "test_7"
        arr=bsf.createArray("int.class", 5, 5, 5, 5, 5, 5)
```

```
        r=BSF("arrayPut", arr, 12345, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3], 12345)

::method "test_8"
        arr=bsf.createArray("int.class", 5, 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 12345, 2, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3,3], 12345)

::method "test_9"
        arr=bsf.createArray("int.class", 128623)
        r=BSF("arrayPut", arr, 12345, 54321)
        self~assertEquals(arr[54322], 12345)

::method "test_10"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("int.class", foo)

::method "test_11"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("int.class", -1)

::method "test_12"
        arr=bsf.createArray("int.class", 0)
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 12345, 0)

::method "test_13"
        arr=bsf.createArray("int.class", 5)
        r=BSF("arrayPut", arr, -12345, 2)
        self~assertEquals(arr[3], -12345)

::method "test_14"
        arr=bsf.createArray("int.class", 5)
        r=BSF("arrayPut", arr, 2147483647, 2)
        self~assertEquals(arr[3], 2147483647)

::method "test_15"
        arr=bsf.createArray("int.class", 5)
        r=BSF("arrayPut", arr, '-2147483648', 2)
        self~assertEquals(arr[3], '-2147483648')

::method "test_16"
        arr=bsf.createArray("int.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, '2147483648', 2)

::method "test_17"
        arr=bsf.createArray("int.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, '-2147483649', 2)
```

```
::method "test_18"
        show_msg = BsfShowErrorMessage(0)
        arr=bsf.createArray("int.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, foo, 2)

::method "test_19"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("int.class", 5)
        r=BSF("arrayPut", arr, "foo", 2)

::method "test_20"
        arr=bsf.createArray("int.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 12345, -1)

::method "test_21"
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("foo", 5)

::method "test_22"
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", foo, 12345, 2)

::method "test_23"
        arr=bsf.createArray("int.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 12345)

::method "test_24"
        arr=bsf.createArray("int.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr)

::method "test_25"
        arr=bsf.createArray("int.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, , )

::method "test_26"
        arr=bsf.createArray("String.class", 128623)
        r=BSF("arrayPut", arr, 12345, 54321)
        self~assertEquals(arr[54322], 12345)

::method "test_27"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("String.class", foo)
```

```
::method "test_28"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("String.class", -1)

::method "test_29"
        arr=bsf.createArray("String.class", 0)
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, "foo", 0)

::method "test_30"
        arr=bsf.createArray("String.class", 5)
        r=BSF("arrayPut", arr, -12345, 2)
        self~assertEquals(arr[3], -12345)

::method "test_31"
        arr=bsf.createArray("String.class", 5)
        r=BSF("arrayPut", arr, "foo", 2)
        self~assertEquals(arr[3], "foo")

::method "test_32"
        arr=bsf.createArray("String.class", 5)
        r=BSF("arrayPut", arr, "foo", 2)
        self~assertNotEquals(arr[3], "bar")

::method "test_33"
        arr=bsf.createArray("String.class", 5)
        r=BSF("arrayPut", arr, '"', 2)
        self~assertEquals(arr[3], '"')

::method "test_34"
        arr=bsf.createArray("String.class", 5)
        r=BSF("arrayPut", arr, "'", 2)
        self~assertEquals(arr[3], "'")

::method "test_35"
        arr=bsf.createArray("String.class", 5)
        r=BSF("arrayPut", arr, "AAAaaa", 2)
        self~assertEquals(arr[3], "AAAaaa")

::method "test_36"
        arr=bsf.createArray("boolean.class", 128623)
        r=BSF("arrayPut", arr, .true, 54321)
        self~assertEquals(arr[54322], .false)

::method "test_37"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("boolean.class", foo)

::method "test_38"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("boolean.class", -1)
```

```
::method "test_39"
        arr=bsf.createArray("boolean.class", 0)
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 1, 0)

::method "test_40"
        arr=bsf.createArray("boolean.class", 5)
        r=BSF("arrayPut", arr, .false, 2)
        self~assertEquals(arr[3], .false)

::method "test_41"
        arr=bsf.createArray("boolean.class", 5)
        r=BSF("arrayPut", arr, .true, 2)
        self~assertEquals(arr[3], .false)

::method "test_42"
        arr=bsf.createArray("boolean.class", 5)
        r=BSF("arrayPut", arr, 0, 2)
        self~assertEquals(arr[3], 0)

::method "test_43"
        arr=bsf.createArray("boolean.class", 5)
        r=BSF("arrayPut", arr, 2, 2)
        self~assertEquals(arr[3], .false)

::method "test_44"
        show_msg = BsfShowErrorMessage(0)
        arr=bsf.createArray("boolean.class", 5)
        r=BSF("arrayPut", arr, "foo", 2)
        self~assertEquals(arr[3], .false)

::method "test_45"
        arr=bsf.createArray("boolean.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 1, -1)

::method "test_46"
        arr=bsf.createArray("byte.class", 5)
        r=BSF("arrayPut", arr, 123, 2)
        self~assertEquals(arr[3], 123)

::method "test_47"
        show_msg = BsfShowErrorMessage(0)
        arr=bsf.createArray("byte.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 123, 7)

::method "test_48"
        arr=bsf.createArray("byte.class", 5, 5)
        r=BSF("arrayPut", arr, 123, 2, 2)
        self~assertEquals(arr[3,3], 123)

::method "test_49"
        arr=bsf.createArray("byte.class", 5, 5, 5)
```

```
        r=BSF("arrayPut", arr, 123, 2, 2, 2)
        self~assertEquals(arr[3,3,3], 123)

::method "test_50"
        arr=bsf.createArray("byte.class", 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 123, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3], 123)

::method "test_51"
        arr=bsf.createArray("byte.class", 5, 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 123, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3], 123)

::method "test_52"
        arr=bsf.createArray("byte.class", 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 123, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3], 123)

::method "test_53"
        arr=bsf.createArray("byte.class", 5, 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 123, 2, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3,3], 123)

::method "test_54"
        arr=bsf.createArray("byte.class", 128623)
        r=BSF("arrayPut", arr, 123, 54321)
        self~assertEquals(arr[54322], 123)

::method "test_55"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("byte.class", foo)

::method "test_56"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("byte.class", -1)

::method "test_57"
        arr=bsf.createArray("byte.class", 0)
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 123, 0)

::method "test_58"
        arr=bsf.createArray("byte.class", 5)
        r=BSF("arrayPut", arr, -123, 2)
        self~assertEquals(arr[3], -123)

::method "test_59"
        arr=bsf.createArray("byte.class", 5)
        r=BSF("arrayPut", arr, 127, 2)
        self~assertEquals(arr[3], 127)

::method "test_60"
        arr=bsf.createArray("byte.class", 5)
```

```
        r=BSF("arrayPut", arr, -128, 2)
        self~assertEquals(arr[3], -128)

::method "test_61"
        arr=bsf.createArray("byte.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 128, 2)

::method "test_62"
        arr=bsf.createArray("byte.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, -129, 2)

::method "test_63"
        arr=bsf.createArray("byte.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 255, 2)

::method "test_64"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("byte.class", 5)
        r=BSF("arrayPut", arr, "foo", 2)

::method "test_65"
        arr=bsf.createArray("byte.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 123, -1)

::method "test_66"
        arr=bsf.createArray("char.class", 5)
        r=BSF("arrayPut", arr, 'a', 2)
        self~assertEquals(arr[3], 'a')

::method "test_67"
        show_msg = BsfShowErrorMessage(0)
        arr=bsf.createArray("char.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 'a', 7)

::method "test_68"
        arr=bsf.createArray("char.class", 5, 5)
        r=BSF("arrayPut", arr, 'a', 2, 2)
        self~assertEquals(arr[3,3], 'a')

::method "test_69"
        arr=bsf.createArray("char.class", 5, 5, 5)
        r=BSF("arrayPut", arr, 'a', 2, 2, 2)
        self~assertEquals(arr[3,3,3], 'a')

::method "test_70"
        arr=bsf.createArray("char.class", 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 'a', 2, 2, 2, 2)
```

```
        self~assertEquals(arr[3,3,3,3], 'a')

::method "test_71"
        arr=bsf.createArray("char.class", 5, 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 'a', 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3], 'a')

::method "test_72"
        arr=bsf.createArray("char.class", 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 'a', 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3], 'a')

::method "test_73"
        arr=bsf.createArray("char.class", 5, 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 'a', 2, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3,3], 'a')

::method "test_74"
        arr=bsf.createArray("char.class", 128623)
        r=BSF("arrayPut", arr, 'a', 54321)
        self~assertEquals(arr[54322], 'a')

::method "test_75"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("char.class", foo)

::method "test_76"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("char.class", -1)

