Namespaces for $\varepsilon xTEX$

Gerd Neugebauer

Overview
Requirements
Encapsulation
Backward Compatibility and Initialization
Concept
Definition of Namespaces
Communication between Namespaces
Namespaces and Groups
Namespaces and Expansion
Explicit Expansion without Import
Namespaces and the Basic Definitions
Design
$\varepsilon xTEX$
Integration into $\varepsilon xTEX$
Tokens
Context
Scanner
Implementation and Extensions
Extensions
Conclusion

EuroTEX 2005
March 2005
Pont-à-Mousson, France
Overview

- Requirements
- Concept
- Design
- Implementation
Requirements

- Information hiding and privacy are basic principles in modern software engineering
- Module system/package system/namespaces provide privacy
- $\TeX$ has no real module system
- $\LaTeX$ packages use naming conventions and a redefined catcode to protect internals
- An namespace extension of $\TeX$ is needed
- The existing code should not be affected
Encapsulation

- Encapsulation: Hiding the current meaning
  - Macros
  - Active Characters
  - Registers
    (count, dimen, toks, ...)
  - Catcodes

- Focus here: Macros and Active Characters
Backward Compatibility and Initialization

- The extension should be backward compatible.
- The operation should be performed in the default namespace if not specified.
- Namespaces must be properly initialized without too much overhead.

- The attempt should work without syntactic sugar: KISS
Definition of Namespaces

- Special toks register for the current namespace.
- Assignment to this register changes the current namespace.

```latex
\namespace{tex.latex.dtk}
\namespace={tex.latex.dtk}
```
Default Namespace

- The default namespace has the empty toks register.

\namespace={}
Accessing the Current Namespace

\the \ and \showthe \ can be used to get access to the current namespace.

\namespace{\texlatexdtk}
\the\namespace

\rightarrow \texlatexdtk
Namespaces provide encapsulation.

Some entities need to be visible outside.

A primitive \texttt{\textbackslash export} should be used to specify potentially visible entities.

\begin{verbatim}
\texttt{\textbackslash export}\{\texttt{\textbackslash abc \textbackslash xyz ~}\}
\end{verbatim}

\texttt{\textbackslash export} acts like a special toks register.

The tokens are stored locally for the namespace.
Communication between Namespaces: Import

A primitive \import should be used to specify potentially visible entities in the target namespace.

\import{tex.latex.dtk}

The import is performed into the current namespace.

All entities exported from the namespace are imported.

The import works like \let

The modification of the definition in both namespaces are independent.
Namespaces and Groups

Namespace interact with the group in the expected way.

- Local definitions are discarded at the end of the current group.

\begin{group}
\namespace{tex}
\gdef\x{123}
\end{group}
\def\y{123}

- \textbackslash x is undefined afterwards
- \textbackslash y is defined in the outer namespace
Namespaces and Groups (2)

- \import defines the entities “group local”
- \import honors the prefix \global
- \global \import imports into the top group:

\begingroup
  \global\import{tex.latex.dtk}
\endgroup

- The imports are preserved past the end of the group
Namespaces and Groups (3)

\begingroup
  \namespace{one}
  \global\export{\x}
  \gdef\x#1{{-#1-}}
\endgroup

- The grouping restricts the effect of namespace
- The \global\export makes the export survive the end of the group
- the \gdef makes the maxro survive the end of the group
Namespaces and Expansion

- The same control sequence name can have different bindings

```latex
\begin{verbatim}
\namespace{two}
\begingroup
  \namespace{one}
  \global\export{\x}
  \gdef\x#1{-#1 \y -}
  \gdef\y{in one}
\endgroup
\import{one}
\def\y{two}
\x \y
\end{verbatim}
```

\[ \text{two in one} \]
Explicit Expansion without Import

\begingroup
\namespace{tex}\aftergroup\abc
\endgroup

- The namespace is attached to a token when it is created and not, when it is expanded
- The token \abc will carry the namespace tex
- The grouping restricts the namespace to the two tokens \expandafter and \abc.
- \expandafter delays the expansion until the group is closed
Namespaces and the Basic Definitions

- A new namespace should not start empty – like \texttt{init}\TeX

\texttt{plain.tex/\LaTeX/context} provide many useful macros

Solution: Search Strategy for Definitions
- Search in the specified namespace first
- Search in the default namespace if needed
Namespaces for \texttt{\LaTeX}

Gerd Neugebauer

Overview

Requirements

Encapsulation

Backward Compatibility and Initialization

Concept

Definition of Namespaces

Communication between Namespaces

Namespaces and Groups

Namespaces and Expansion

Explicit Expansion without Import

Namespaces and the Basic Definitions

Design

\texttt{\LaTeX}

Integration into \texttt{\LaTeX}

Tokens

Context

Scanner

Implementation and Extensions

Extensions

Conclusion

\begin{itemize}
  \item “Extensible” \LaTeX
  \item Object-oriented reimplementation of \LaTeX
  \item \url{http://www.extex.org}
\end{itemize}
Integration into \textit{\LaTeX}

- Extend (some) tokens.
- Extend the Group and the Context.
- Extend The binding mechanisms for control sequences and active characters needs to be extended to take into account the fallback to the default namespace.
- Implement the primitive \texttt{\namespace}.
- Implement the primitive \texttt{\export}.
- Implement the primitive \texttt{\import}.
The changes are localizable at a few places.
Extend the containers for control sequence tokens and active character tokens.

Other tokens are not affected.
Context

- The Context is the container for all data (like the eq table)
- The Context maintains a stack of Groups in its current implementation

<table>
<thead>
<tr>
<th>Context</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ getNamespace() : String</td>
<td>+ getNamespace() : String</td>
</tr>
<tr>
<td>+ setNamespace(namespace : String) : void</td>
<td>+ setNamespace(namespace : String) : void</td>
</tr>
</tbody>
</table>

- Extend the Context and the Group with getters and setters for the current namespace.
- Extend `getCode()` in the Context to contain the search strategy.
The invocation of the token factory is extended to contain the namespace.

The token factory has to be extended accordingly.
Implementation

```java
/**...*/
public class ControlSequenceToken extends AbstractToken
    implements CodeToken {

    /**...*/
    private String name;

    /**...*/
    private String namespace;

    /**...*/
    protected ControlSequenceToken(final UnicodeChar esc,
        final String name,
        final String namespace) {

        super(esc);
        this.namespace = namespace;
        this.name = name;
    }
}
```

It works!
Extensions

- Extension of namespaces to registers. Partially implemented in \( \varepsilon \LaTeX \) (compile-time configuration)
  - Interferes with plain.tex
  - Experiments not convincing yet
- Selective import of dedicated tokens
- Renaming during the import
- Search strategy with intermediate levels of packages (decomposition of namespace identifier)
- Syntactic sugar
Conclusion

- Namespaces can be provided with a few modifications of $\varepsilon\chi\TeX$.
- Namespaces for control sequences and active characters are a good first step.
- The extension is performed minimalistically.
- Namespaces are an offer for macro writers.
- Make the best use of it.