We present a poster of the Elastic JavaDoc (EJD), a documentation presentation system. The system is based on the JavaDoc idea about keeping documentation in the source code (the principle of high proximity), and structuring this documentation in named chunks (i.e. an author chunk, a bug chunk, a todo chunk). Before using the system the EJD reads the source codes, extracts the documentation information and stores it in a database. The system is now ready to present the documentation.

The presentation layer in the EJD is made up of Java ServerPages custom tags and a JSP webserver. The custom tags include iterators, if’s, links and “information tags”. The iterators can iterate classes, methods, fields etc., the if’s creates a branching mechanism, the link tags creates links and bookmark markers on the pages. Finally the information tags output unformatted information. The tags are to be used in conjunction with a formatting mechanism, in our case we have chosen HTML, but \LaTeX{} or any markup languages could be used just as well.

Creating presentation pages is almost like programming a program. Information tags can be placed in any order (as long as they are in the scope of their iterator) and the branching mechanism allows custom presentation possibilities, i.e. using pictures or changing typography (colour or style) according to i.e. access modifiers or bug issues.

Each field in computer science tends to have specific and different needs on what to be documented. What’s important in mobile networks are usually not the same as what is important in the realm of algorithms. Extending the EJD system to understand new documentation chunks requires development of new tags and extending the EJD source code parser. To the average Java programmer this should yield no problem.

Rationale
The absence of identifiable chunks of information (documentation) in the source codes cripples the EJD system somewhat. It would still be superior to the static produced HTML files the Java API is made up from. However, many interesting features will cease to exist. The rationale of the EJD is the presence of documentation, and for three reasons

- According to litterature in the field of documentation “extra-semantic constraints” are needed in order to be able to properly use and re-use code. These extra-semantic constraints are natural language descriptions associated with program structure and semantic constraints not expressed by the programming language itself.
- By letting the programmer produce documentation in pre-defined chunks, important information will unlikely be left out or forgotten. Also the programmer is forced to go through the code on a higher cognitive level, which probably will result in better programs.
- When programming on a large project (time or size) a lot of maintenance and accounting is required (such as bugs, todos, change request, author of the code, versioning etc). Keeping such information in the source code enables the EJD to utilized it; i.e. a programmer could see all the “todo’s” in the parts of the source code which he authored, or maybe the project manager wishes to see which parts of the program were changed due to a customers change request.

Using the system
Another key feature of the EJD is the tailoring of information to the users needs. Along with the presentation tags, special tags for configuring the output has been developed. Typically the work situation shifts rapidly, i.e. from re-user to maintainer, from the EJD systems point of view this would roughly mean showing only public methods to showing all methods in a class. Changing views is sought to be easiest possible.

Limits
There are limits in the system. Apart from the requirements of the documentation, the tags for presenting the information is somewhat limited. The tags were developed to be easy to use, therefore program language elements such as variables and pointers has been omitted. Instead of a tag points to another tag, tags are “associated”. Within a method-iterator all tags concerning methods are available (such as the name of the current method, its return value, access modifiers).

Finally the system is server based, meaning all request are sent to a server. This reduces the requirements for installed software on the clients, but may result in the system becoming too slow in its responses with the risk that users will be reluctant to constantly change the layout configuration. However from a scientific point of view, making the program
server based, enables research in user behaviour and may re-
veal the users true needs, which could otherwise be hard to
find by interviews.