::method "test_77"
        arr=bsf.createArray("char.class", 0)
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 'a', 0)

::method "test_78"
        arr=bsf.createArray("char.class", 5)
        r=BSF("arrayPut", arr, '1', 2)
        self~assertEquals(arr[3], '1')

::method "test_79"
        arr=bsf.createArray("char.class", 5)
        r=BSF("arrayPut", arr, '"', 2)
        self~assertEquals(arr[3], '"')

::method "test_80"
        show_msg = BsfShowErrorMessage(0)
        arr=bsf.createArray("char.class", 5)
        r=BSF("arrayPut", arr, "foo", 2)
        self~assertEquals(arr[3], 'f')

::method "test_81"
        arr=bsf.createArray("char.class", 5)
```

```
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 'a', -1)

::method "test_82"
        arr=bsf.createArray("short.class", 5)
        r=BSF("arrayPut", arr, 12345, 2)
        self~assertEquals(arr[3], 12345)

::method "test_83"
        show_msg = BsfShowErrorMessage(0)
        arr=bsf.createArray("short.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 12345, 7)

::method "test_84"
        arr=bsf.createArray("short.class", 5, 5)
        r=BSF("arrayPut", arr, 12345, 2, 2)
        self~assertEquals(arr[3,3], 12345)

::method "test_85"
        arr=bsf.createArray("short.class", 5, 5, 5)
        r=BSF("arrayPut", arr, 12345, 2, 2, 2)
        self~assertEquals(arr[3,3,3], 12345)

::method "test_86"
        arr=bsf.createArray("short.class", 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 12345, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3], 12345)

::method "test_87"
        arr=bsf.createArray("short.class", 5, 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 12345, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3], 12345)

::method "test_88"
        arr=bsf.createArray("short.class", 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 12345, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3], 12345)

::method "test_89"
        arr=bsf.createArray("short.class", 5, 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 12345, 2, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3,3], 12345)

::method "test_90"
        arr=bsf.createArray("short.class", 128623)
        r=BSF("arrayPut", arr, 12345, 54321)
        self~assertEquals(arr[54322], 12345)

::method "test_91"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("short.class", foo)

::method "test_92"
        show_msg = BsfShowErrorMessage(0)
```

```
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("short.class", -1)

::method "test_93"
        arr=bsf.createArray("short.class", 0)
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 12345, 0)

::method "test_94"
        arr=bsf.createArray("short.class", 5)
        r=BSF("arrayPut", arr, -12345, 2)
        self~assertEquals(arr[3], -12345)

::method "test_95"
        arr=bsf.createArray("short.class", 5)
        r=BSF("arrayPut", arr, 32767, 2)
        self~assertEquals(arr[3], 32767)

::method "test_96"
        arr=bsf.createArray("short.class", 5)
        r=BSF("arrayPut", arr, -32767, 2)
        self~assertEquals(arr[3], -32767)

::method "test_97"
        arr=bsf.createArray("short.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 32768, 2)

::method "test_98"
        arr=bsf.createArray("short.class", 5)
        r=BSF("arrayPut", arr, -32768, 2)
        self~assertEquals(arr[3], -32768)

::method "test_99"
        arr=bsf.createArray("short.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 32769, 2)

::method "test_100"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("short.class", 5)
        r=BSF("arrayPut", arr, "foo", 2)

::method "test_101"
        arr=bsf.createArray("short.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 12345, -1)

::method "test_102"
        arr=bsf.createArray("long.class", 5)
        r=BSF("arrayPut", arr, 12345, 2)
```

```
        self~assertEquals(arr[3], 12345)

::method "test_103"
        show_msg = BsfShowErrorMessage(0)
        arr=bsf.createArray("long.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 12345, 7)

::method "test_104"
        arr=bsf.createArray("long.class", 5, 5)
        r=BSF("arrayPut", arr, 12345, 2, 2)
        self~assertEquals(arr[3,3], 12345)

::method "test_105"
        arr=bsf.createArray("long.class", 5, 5, 5)
        r=BSF("arrayPut", arr, 12345, 2, 2, 2)
        self~assertEquals(arr[3,3,3], 12345)

::method "test_106"
        arr=bsf.createArray("long.class", 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 12345, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3], 12345)

::method "test_107"
        arr=bsf.createArray("long.class", 5, 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 12345, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3], 12345)

::method "test_108"
        arr=bsf.createArray("long.class", 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 12345, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3], 12345)

::method "test_109"
        arr=bsf.createArray("long.class", 5, 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 12345, 2, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3,3], 12345)

::method "test_110"
        arr=bsf.createArray("long.class", 128623)
        r=BSF("arrayPut", arr, 12345, 54321)
        self~assertEquals(arr[54322], 12345)

::method "test_111"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("long.class", foo)

::method "test_112"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("long.class", -1)

::method "test_113"
        arr=bsf.createArray("long.class", 0)
        show_msg = BsfShowErrorMessage(0)
```

```
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 12345, 0)

::method "test_114"
        arr=bsf.createArray("long.class", 5)
        r=BSF("arrayPut", arr, -12345, 2)
        self~assertEquals(arr[3], -12345)

::method "test_115"
        arr=bsf.createArray("long.class", 5)
        r=BSF("arrayPut", arr, 9223372036854775807, 2)
        self~assertEquals(arr[3], 9223372036854775807)

::method "test_116"
        arr=bsf.createArray("long.class", 5)
        r=BSF("arrayPut", arr, '-9223372036854775808', 2)
        self~assertEquals(arr[3], '-9223372036854775808')

::method "test_117"
        arr=bsf.createArray("long.class", 5)
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 9223372036854775808, 2)

::method "test_118"
        arr=bsf.createArray("long.class", 5)
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, '-9223372036854775809', 2)

::method "test_119"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("long.class", 5)
        r=BSF("arrayPut", arr, "foo", 2)

::method "test_120"
        arr=bsf.createArray("long.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 12345, -1)

::method "test_121"
        arr=bsf.createArray("float.class", 5)
        r=BSF("arrayPut", arr, 12345.123, 2)
        self~assertEquals(arr[3], 12345.123)

::method "test_122"
        show_msg = BsfShowErrorMessage(0)
        arr=bsf.createArray("float.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 12345.123, 7)

::method "test_123"
```

```
        arr=bsf.createArray("float.class", 5, 5)
        r=BSF("arrayPut", arr, 12345.123, 2, 2)
        self~assertEquals(arr[3,3], 12345.123)

::method "test_124"
        arr=bsf.createArray("float.class", 5, 5, 5)
        r=BSF("arrayPut", arr, 12345.123, 2, 2, 2)
        self~assertEquals(arr[3,3,3], 12345.123)

::method "test_125"
        arr=bsf.createArray("float.class", 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 12345.123, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3], 12345.123)

::method "test_126"
        arr=bsf.createArray("float.class", 5, 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 12345.123, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3], 12345.123)

::method "test_127"
        arr=bsf.createArray("float.class", 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 12345.123, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3], 12345.123)

::method "test_128"
        arr=bsf.createArray("float.class", 5, 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 12345.123, 2, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3,3], 12345.123)

::method "test_129"
        arr=bsf.createArray("float.class", 128623)
        r=BSF("arrayPut", arr, 12345.123, 54321)
        self~assertEquals(arr[54322], 12345.123)

::method "test_130"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("float.class", foo)

::method "test_131"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("float.class", -1)

::method "test_132"
        arr=bsf.createArray("float.class", 0)
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 12345.123, 0)

::method "test_133"
        arr=bsf.createArray("float.class", 5)
        r=BSF("arrayPut", arr, '-12345.123', 2)
        self~assertEquals(arr[3], '-12345.123')

::method "test_134"
```

```
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("float.class", 5)
        r=BSF("arrayPut", arr, "foo", 2)

::method "test_135"
        arr=bsf.createArray("float.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 12345, -1)

::method "test_136"
        arr=bsf.createArray("float.class", 5)
        r=BSF("arrayPut", arr, 3.14E5, 2)
        self~assertEquals(arr[3], 314000.0)

::method "test_137"
        arr=bsf.createArray("float.class", 5)
        r=BSF("arrayPut", arr, 3.14E5, 2)
        self~assertEquals(arr[3], 3.14E5)

::method "test_138"
        arr=bsf.createArray("float.class", 5)
        r=BSF("arrayPut", arr, 3.14E5F, 2)
        self~assertEquals(arr[3], 3.14E5)

::method "test_139"
        arr=bsf.createArray("float.class", 5)
        r=BSF("arrayPut", arr, 314000.0, 2)
        self~assertEquals(arr[3], 3.14E5)

::method "test_140"
        arr=bsf.createArray("double.class", 5)
        r=BSF("arrayPut", arr, 12345.123, 2)
        self~assertEquals(arr[3], 12345.123)

