



# A generic R plugin dispatcher for BASE

Henrik Bengtsson

Mathematical Statistics, Centre for Mathematical Sciences,  
Lund University, Sweden  
hb@maths.lth.se



## Abstract

THE BioArray Software Environment (BASE) [5] is a free web-based database system for microarray experiments. It stores and analyses large amounts of data covering everything from array design to statistical analysis. In addition to built-in methods, BASE utilizes third-party plugin modules to analyze data. To date, plugins have been written mainly in Perl, C/C++ and Java, but few supportive methods are available to do this in a generic manner.

We have developed a plugin dispatcher that ease the effort to develop BASE plugins in R and which provides immediate access to the large CRAN and Bioconductor repositories for statistical analysis. The plugin dispatcher encapsulates common tasks such as parsing BASE file structures passed to the plugin from BASE, loads plugin specific R code, provides rich log methods, calls the plugin specific code with automatic exception handling, and returns data back to BASE. In addition, reports can easily be generated from R Server Page (RSP) templates. The actual plugin is defined by a single method that takes the pre-processed assay data, which is accessible via a well-defined application protocol interface (API), as the first argument, and the plugin parameters as the following arguments. Data returned by the method is interpreted and returned to BASE in correct format by the plugin dispatcher.

The platform-independent plugin dispatcher is implemented in the R package *aroma.Base* available via <http://www.maths.lth.se/bioinformatics/>.

## 1. Introduction

In order to correctly analyze microarray data, high-quality and well-tested statistical tools that implement peer-reviewed methods must be used. The majority of these tools are published on the CRAN and the Bioconductor package repositories. Since these packages are written in R, and current plugins for BASE have mainly been developed in Perl and C/C++, the recent progress in statistical microarray analysis is de facto *not* available in BASE.

By developing an R *plugin dispatcher* for BASE, which can run virtually any R code and R packages, we hope that the rich set of statistical methods published soon will be available also in BASE. Plugins for *aroma* [2] are available.

## 2. Objectives

Our objectives has been:

- Use R for statistical microarray analysis in BASE.
- Easy access to Bioconductor and CRAN in BASE.
- Generate reports in HTML and other formats.
- Standardize how BASE plugins are written in R.
- Encapsulate data transfer to and from BASE.
- Preserve memory.
- Simplify exception handling and event logging.
- Simple installation and platform independent.
- Open source.

## 3. Overview

The plugin dispatcher operates inbetween BASE and the R plugin code. It supports the R plugin by parsing and loading data file structures send from BASE, evaluates its code, catches exceptions, generates reports and returns updated data to BASE in a valid format. See also Figure 1.

With this, an R plugin for BASE becomes a single method named `onRun()` defined in, say, `onRun.R` put in the plugin directory (see Section 9). This method has full access to all of R and does *not* have to communicate with BASE directly. This is taken care of by the dispatcher.

## 4. Event logging

Events occuring during startup, running, failure, report generation, and completion are logged in full detail together with system and timing information, and available for troubleshooting. A progress bar is ready for reporting progress to the BASE user interface.

## 5. Smart comments

Comments is used to improve readability of the plugin's source code. Why not write these comments to log too? LComments, a subclass of SmartComments, have format `"#L*# <comment>"` where `"#"` is a single character that controls how the comment is written to log, cf. Figure 2.

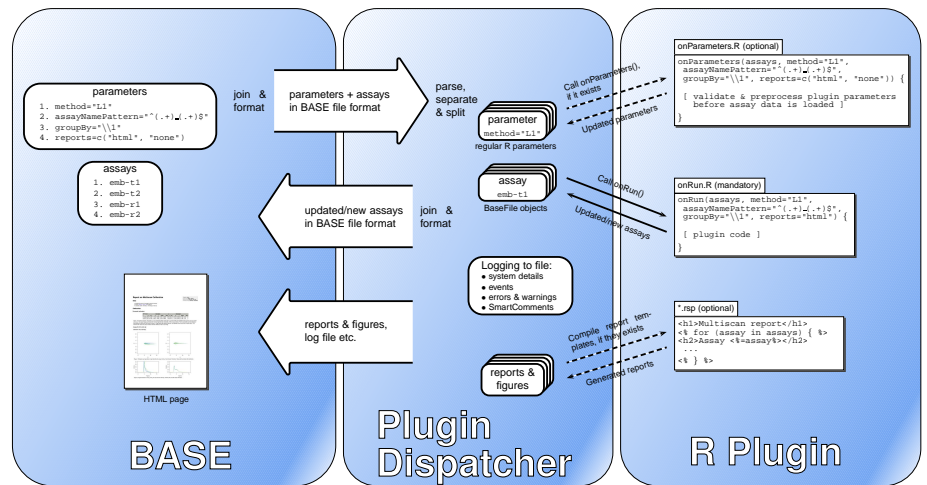


Figure 1. Figure illustrating how the plugin dispatcher retrieves plugin parameters and assay data from BASE, wraps them up in standard R objects and passes them on to the R plugin, which is a single method named `onRun()`. Data returned by `onRun()` is formatted and send back to BASE together with log files and reports generated from RSP templates.

```

for (array in assays) {
  #L# Calibrating array S[array]
  ...
  #L# Number of scans: ${nbOfScans}
  ...
  #L#
}
plugin code (before)
  
```

```

for (array in assays) {
  enter(log, "Calibrating array S[array]")
  ...
  cat(log, "Number of scans: ${nbOfScans}")
  ...
  wait(log)
}
internal code (after)
  
```

Figure 2. Preprocessing of SmartComments. Left: The plugin developer includes LComments in the code to increase readability and automatize logging. Right: The plugin code is preprocessed by the plugin dispatcher so that LComments are expanded to log statements.

The amount of details logged is controlled by a threshold. Automatic indentation ("begin-end statements"), various output formats, headers and rulers are available. Log statements are processed as GStrings, e.g. `$(array)` expands to the value of array. SmartComments has zero overhead if not preprocessed.

## 6. Report generator

Reports in HTML and other formats can be generated from R Server Page (RSP) templates. An example is shown in Figure 3. An RSP is similar to a Java Server Page (JSP), and can contain R code snippets to control output. RSPs are processed by compilers in the R.rsp package. See also example in Figure 1.

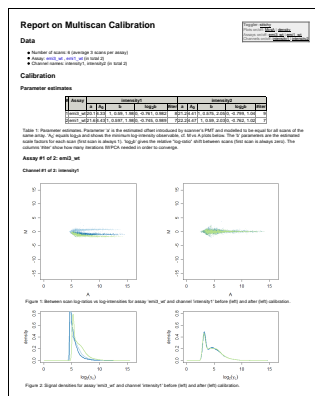


Figure 3. An example of an HTML report on multiscan calibration [4, 3]. The result table and the graphs are generated dynamically. Virtually any output format is supported.

## 7. Exception handling

When an error occurs that is not taken care of by the plugin, the dispatcher catches it, reports it to the log, calls optional plugin method `onError()`, and returns control to BASE. Warnings are written to both the log and the plugin message output.

## 8. Memory optimization

When the plugin dispatcher reads data from BASE, it separates the signals from each assay into a separate object, cf. Figure 1. The data for each assay is cached to file, and only read into memory when needed. This procedure is fully taken care of by the object-oriented API built upon the R.oo package [1]. Thus, there is no limit in the number of arrays/plugins for methods such as curve-fit normalization can handle.

## 9. How

1. Create plugin directory, e.g. `$user/plugins/calibrateMultiscan/`.
2. Copy \*.R files to this directory. One file should define `onRun(assays, arg1, ...)` where `assays` is a BaseFile object and `arg1` etc. are plugin parameters.
3. Copy or link shell script `basePluginDispatcher` into the same directory. Set executable rights for everyone.
4. In BASE, setup a new plugin. Set executable to `calibrateMultiscan/basePluginDispatcher`. Use `serial` file format.
5. Put any RSP report templates in `$user/plugins/calibrateMultiscan/rsp/`.
6. Test it.

\*) Required R packages, including *aroma.Base* & *co*, can be installed locally in `$user/plugins/library/` without admin rights.

## 10. Requirements

- BASE v1.2 or newer\*
- R v2.0 or newer
- R packages *aroma.Base*, *R.rsp*, *R.utils* & *R.oo*.

\*) Plugins and the dispatcher can run & tested without BASE.

## References

- [1] Henrik Bengtsson. The R.oo package - object-oriented programming with references using standard R code. In Kurt Hornik, Friedrich Leisch, and Achim Zeileis, editors, *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)*, Vienna, Austria, 2003.
- [2] Henrik Bengtsson. *aroma* - An R Object-oriented Microarray Analysis environment. Preprint in Mathematical Sciences 2004:18, Centre for Math. Sci., Lund Univ., 2004.
- [3] Henrik Bengtsson and Ola Hössjer. Methodological study of affine transformations of gene expression data with proposed normalization method. Preprints in Mathematical Sciences 2003:38, Centre for Math. Sci., Lund Univ., 2003.
- [4] Henrik Bengtsson, Göran Jönsson, and Johan Vallon-Christersson. Calibration and assessment of channel-specific biases in microarray data with extended dynamical range. *BMC Bioinfo.*, 5(177), 2004.
- [5] Lao H. Saal, Carl Troein, Johan Vallon-Christersson, Sofia Gruberger, Ake Borg, and Carsten Peterson. BioArray Software Environment (BASE): a platform for comprehensive management and analysis of microarray data. *Genome Bio.*, 3(8):SOFTWARE0003, 2002.