::method "test_141"
        show_msg = BsfShowErrorMessage(0)
        arr=bsf.createArray("double.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 12345.123, 7)

::method "test_142"
        arr=bsf.createArray("double.class", 5, 5)
        r=BSF("arrayPut", arr, 12345.123, 2, 2)
        self~assertEquals(arr[3,3], 12345.123)

::method "test_143"
        arr=bsf.createArray("double.class", 5, 5, 5)
        r=BSF("arrayPut", arr, 12345.123, 2, 2, 2)
        self~assertEquals(arr[3,3,3], 12345.123)

::method "test_144"
        arr=bsf.createArray("double.class", 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 12345.123, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3], 12345.123)
```

```
::method "test_145"
        arr=bsf.createArray("double.class", 5, 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 12345.123, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3], 12345.123)

::method "test_146"
        arr=bsf.createArray("double.class", 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 12345.123, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3], 12345.123)

::method "test_147"
        arr=bsf.createArray("double.class", 5, 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPut", arr, 12345.123, 2, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3,3], 12345.123)

::method "test_148"
        arr=bsf.createArray("double.class", 128623)
        r=BSF("arrayPut", arr, 12345.123, 54321)
        self~assertEquals(arr[54322], 12345.123)

::method "test_149"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("double.class", foo)

::method "test_150"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("double.class", -1)

::method "test_151"
        arr=bsf.createArray("double.class", 0)
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 12345.123, 0)

::method "test_152"
        arr=bsf.createArray("double.class", 5)
        r=BSF("arrayPut", arr, -12345.123, 2)
        self~assertEquals(arr[3], -12345.123)

::method "test_153"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("double.class", 5)
        r=BSF("arrayPut", arr, "foo", 2)

::method "test_154"
        arr=bsf.createArray("double.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPut", arr, 12345, -1)

::method "test_155"
        arr=bsf.createArray("double.class", 5)
```

```
        r=BSF("arrayPut", arr, 3.14E5, 2)
        self~assertEquals(arr[3], 314000.0)

::method "test_156"
        arr=bsf.createArray("double.class", 5)
        r=BSF("arrayPut", arr, 3.14E5, 2)
        self~assertEquals(arr[3], 3.14E5)

::method "test_157"
        arr=bsf.createArray("double.class", 5)
        r=BSF("arrayPut", arr, 3.14E5F, 2)
        self~assertEquals(arr[3], 3.14E5)

::method "test_158"
        arr=bsf.createArray("double.class", 5)
        r=BSF("arrayPut", arr, 314000.0, 2)
        self~assertEquals(arr[3], 3.14E5)

::method "test_159"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("", 5)

::method "test_160"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray(, 5)

::method "test_161"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray(5)

::method "test_162"
        arr=bsf.createArray("int.class", 5)
        r=BSF("arrayPutStrict", arr, "int.class", 12345, 2)
        self~assertEquals(arr[3], 12345)

::method "test_163"
        show_msg = BsfShowErrorMessage(0)
        arr=bsf.createArray("int.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "int.class", 12345, 7)

::method "test_164"
        arr=bsf.createArray("int.class", 5, 5)
        r=BSF("arrayPutStrict", arr, "int.class", 12345, 2, 2)
        self~assertEquals(arr[3,3], 12345)

::method "test_165"
        arr=bsf.createArray("int.class", 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "int.class", 12345, 2, 2, 2)
        self~assertEquals(arr[3,3,3], 12345)

::method "test_166"
```

```
        arr=bsf.createArray("int.class", 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "int.class", 12345, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3], 12345)

::method "test_167"
        arr=bsf.createArray("int.class", 5, 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "int.class", 12345, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3], 12345)

::method "test_168"
        arr=bsf.createArray("int.class", 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "int.class", 12345, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3], 12345)

::method "test_169"
        arr=bsf.createArray("int.class", 5, 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "int.class", 12345, 2, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3,3], 12345)

::method "test_170"
        arr=bsf.createArray("int.class", 128623)
        r=BSF("arrayPutStrict", arr, "int.class", 12345, 54321)
        self~assertEquals(arr[54322], 12345)

::method "test_171"
        arr=bsf.createArray("int.class", 0)
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "int.class", 12345, 0)

::method "test_172"
        arr=bsf.createArray("int.class", 5)
        r=BSF("arrayPutStrict", arr, "int.class", -12345, 2)
        self~assertEquals(arr[3], -12345)

::method "test_173"
        arr=bsf.createArray("int.class", 5)
        r=BSF("arrayPutStrict", arr, "int.class", 2147483647, 2)
        self~assertEquals(arr[3], 2147483647)

::method "test_174"
        arr=bsf.createArray("int.class", 5)
        r=BSF("arrayPutStrict", arr, "int.class", '-2147483648', 2)
        self~assertEquals(arr[3], '-2147483648')

::method "test_175"
        arr=bsf.createArray("int.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "int.class", '2147483648', 2)

::method "test_176"
        arr=bsf.createArray("int.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "int.class", '-2147483649', 2)

::method "test_177"
```

```
        show_msg = BsfShowErrorMessage(0)
        arr=bsf.createArray("int.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "int.class", foo, 2)

::method "test_178"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("int.class", 5)
        r=BSF("arrayPutStrict", arr, "int.class", "foo", 2)

::method "test_179"
        arr=bsf.createArray("int.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "int.class", 12345, -1)

::method "test_180"
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", foo, "int.class", 12345, 2)

::method "test_181"
        arr=bsf.createArray("int.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "int.class", 12345)

::method "test_182"
        arr=bsf.createArray("int.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr)

::method "test_183"
        arr=bsf.createArray("int.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, , )

::method "test_184"
        arr=bsf.createArray("String.class", 128623)
        r=BSF("arrayPutStrict", arr, "String.class", 12345, 54321)
        self~assertEquals(arr[54322], 12345)

::method "test_185"
        arr=bsf.createArray("String.class", 0)
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "String.class", "foo", 0)

::method "test_186"
        arr=bsf.createArray("String.class", 5)
        r=BSF("arrayPutStrict", arr, "String.class", -12345, 2)
        self~assertEquals(arr[3], -12345)
```

```
::method "test_187"
        arr=bsf.createArray("String.class", 5)
        r=BSF("arrayPutStrict", arr, "String.class", "foo", 2)
        self~assertEquals(arr[3], "foo")

::method "test_188"
        arr=bsf.createArray("String.class", 5)
        r=BSF("arrayPutStrict", arr, "String.class", "foo", 2)
        self~assertNotEquals(arr[3], "bar")

::method "test_189"
        arr=bsf.createArray("String.class", 5)
        r=BSF("arrayPutStrict", arr, "String.class", '"', 2)
        self~assertEquals(arr[3], '"')

::method "test_190"
        arr=bsf.createArray("String.class", 5)
        r=BSF("arrayPutStrict", arr, "String.class", "'", 2)
        self~assertEquals(arr[3], "'")

::method "test_191"
        arr=bsf.createArray("String.class", 5)
        r=BSF("arrayPutStrict", arr, "String.class", "AAAaaa", 2)
        self~assertEquals(arr[3], "AAAaaa")

::method "test_192"
        arr=bsf.createArray("boolean.class", 0)
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "boolean.class", 1, 0)

::method "test_193"
        arr=bsf.createArray("boolean.class", 5)
        r=BSF("arrayPutStrict", arr, "boolean.class", .false, 2)
        self~assertEquals(arr[3], .false)

::method "test_194"
        arr=bsf.createArray("boolean.class", 5)
        r=BSF("arrayPutStrict", arr, "boolean.class", 0, 2)
        self~assertEquals(arr[3], 0)

::method "test_195"
        arr=bsf.createArray("boolean.class", 5)
        r=BSF("arrayPutStrict", arr, "boolean.class", 2, 2)
        self~assertEquals(arr[3], .false)

::method "test_196"
        show_msg = BsfShowErrorMessage(0)
        arr=bsf.createArray("boolean.class", 5)
        r=BSF("arrayPutStrict", arr, "boolean.class", "foo", 2)
        self~assertEquals(arr[3], .false)

::method "test_197"
        arr=bsf.createArray("boolean.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "boolean.class", 1, -1)
```

```
::method "test_198"
        arr=bsf.createArray("byte.class", 5)
        r=BSF("arrayPutStrict", arr, "byte.class", 123, 2)
        self~assertEquals(arr[3], 123)

::method "test_199"
        show_msg = BsfShowErrorMessage(0)
        arr=bsf.createArray("byte.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "byte.class", 123, 7)

::method "test_200"
        arr=bsf.createArray("byte.class", 5, 5)
        r=BSF("arrayPutStrict", arr, "byte.class", 123, 2, 2)
        self~assertEquals(arr[3,3], 123)

::method "test_201"
        arr=bsf.createArray("byte.class", 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "byte.class", 123, 2, 2, 2)
        self~assertEquals(arr[3,3,3], 123)

::method "test_202"
        arr=bsf.createArray("byte.class", 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "byte.class", 123, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3], 123)

::method "test_203"
        arr=bsf.createArray("byte.class", 5, 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "byte.class", 123, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3], 123)

::method "test_204"
        arr=bsf.createArray("byte.class", 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "byte.class", 123, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3], 123)

::method "test_205"
        arr=bsf.createArray("byte.class", 5, 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "byte.class", 123, 2, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3,3], 123)

::method "test_206"
        arr=bsf.createArray("byte.class", 128623)
        r=BSF("arrayPutStrict", arr, "byte.class", 123, 54321)
        self~assertEquals(arr[54322], 123)

::method "test_207"
        arr=bsf.createArray("byte.class", 0)
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "byte.class", 123, 0)

::method "test_208"
        arr=bsf.createArray("byte.class", 5)
        r=BSF("arrayPutStrict", arr, "byte.class", -123, 2)
        self~assertEquals(arr[3], -123)
```

```
::method "test_209"
        arr=bsf.createArray("byte.class", 5)
        r=BSF("arrayPutStrict", arr, "byte.class", 127, 2)
        self~assertEquals(arr[3], 127)

::method "test_210"
        arr=bsf.createArray("byte.class", 5)
        r=BSF("arrayPutStrict", arr, "byte.class", -128, 2)
        self~assertEquals(arr[3], -128)

::method "test_211"
        arr=bsf.createArray("byte.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "byte.class", 128, 2)

::method "test_212"
        arr=bsf.createArray("byte.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "byte.class", -129, 2)

::method "test_213"
        arr=bsf.createArray("byte.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "byte.class", 255, 2)

::method "test_214"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("byte.class", 5)
        r=BSF("arrayPutStrict", arr, "byte.class", "foo", 2)

::method "test_215"
        arr=bsf.createArray("byte.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "byte.class", 123, -1)

::method "test_216"
        arr=bsf.createArray("char.class", 5)
        r=BSF("arrayPutStrict", arr, "char.class", 'a', 2)
        self~assertEquals(arr[3], 'a')

::method "test_217"
        show_msg = BsfShowErrorMessage(0)
        arr=bsf.createArray("char.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "char.class", 'a', 7)

::method "test_218"
        arr=bsf.createArray("char.class", 5, 5)
        r=BSF("arrayPutStrict", arr, "char.class", 'a', 2, 2)
        self~assertEquals(arr[3,3], 'a')

::method "test_219"
```

```
        arr=bsf.createArray("char.class", 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "char.class", 'a', 2, 2, 2)
        self~assertEquals(arr[3,3,3], 'a')

::method "test_220"
        arr=bsf.createArray("char.class", 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "char.class", 'a', 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3], 'a')

::method "test_221"
        arr=bsf.createArray("char.class", 5, 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "char.class", 'a', 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3], 'a')

::method "test_222"
        arr=bsf.createArray("char.class", 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "char.class", 'a', 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3], 'a')

::method "test_223"
        arr=bsf.createArray("char.class", 5, 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "char.class", 'a', 2, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3,3], 'a')

::method "test_224"
        arr=bsf.createArray("char.class", 128623)
        r=BSF("arrayPutStrict", arr, "char.class", 'a', 54321)
        self~assertEquals(arr[54322], 'a')

::method "test_225"
        arr=bsf.createArray("char.class", 0)
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "char.class", 'a', 0)

::method "test_226"
        arr=bsf.createArray("char.class", 5)
        r=BSF("arrayPutStrict", arr, "char.class", '1', 2)
        self~assertEquals(arr[3], '1')

::method "test_227"
        arr=bsf.createArray("char.class", 5)
        r=BSF("arrayPutStrict", arr, "char.class", '"', 2)
        self~assertEquals(arr[3], '"')

::method "test_228"
        show_msg = BsfShowErrorMessage(0)
        arr=bsf.createArray("char.class", 5)
        r=BSF("arrayPutStrict", arr, "char.class", "foo", 2)
        self~assertEquals(arr[3], 'f')

::method "test_229"
        arr=bsf.createArray("char.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "char.class", 'a', -1)

::method "test_230"
```

```
        arr=bsf.createArray("short.class", 5)
        r=BSF("arrayPutStrict", arr, "short.class", 12345, 2)
        self~assertEquals(arr[3], 12345)

::method "test_231"
        show_msg = BsfShowErrorMessage(0)
        arr=bsf.createArray("short.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "short.class", 12345, 7)

::method "test_232"
        arr=bsf.createArray("short.class", 5, 5)
        r=BSF("arrayPutStrict", arr, "short.class", 12345, 2, 2)
        self~assertEquals(arr[3,3], 12345)

::method "test_233"
        arr=bsf.createArray("short.class", 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "short.class", 12345, 2, 2, 2)
        self~assertEquals(arr[3,3,3], 12345)

::method "test_234"
        arr=bsf.createArray("short.class", 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "short.class", 12345, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3], 12345)

::method "test_235"
        arr=bsf.createArray("short.class", 5, 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "short.class", 12345, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3], 12345)

::method "test_236"
        arr=bsf.createArray("short.class", 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "short.class", 12345, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3], 12345)

::method "test_237"
        arr=bsf.createArray("short.class", 5, 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "short.class", 12345, 2, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3,3], 12345)

::method "test_238"
        arr=bsf.createArray("short.class", 128623)
        r=BSF("arrayPutStrict", arr, "short.class", 12345, 54321)
        self~assertEquals(arr[54322], 12345)

::method "test_239"
        arr=bsf.createArray("short.class", 0)
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "short.class", 12345, 0)

::method "test_240"
        arr=bsf.createArray("short.class", 5)
        r=BSF("arrayPutStrict", arr, "short.class", -12345, 2)
        self~assertEquals(arr[3], -12345)

::method "test_241"
```

```
        arr=bsf.createArray("short.class", 5)
        r=BSF("arrayPutStrict", arr, "short.class", 32767, 2)
        self~assertEquals(arr[3], 32767)

::method "test_242"
        arr=bsf.createArray("short.class", 5)
        r=BSF("arrayPutStrict", arr, "short.class", -32767, 2)
        self~assertEquals(arr[3], -32767)

::method "test_243"
        arr=bsf.createArray("short.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "short.class", 32768, 2)

::method "test_244"
        arr=bsf.createArray("short.class", 5)
        r=BSF("arrayPutStrict", arr, "short.class", -32768, 2)
        self~assertEquals(arr[3], -32768)

::method "test_245"
        arr=bsf.createArray("short.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "short.class", 32769, 2)

::method "test_246"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("short.class", 5)
        r=BSF("arrayPutStrict", arr, "short.class", "foo", 2)

::method "test_247"
        arr=bsf.createArray("short.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "short.class", 12345, -1)

::method "test_248"
        arr=bsf.createArray("long.class", 5)
        r=BSF("arrayPutStrict", arr, "long.class", 12345, 2)
        self~assertEquals(arr[3], 12345)

::method "test_249"
        show_msg = BsfShowErrorMessage(0)
        arr=bsf.createArray("long.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "long.class", 12345, 7)

::method "test_250"
        arr=bsf.createArray("long.class", 5, 5)
        r=BSF("arrayPutStrict", arr, "long.class", 12345, 2, 2)
        self~assertEquals(arr[3,3], 12345)

::method "test_251"
        arr=bsf.createArray("long.class", 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "long.class", 12345, 2, 2, 2)
```

```
        self~assertEquals(arr[3,3,3], 12345)

::method "test_252"
        arr=bsf.createArray("long.class", 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "long.class", 12345, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3], 12345)

::method "test_253"
        arr=bsf.createArray("long.class", 5, 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "long.class", 12345, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3], 12345)

::method "test_254"
        arr=bsf.createArray("long.class", 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "long.class", 12345, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3], 12345)

::method "test_255"
        arr=bsf.createArray("long.class", 5, 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "long.class", 12345, 2, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3,3], 12345)

::method "test_256"
        arr=bsf.createArray("long.class", 128623)
        r=BSF("arrayPutStrict", arr, "long.class", 12345, 54321)
        self~assertEquals(arr[54322], 12345)

::method "test_257"
        arr=bsf.createArray("long.class", 0)
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "long.class", 12345, 0)

::method "test_258"
        arr=bsf.createArray("long.class", 5)
        r=BSF("arrayPutStrict", arr, "long.class", -12345, 2)
        self~assertEquals(arr[3], -12345)

::method "test_259"
        arr=bsf.createArray("long.class", 5)
        r=BSF("arrayPutStrict", arr, "long.class", 9223372036854775807, 2)
        self~assertEquals(arr[3], 9223372036854775807)

::method "test_260"
        arr=bsf.createArray("long.class", 5)
        r=BSF("arrayPutStrict", arr, "long.class", '-9223372036854775808', 2)
        self~assertEquals(arr[3], '-9223372036854775808')

::method "test_261"
        arr=bsf.createArray("long.class", 5)
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "long.class", 9223372036854775808, 2)

::method "test_262"
        arr=bsf.createArray("long.class", 5)
        show_msg = BsfShowErrorMessage(0)
```

```
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "long.class", '-9223372036854775809', 2)

::method "test_263"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("long.class", 5)
        r=BSF("arrayPutStrict", arr, "long.class", "foo", 2)

::method "test_264"
        arr=bsf.createArray("long.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "long.class", 12345, -1)

::method "test_265"
        arr=bsf.createArray("float.class", 5)
        r=BSF("arrayPutStrict", arr, "float.class", 12345.123, 2)
        self~assertEquals(arr[3], 12345.123)

::method "test_266"
        show_msg = BsfShowErrorMessage(0)
        arr=bsf.createArray("float.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "float.class", 12345.123, 7)

::method "test_267"
        arr=bsf.createArray("float.class", 5, 5)
        r=BSF("arrayPutStrict", arr, "float.class", 12345.123, 2, 2)
        self~assertEquals(arr[3,3], 12345.123)

::method "test_268"
        arr=bsf.createArray("float.class", 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "float.class", 12345.123, 2, 2, 2)
        self~assertEquals(arr[3,3,3], 12345.123)

::method "test_269"
        arr=bsf.createArray("float.class", 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "float.class", 12345.123, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3], 12345.123)

::method "test_270"
        arr=bsf.createArray("float.class", 5, 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "float.class", 12345.123, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3], 12345.123)

::method "test_271"
        arr=bsf.createArray("float.class", 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "float.class", 12345.123, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3], 12345.123)

::method "test_272"
        arr=bsf.createArray("float.class", 5, 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "float.class", 12345.123, 2, 2, 2, 2, 2, 2, 2
)
        self~assertEquals(arr[3,3,3,3,3,3,3], 12345.123)
```

```
::method "test_273"
        arr=bsf.createArray("float.class", 128623)
        r=BSF("arrayPutStrict", arr, "float.class", 12345.123, 54321)
        self~assertEquals(arr[54322], 12345.123)

::method "test_274"
        arr=bsf.createArray("float.class", 0)
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "float.class", 12345.123, 0)

::method "test_275"
        arr=bsf.createArray("float.class", 5)
        r=BSF("arrayPutStrict", arr, "float.class", '-12345.123', 2)
        self~assertEquals(arr[3], '-12345.123')

::method "test_276"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("float.class", 5)
        r=BSF("arrayPutStrict", arr, "float.class", "foo", 2)

::method "test_277"
        arr=bsf.createArray("float.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "float.class", 12345, -1)

::method "test_278"
        arr=bsf.createArray("float.class", 5)
        r=BSF("arrayPutStrict", arr, "float.class", 3.14E5, 2)
        self~assertEquals(arr[3], 314000.0)

::method "test_279"
        arr=bsf.createArray("float.class", 5)
        r=BSF("arrayPutStrict", arr, "float.class", 3.14E5, 2)
        self~assertEquals(arr[3], 3.14E5)

::method "test_280"
        arr=bsf.createArray("float.class", 5)
        r=BSF("arrayPutStrict", arr, "float.class", 3.14E5F, 2)
        self~assertEquals(arr[3], 3.14E5)

::method "test_281"
        arr=bsf.createArray("float.class", 5)
        r=BSF("arrayPutStrict", arr, "float.class", 314000.0, 2)
        self~assertEquals(arr[3], 3.14E5)

::method "test_282"
        arr=bsf.createArray("double.class", 5)
        r=BSF("arrayPutStrict", arr, "double.class", 12345.123, 2)
        self~assertEquals(arr[3], 12345.123)

::method "test_283"
        show_msg = BsfShowErrorMessage(0)
        arr=bsf.createArray("double.class", 5)
```

```
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "double.class", 12345.123, 7)

::method "test_284"
        arr=bsf.createArray("double.class", 5, 5)
        r=BSF("arrayPutStrict", arr, "double.class", 12345.123, 2, 2)
        self~assertEquals(arr[3,3], 12345.123)

::method "test_285"
        arr=bsf.createArray("double.class", 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "double.class", 12345.123, 2, 2, 2)
        self~assertEquals(arr[3,3,3], 12345.123)

::method "test_286"
        arr=bsf.createArray("double.class", 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "double.class", 12345.123, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3], 12345.123)

::method "test_287"
        arr=bsf.createArray("double.class", 5, 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "double.class", 12345.123, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3], 12345.123)

::method "test_288"
        arr=bsf.createArray("double.class", 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "double.class", 12345.123, 2, 2, 2, 2, 2, 2)
        self~assertEquals(arr[3,3,3,3,3,3], 12345.123)

::method "test_289"
        arr=bsf.createArray("double.class", 5, 5, 5, 5, 5, 5, 5)
        r=BSF("arrayPutStrict", arr, "double.class", 12345.123, 2, 2, 2, 2, 2, 2,
2)
        self~assertEquals(arr[3,3,3,3,3,3,3], 12345.123)

::method "test_290"
        arr=bsf.createArray("double.class", 128623)
        r=BSF("arrayPutStrict", arr, "double.class", 12345.123, 54321)
        self~assertEquals(arr[54322], 12345.123)

::method "test_291"
        arr=bsf.createArray("double.class", 0)
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "double.class", 12345.123, 0)

::method "test_292"
        arr=bsf.createArray("double.class", 5)
        r=BSF("arrayPutStrict", arr, "double.class", -12345.123, 2)
        self~assertEquals(arr[3], -12345.123)

::method "test_293"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("double.class", 5)
        r=BSF("arrayPutStrict", arr, "double.class", "foo", 2)
```

```
::method "test_294"
        arr=bsf.createArray("double.class", 5)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "double.class", 12345, -1)

::method "test_295"
        arr=bsf.createArray("double.class", 5)
        r=BSF("arrayPutStrict", arr, "double.class", 3.14E5, 2)
        self~assertEquals(arr[3], 314000.0)

::method "test_296"
        arr=bsf.createArray("double.class", 5)
        r=BSF("arrayPutStrict", arr, "double.class", 3.14E5, 2)
        self~assertEquals(arr[3], 3.14E5)

::method "test_297"
        arr=bsf.createArray("double.class", 5)
        r=BSF("arrayPutStrict", arr, "double.class", 3.14E5F, 2)
        self~assertEquals(arr[3], 3.14E5)

::method "test_298"
        arr=bsf.createArray("double.class", 5)
        r=BSF("arrayPutStrict", arr, "double.class", 314000.0, 2)
        self~assertEquals(arr[3], 3.14E5)

::method "test_299"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("int", 5)

::method "test_300"
        arr=bsf.createArray(.bsf4rexx~int, 5)
        r=BSF("arrayPut", arr, 12345, 2)
        self~assertEquals(arr[3], 12345)

::method "test_301"
        arr=bsf.createArray(.bsf4rexx~byte, 5)
        r=BSF("arrayPut", arr, 123, 2)
        self~assertEquals(arr[3], 123)

::method "test_302"
        arr=bsf.createArray(.bsf4rexx~char, 5)
        r=BSF("arrayPut", arr, 'a', 2)
        self~assertEquals(arr[3], 'a')

::method "test_303"
        arr=bsf.createArray(.bsf4rexx~double, 5)
        r=BSF("arrayPut", arr, 12345, 2)
        self~assertEquals(arr[3], 12345)

::method "test_304"
        arr=bsf.createArray(.bsf4rexx~float, 5)
        r=BSF("arrayPut", arr, 12345, 2)
        self~assertEquals(arr[3], 12345)

::method "test_305"
        arr=bsf.createArray(.bsf4rexx~long, 5)
```

```
        r=BSF("arrayPut", arr, 12345, 2)
        self~assertEquals(arr[3], 12345)

::method "test_306"
        arr=bsf.createArray(.bsf4rexx~short, 5)
        r=BSF("arrayPut", arr, 123, 2)
        self~assertEquals(arr[3], 123)

::method "test_307"
        arr=bsf.createArray(.bsf4rexx~boolean, 5)
        r=BSF("arrayPut", arr, .false, 2)
        self~assertEquals(arr[3], .false)

::method "test_308"
        arr=bsf.createArray("int.class", 5)
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("arrayPutStrict", arr, "float.class", 12345, 2)

::method "test_309"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("byte", 5)

::method "test_310"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("char", 5)

::method "test_311"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("double", 5)

::method "test_312"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("float", 5)

::method "test_313"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("long", 5)

::method "test_314"
        show_msg = BsfShowErrorMessage(0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        arr=bsf.createArray("short", 5)

return
```

Bean.testGroup

```
#!/usr/bin/rexx
parse source . . fileSpec;

  group = .TestGroup~new(fileSpec)
  group~add(.Bean.testGroup)

  if group~isAutomatedTest then return group

  testResult = group~suite~execute~~print

return testResult
-- End of entry point.

::requires "ooTest.frm"      -- load the ooRexxUnit classes
::requires "BSF.cls"

::class "Bean.testGroup" subclass ooTestCase public

::method "test_1"
        .local~ver = 2

        show_msg = BsfShowErrorMessage(0)
        .bsf~bsf.import("java.lang.String","JString")
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        s=.JString~newStrict("int", 5)

::method "test_2"
        .bsf~bsf.import("java.awt.TextField", "Jatf")
        ftf=.Jatf~newStrict("int", 5)
        self~assertEquals(ftf~getColumns(),5)

::method "test_3"
        show_msg = BsfShowErrorMessage(0)
        .bsf~bsf.import("java.awt.TextField", "Jatf")
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        ftf=.Jatf~newStrict("int", 1, 2)

::method "test_4"
        show_msg = BsfShowErrorMessage(0)
        .bsf~bsf.import("java.awt.TextField", "Jatf")
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        ftf=.Jatf~newStrict("double", 5)

::method "test_5"
        .bsf~bsf.import("java.awt.TextField", "Jatf")
        r=BSF("setRexxNullString", "newnullstring")
        ftf=.Jatf~newStrict("int", 5)
        var=bsf.unregisterBean("Jatf")
        self~assertEquals(var, "newnullstring")

::method "test_6"
        .bsf~bsf.import("java.awt.TextField", "Jatf")
        r=BSF("setRexxNullString", "newnullstring")
```

```
        ftf=.Jatf~newStrict("int", 5)
        var=bsf.unregisterBean("a_not_existing_bean")
        self~assertEquals(var, "newnullstring")

::method "test_7"
        show_msg = BsfShowErrorMessage(0)
        .bsf~bsf.import("java.awt.TextField", "Jatf")
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        ftf=.Jatf~newStrict("not_existing", 5)

::method "test_8"
        r=BSF("bsfPrefixReturnValue", 0)
        r=BSF("setRexxNullString", "newnullstring")
        b=BSF("lookupBean", "not_there")
        self~assertEquals(b, "newnullstring")
        r=BSF("bsfPrefixReturnValue", 1)

::method "test_9"
        show_msg = BsfShowErrorMessage(0)
        .bsf~bsf.import("java.lang.String","JString")
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        s=.JString~new("foo", 5)

::method "test_10"
        .bsf~bsf.import("java.awt.TextField", "Jatf")
        ftf=.Jatf~new("foo", 5)
        self~assertEquals(ftf~getColumns(),5)

::method "test_11"
        show_msg = BsfShowErrorMessage(0)
        .bsf~bsf.import("java.awt.TextField", "Jatf")
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        ftf=.Jatf~new("foo", 1, 2)

::method "test_12"
        .bsf~bsf.import("java.awt.TextField", "Jatf")
        r=BSF("setRexxNullString", "newnullstring")
        ftf=.Jatf~new("foo", 5)
        var=bsf.unregisterBean("Jatf")
        self~assertEquals(var, "newnullstring")

::method "test_13"
        .bsf~bsf.import("java.awt.TextField", "Jatf")
        r=BSF("setRexxNullString", "newnullstring")
        ftf=.Jatf~new("foo", 5)
        var=bsf.unregisterBean("a_not_existing_bean")
        self~assertEquals(var, "newnullstring")


::method "test_14"
        show_msg = BsfShowErrorMessage(0)
        .bsf~bsf.import("java.lang.String","JString")
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        s=.JString~registerBeanStrict("int", 5)
```

```
::method "test_27"
        cls=BSF("loadclass", "java.awt.TextField")
        ftf=BSF("rawRegisterBean", "jatf", cls)
        self~assertNotNull(ftf)
return
```

BsfInvokedBy.testGroup

```
#!/usr/bin/rexx
  parse source . . fileSpec;

  group = .TestGroup~new(fileSpec)
  group~add(.BsfInvokedBy.testGroup)

  if group~isAutomatedTest then return group

  testResult = group~suite~execute~~print

return testResult
-- End of entry point.

::requires "ooTest.frm"     -- load the ooRexxUnit classes
::requires "BSF.cls"

::class "BsfInvokedBy.testGroup" subclass ooTestCase public

::method "test_1"
        .local~ver = 2

        r=BsfInvokedBy()
        self~assertEquals(r, 2)

::method "test_2"
        IF .ver = 4 THEN errorCode=88.922; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BsfInvokedBy("foo")

return
```

bsfPrefixReturnValue.testGroup

```
#!/usr/bin/rexx
  parse source . . fileSpec;

  group = .TestGroup~new(fileSpec)
  group~add(.bsfPrefixReturnValue.testGroup)

  if group~isAutomatedTest then return group

  testResult = group~suite~execute~~print

return testResult
-- End of entry point.
```

```
::requires "ooTest.frm"     -- load the ooRexxUnit classes
::requires "BSF.cls"

::class "bsfPrefixReturnValue.testGroup" subclass ooTestCase public

::method "test_1"
        .local~ver = 2

        pre=BSF("bsfPrefixReturnValue", 0)
        self~assertEquals(pre, 0)

::method "test_2"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("Array.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)

::method "test_3"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("Class.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)

::method "test_4"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("Method.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)

::method "test_5"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("Object.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)

::method "test_6"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("String.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)

::method "test_7"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("System.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)

::method "test_8"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("Thread.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)
```

```
::method "test_9"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("boolean.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)

::method "test_10"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("Boolean.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)

::method "test_11"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("byte.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)

::method "test_12"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("Byte.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)

::method "test_13"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("char.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)

::method "test_14"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("Character.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)

::method "test_15"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("double.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)

::method "test_16"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("Double.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)

::method "test_17"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("float.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
```

```
        self~assertEquals(len, 5)

::method "test_18"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("Float.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)

::method "test_19"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("int.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)

::method "test_20"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("Integer.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)

::method "test_21"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("long.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)

::method "test_22"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("Long.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)

::method "test_23"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("short.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)

::method "test_24"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("Short.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)

::method "test_25"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("Void.class", 5)
        pre=BSF("bsfPrefixReturnValue", 0)
        len=BSF("arrayLength", arr)
        self~assertEquals(len, 5)

::method "test_26"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("int.class", 5)
```

```
        pre=BSF("bsfPrefixReturnValue", 0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        len=BSF("arrayLength", foo)

::method "test_27"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("int.class", 5)
        r=BSF("arrayPut", arr, 12345, 2)
        pre=BSF("bsfPrefixReturnValue", 0)
        x=BSF("arrayAt", arr, 2)
        self~assertEquals(x, 12345)

::method "test_28"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("int.class", 5)
        r=BSF("arrayPut", arr, 12345, 2)
        pre=BSF("bsfPrefixReturnValue", 0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        x=BSF("arrayAt", arr, 7)

::method "test_29"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("int.class", 5)
        r=BSF("arrayPut", arr, 12345, 2)
        pre=BSF("bsfPrefixReturnValue", 0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        x=BSF("arrayAt", arr,)

::method "test_30"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("int.class", 5)
        r=BSF("arrayPut", arr, 12345, 2)
        pre=BSF("bsfPrefixReturnValue", 0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        x=BSF("arrayAt", arr)

::method "test_31"
        pre=BSF("bsfPrefixReturnValue", 1)
        arr=bsf.createArray("int.class", 5)
        r=BSF("arrayPut", arr, 12345, 2)
        pre=BSF("bsfPrefixReturnValue", 0)
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        x=BSF("arrayAt",)

return
```

BSFVersion.testGroup

```
#!/usr/bin/rexx
  parse source . . fileSpec;

  group = .TestGroup~new(fileSpec)
```

```
  group~add(.BSFVersion.testGroup)

  if group~isAutomatedTest then return group

  testResult = group~suite~execute~~print

return testResult
-- End of entry point.

::requires "ooTest.frm"     -- load the ooRexxUnit classes
::requires "BSF.cls"
::requires "rxregexp.cls"

::class "BSFVersion.testGroup" subclass ooTestCase public

::method "test_1"
        .local~ver = 2

        r=BSFVersion()
        self~AssertNotNull(r)

::method "test_2"
        IF .ver = 4 THEN errorCode=88.922; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSFVersion("foo")

return
```

EventListener.testGroup

```
#!/usr/bin/rexx
  parse source . . fileSpec;

  group = .TestGroup~new(fileSpec)
  group~add(.EventListener.testGroup)

  if group~isAutomatedTest then return group

  testResult = group~suite~execute~~print

return testResult
-- End of entry point.

::requires "ooTest.frm"     -- load the ooRexxUnit classes
::requires "BSF.cls"

::class "EventListener.testGroup" subclass ooTestCase public

::method "test_1"
        .local~ver = 2

        e=BSF("postEventText", "eventtest")
        self~assertEquals(e, "eventtest")

::method "test_2"
        e=BSF("postEventText", "eventtest", 0)
        self~assertEquals(e, "eventtest")
```

```
::method "test_11"
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        e=BSF("postEventText", "eventtest", 1, 1)


return
```

eventText.testGroup

```
#!/usr/bin/rexx
  parse source . . fileSpec;

  group = .TestGroup~new(fileSpec)
  group~add(.eventText.testGroup)

  if group~isAutomatedTest then return group

  testResult = group~suite~execute~~print

return testResult
-- End of entry point.

::requires "ooTest.frm"     -- load the ooRexxUnit classes
::requires "BSF.cls"

::class "eventText.testGroup" subclass ooTestCase public

::method "test_1"
        .local~ver = 2

        e=BSF("postEventText", "eventtest")
        self~assertEquals(e, "eventtest")

::method "test_2"
        e=BSF("postEventText", "eventtest", 0)
        self~assertEquals(e, "eventtest")

::method "test_3"
        e=BSF("postEventText", "eventtest", 1)
        self~assertEquals(e, "eventtest")

::method "test_4"
        e=BSF("postEventText", "eventtest", 2)
        self~assertEquals(e, "eventtest")

::method "test_5"
        e=BSF("postEventText", "eventtest", 3)
        self~assertEquals(e, "eventtest")

::method "test_6"
        e=BSF("postEventText", "eventtest", 1111111111)
        self~assertEquals(e, "eventtest")

::method "test_7"
        e=BSF("postEventText", "eventtest")
```

```
        r=BSF("pollEventText", 100)
        self~assertEquals(r, "eventtest")

::method "test_8"
        e=BSF("postEventText", "eventtest")
        r=BSF("pollEventText", 500)
        self~assertEquals(r, "eventtest")

::method "test_9"
        e=BSF("postEventText", "eventtest")
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("pollEventText", -1)

::method "test_10"
        e=BSF("postEventText", "eventtest")
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("pollEventText", -500, 1, 1, 1, 1, 1)

::method "test_11"
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        e=BSF("postEventText", "eventtest", 1, 1)

/*
::method "test_12"
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("pollEventText", 1, 1, 1)
*/

::method "test_13"
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        e=BSF("pollEventText", 1, 1, 1, 1, 1, 1, "foo")

return
```

FieldValue.testGroup

```
#!/usr/bin/rexx
  parse source . . fileSpec;

  group = .TestGroup~new(fileSpec)
  group~add(.FieldValue.testGroup)

  if group~isAutomatedTest then return group

  testResult = group~suite~execute~~print

return testResult
-- End of entry point.

::requires "ooTest.frm"     -- load the ooRexxUnit classes
::requires "BSF.cls"
```

```
::class "FieldValue.testGroup" subclass ooTestCase public

::method "test_1"
        .local~ver = 2

        n=BSF("new", "jad", "java.awt.Dimension")
        f=BSF("getFieldValue", "jad", "width")
        self~assertEquals(f, 0)

::method "test_2"
        n=BSF("new", "jad", "java.awt.Dimension")
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        f=BSF("getFieldValue", "jad", "foo")

::method "test_3"
        n=BSF("new", "jad", "java.awt.Dimension")
        f=BSF("setFieldValue", "jad", "width", "10")
        f=BSF("getFieldValue", "jad", "width")
        self~assertEquals(f, 10)

::method "test_4"
        n=BSF("new", "jad", "java.awt.Dimension")
        f=BSF("setFieldValue", "jad", "width", 10)
        f=BSF("getFieldValue", "jad", "width")
        self~assertEquals(f, 10)

::method "test_5"
        n=BSF("new", "jad", "java.awt.Dimension")
        f=BSF("setFieldValue", "jad", "width", "-10")
        f=BSF("getFieldValue", "jad", "width")
        self~assertEquals(f, "-10")

::method "test_6"
        n=BSF("new", "jad", "java.awt.Dimension")
        f=BSF("setFieldValue", "jad", "width", -10)
        f=BSF("getFieldValue", "jad", "width")
        self~assertEquals(f, -10)

::method "test_7"
        n=BSF("new", "jad", "java.awt.Dimension")
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        f=BSF("setFieldValue", "jad", "width", "10.3")

::method "test_8"
        n=BSF("new", "jad", "java.awt.Dimension")
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        f=BSF("setFieldValue", "jad", "width", 10.3)

::method "test_9"
        n=BSF("new", "jad", "java.awt.Dimension")
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        f=BSF("setFieldValue", "jad", "foo", 10)

::method "test_10"
        n=BSF("new", "jad", "java.awt.Dimension")
        f=BSF("setFieldValueStrict", "jad", "width", "int", "10")
        f=BSF("getFieldValueStrict", "jad", "width")
```

```
        self~assertEquals(f, 10)

::method "test_11"
        n=BSF("new", "jad", "java.awt.Dimension")
        f=BSF("setFieldValueStrict", "jad", "width", "int", "10")
        f=BSF("getFieldValue", "jad", "width")
        self~assertEquals(f, 10)

::method "test_12"
        n=BSF("new", "jad", "java.awt.Dimension")
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        f=BSF("getFieldValueStrict", "jad", "foo")

::method "test_13"
        n=BSF("new", "jad", "java.awt.Dimension")
        f=BSF("setFieldValueStrict", "jad", "width", "int", "10")
        f=BSF("getFieldValueStrict", "jad", "width")
        self~assertEquals(f, 10)

::method "test_14"
        n=BSF("new", "jad", "java.awt.Dimension")
        f=BSF("setFieldValue", "jad", "width", 10)
        f=BSF("getFieldValueStrict", "jad", "width")
        self~assertEquals(f, 10)

::method "test_15"
        n=BSF("new", "jad", "java.awt.Dimension")
        f=BSF("setFieldValueStrict", "jad", "width", "-10")
        f=BSF("getFieldValue", "jad", "width")
        self~assertEquals(f, "-10")

::method "test_16"
        n=BSF("new", "jad", "java.awt.Dimension")
        f=BSF("setFieldValueStrict", "jad", "width", "int", -10)
        f=BSF("getFieldValueStrict", "jad", "width")
        self~assertEquals(f, -10)

::method "test_17"
        n=BSF("new", "jad", "java.awt.Dimension")
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        f=BSF("setFieldValueStrict", "jad", "width", "int", "10.3")

::method "test_18"
        n=BSF("new", "jad", "java.awt.Dimension")
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        f=BSF("setFieldValueStrict", "jad", "width", "int", 10.3)

::method "test_19"
        n=BSF("new", "jad", "java.awt.Dimension")
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        f=BSF("setFieldValueStrict", "jad", "foo", "int", 10)

::method "test_20"
        n=BSF("new", "jad", "java.awt.Dimension")
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        f=BSF("setFieldValueStrict", "jad", "foo", "long", 10)
```

```
return
```

functions.testGroup

```
#!/usr/bin/rexx
  parse source . . fileSpec;

  group = .TestGroup~new(fileSpec)
  group~add(.functions.testGroup)

  if group~isAutomatedTest then return group

  testResult = group~suite~execute~~print

return testResult
-- End of entry point.

::requires "ooTest.frm"     -- load the ooRexxUnit classes
::requires "BSF.cls"
::requires "rxregexp.cls"

::class "functions.testGroup" subclass ooTestCase public

::method "test_1"
        ret=BsfQueryRegisteredFunctions()
        self~assertNotNull(BsfQueryRegisteredFunctions.0)

::method "test_2"
        ret=BsfQueryAllFunctions()
        self~assertNotNull(BsfQueryAllFunctions.0)

::method "test_3"
        ret=BsfQueryRegisteredFunctions(s.)
        self~assertNotNull(s.0)

::method "test_4"
        ret=BsfQueryAllFunctions(s.)
        self~assertNotNull(s.0)

return
```

getBSFManager.testGroup

```
#!/usr/bin/rexx
  parse source . . fileSpec;

  group = .TestGroup~new(fileSpec)
  group~add(.getBSFManager.testGroup)

  if group~isAutomatedTest then return group

  testResult = group~suite~execute~~print
```

```
return testResult
-- End of entry point.

::requires "ooTest.frm"     -- load the ooRexxUnit classes
::requires "BSF.cls"

::class "getBSFManager.testGroup" subclass ooTestCase public

::method "test_1"
        .local~ver = 2

        mgr=BSF("getBSFManager")
        self~assertNotNull(mgr)

::method "test_2"
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        mgr=BSF("getBSFManager", "foo")

return
```

getStaticValue.testGroup

```
#!/usr/bin/rexx
  parse source . . fileSpec;

  group = .TestGroup~new(fileSpec)
  group~add(.getStaticValue.testGroup)

  if group~isAutomatedTest then return group

  testResult = group~suite~execute~~print

return testResult
-- End of entry point.

::requires "ooTest.frm"     -- load the ooRexxUnit classes
::requires "BSF.cls"

::class "getStaticValue.testGroup" subclass ooTestCase public

::method "test_1"
        .local~ver = 2

        m=bsf("getStaticValue","java.lang.Integer", "MAX_VALUE")
        self~assertEquals(m, 2147483647)

::method "test_2"
        m=bsf("getStaticValueStrict","java.lang.Integer", "MAX_VALUE")
        self~assertEquals(m, 2147483647)

::method "test_3"
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        m=bsf("getStaticValue","java.lang.Integer", "foo")

::method "test_4"
```

```
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        m=bsf("getStaticValue","java.lang.Integer", )

::method "test_5"
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        m=bsf("getStaticValue","java.lang.Integer", "")

::method "test_6"
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        m=bsf("getStaticValueStrict","java.lang.Integer", "foo")

::method "test_7"
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        m=bsf("getStaticValueStrict","java.lang.Integer", )

::method "test_8"
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        m=bsf("getStaticValueStrict","java.lang.Integer", "")

return
```

invoke.testGroup

```
#!/usr/bin/rexx
  parse source . . fileSpec;

  group = .TestGroup~new(fileSpec)
  group~add(.invoke.testGroup)

  if group~isAutomatedTest then return group

  testResult = group~suite~execute~~print

return testResult
-- End of entry point.

::requires "ooTest.frm"     -- load the ooRexxUnit classes
::requires "BSF.cls"

::class "invoke.testGroup" subclass ooTestCase public

::method "test_1"
        .local~ver = 2

        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        s=BSF("loadClass", "foo")

::method "test_2"
        n=BSF("new", "jls", "java.lang.String", "teststring")
        r=BSF("invoke", "jls", "length")
        self~assertEquals(r, 10)
```

```
::method "test_3"
        n=BSF("new", "jls", "java.lang.String", "teststring")
        r=BSF("invoke", "jls", "compareTo", "teststring")
        self~assertEquals(r, 0)

::method "test_4"
        n=BSF("new", "jls", "java.lang.String", "teststring")
        r=BSF("invoke", "jls", "length")
        self~assertEquals(r, 10)

::method "test_5"
        n=BSF("new", "jls", "java.lang.String", "teststring")
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("invoke", "jls", "length", "foo")

::method "test_6"
        n=BSF("new", "jls", "java.lang.String", "teststring")
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("invoke", "jls", "length", "")

::method "test_7"
        n=BSF("new", "jls", "java.lang.String", "teststring")
        r=BSF("invokeStrict", "jls", "compareTo", "String", "teststrin
g")
        self~assertEquals(r, 0)

::method "test_8"
        n=BSF("new", "jls", "java.lang.String", "teststring")
        r=BSF("invoke", "jls", "compareTo", "foo")
        self~assertNotEquals(r, 0)

::method "test_9"
        n=BSF("new", "jls", "java.lang.String", "teststring")
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        r=BSF("invokeStrict", "jls", "compareTo", "int", "teststring")

::method "test_10"
        n=BSF("new", "jls", "java.lang.String", "teststring")
        r=BSF("invoke", "jls", "valueOf", 2)
        self~assertEquals(r, 2)

::method "test_11"
        n=BSF("new", "jls", "java.lang.String", "teststring")
        r=BSF("invokeStrict", "jls", "valueOf", "double", 2)
        self~assertEquals(r, 2)

return
```

setRexxNullString.testGroup

```
#!/usr/bin/rexx
  parse source . . fileSpec;
```

```
  group = .TestGroup~new(fileSpec)
  group~add(.setRexxNullString.testGroup)

  if group~isAutomatedTest then return group

  testResult = group~suite~execute~~print

return testResult
-- End of entry point.

::requires "ooTest.frm"      -- load the ooRexxUnit classes
::requires "BSF.cls"

::class "setRexxNullString.testGroup" subclass ooTestCase public

::method "test_1"
        .local~ver = 2

        nns=BSF("setRexxNullString", "newnullstring")
        self~assertEquals(nns,"newnullstring")

::method "test_2"
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        nns=BSF("setRexxNullString", "newnullstring", "foo")

::method "test_3"
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        nns=BSF("setRexxNullString", )

return
```

sleep.testGroup

```
#!/usr/bin/rexx
  parse source . . fileSpec;

  group = .TestGroup~new(fileSpec)
  group~add(.sleep.testGroup)

  if group~isAutomatedTest then return group



  testResult = group~suite~execute~~print

return testResult
-- End of entry point.

::requires "ooTest.frm"      -- load the ooRexxUnit classes
::requires "BSF.cls"

::class "sleep.testGroup" subclass ooTestCase public

::method "test_1"
        .local~ver = 2
```

```
        call TIME "R"
        ret=BSF("sleep", 0.5)
        time_elapsed = TIME("R")
        condition=0
        IF (time_elapsed < 0.6) & (time_elapsed > 0.4) THEN
        DO
                condition=1
        END
        self~assertTrue(condition)

::method "test_2"
        call TIME "R"
        ret=BSF("sleep", 1)
        time_elapsed = TIME("R")
        condition=0
        IF (time_elapsed < 1.1) & (time_elapsed > 0.9) THEN
        DO
                condition=1
        END
        self~assertTrue(condition)

::method "test_3"
        call TIME "R"
        ret=BSF("sleep", 1.1)
        time_elapsed = TIME("R")
        condition=0
        IF (time_elapsed < 1.2) & (time_elapsed > 1.0) THEN
        DO
                condition=1
        END
        self~assertTrue(condition)

::method "test_4"
        call TIME "R"
        ret=BSF("sleep", 0)
        time_elapsed = TIME("R")
        condition=0
        IF time_elapsed < 0.1 THEN
        DO
                condition=1
        END
        self~assertTrue(condition)

::method "test_5"
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        ret=BSF("sleep", 0,1)

::method "test_6"
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        ret=BSF("sleep", )

::method "test_7"
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        ret=BSF("sleep")

return
```

version.testGroup

```
#!/usr/bin/rexx
  parse source . . fileSpec;

  group = .TestGroup~new(fileSpec)
  group~add(.version.testGroup)

  if group~isAutomatedTest then return group

  testResult = group~suite~execute~~print

return testResult
-- End of entry point.

::requires "ooTest.frm"      -- load the ooRexxUnit classes
::requires "BSF.cls"
::requires "rxregexp.cls"

::class "version.testGroup" subclass ooTestCase public

::method "test_1"
        .local~ver = 2

        self~AssertNotNull(bsf("version","All"))

::method "test_2"
        self~AssertNotNull(bsf("version","ArrayWrapper"))

::method "test_3"
        self~AssertNotNull(bsf("version","BsfManager"))

::method "test_4"
        self~AssertNotNull(bsf("version","EnumerationWrapper"))

::method "test_5"
        self~AssertNotNull(bsf("version","Java4Rexx"))

::method "test_6"
        self~AssertNotNull(bsf("version","RexxAndJava"))

::method "test_7"
        self~AssertNotNull(bsf("version","RexxEngine"))

::method "test_8"
        self~AssertNotNull(bsf("version","Supplier"))

::method "test_9"
        IF .ver = 4 THEN errorCode=40.900; ELSE errorCode=40.1
        self~expectSyntax(errorCode)
        var=bsf("version","not_existing")

::method "test_10"
        self~AssertNotNull(bsf("version","Al"))

::method "test_11"
```

```
        self~AssertNotNull(bsf("version","AR"))

::method "test_12"
        self~AssertNotNull(bsf("version","B"))

::method "test_13"
        self~AssertNotNull(bsf("version","E"))

::method "test_14"
        self~AssertNotNull(bsf("version","J"))

::method "test_15"
        self~AssertNotNull(bsf("version","RexxA"))

::method "test_16"
        self~AssertNotNull(bsf("version","RexxE"))

::method "test_17"
        self~AssertNotNull(bsf("version","S"))

return
```