Namespaces for $\varepsilon_\mathcal{X}$TEX

Gerd Neugebauer

February 21, 2005

Namespaces for TEX are a long awaited extension. In this talk the requirements for such an extension are described. Namespaces primarily restrict the visibility of macros and active characters. Thus the probability of name clashes is reduced. As addition one can imagine to apply namespaces to other entities like registers, catcodes etc as well.

$\varepsilon_\mathcal{X}$TEX is an attempt to reimplement TEX. The major goals behind the reimplemenbtation are a modular and configurable structure tailored towards experiments and extensibility.

Fortunately the integration of namespaces can be located at very few places in the $\varepsilon_\mathcal{X}$TEX architecture. As a consequence an implementation idea for $\varepsilon_\mathcal{X}$TEX can be sketched and the experimental implementation in $\varepsilon_\mathcal{X}$TEX is shown.

1 Introduction

With the vast amount of packages emerging on CTAN the need came up to separate the packages. In other programming languages this is accomplished by using namespaces – sometimes also called modules or packages. The classical TEX system lacks such a mechanism.

In this proposal an analysis is provided which shows where modifications of a TEX-like system are necessary to implement namespaces. The proposed solution tries to be minimalistic. This means that one goal is to change the underlying system as few as possible. In addition existing TEX code should continue to work without any change.

2 Encapsulation

The primary requirement for namespaces is that they encapsulate the internals of "modules". This means that each information about the state is hidden in the outside world. The access is enabled via well-defined interfaces only.

In this document the focus is put onto namespaces for control sequences and active characters.

3 Backward Compatibility and Initialization

The namespace extension should be backward compatible. This means that the behaviour of the system should not depend of the use of a namespace. This can be achieved by using a default namespace in any case where no other namespace is specified.

On the other hand the definitions in the namespace should be properly initiated. Thus we want to ensure that the namespaces can be used without the burden of too much initialization.

4 Definition of Namespaces

The definition of the current namespace is just a string to be kept somewhere. In terms of TEX it is advisable to store the current namespace as tokens in a special tokens register to allow read and write access to it.

Consider the name `\namespace` for this toks register then the following instruction can be used to advice TEX to set the appropriate namespace:

```
\namespace{tex.latex.dtk}
```

The default namespace should be denoted by the empty toks register:

```
\namespace{}
```

The wellknown primitives `\the` and `\showthe` can be used to get access to the current value of the namespace:

```
\namespace{tex.latex.dtk}
\the\namespace
```

5 Communication between Namespaces

Namespaces provide a means for separation of different modules. Thus we need a way to communicate between namespaces. For this purpose we want to provide a primitive to declare that a certain set of entities are visible from the outside. All entities not declared to be visible are purely private.

The primary entities to consider are macros and active characters. They are characterized by tokens. Thus we can again use a token list to keep this information.

Consider the name `\export` for the names toks register. Then the following example declares that the macros `\abc` and `\xyz` and the (active) character `~` can potentially be accessed non-locally:

```
\export{\abc \xyz ~}
```

The other side of communication is the import of the exported entities into another namespace. In analogy to the name export we want to call this functionality import. For the import it is sufficient to name the namespace to be imported:

```
\import{tex.latex.dtk}
```

The semantics is that all entities exported by the namespace – i.e. contained in the toks register `\export` – are assigned to in the current namespace as well. This is similar to a multitude of let invocations.

As a consequence of this definition the meaning of a macro can be changed in the defining namespace without affecting the meaning in the importing namespace.

6 Namespaces and Groups

Since the main task of the declarations if performed by special tokens registers it is clear that the namespaces are coherent with the groups structure already present in TEX.

In the following example the macro `a` is defined in the namespace `tex`. Since the group ends afterwards the namespace returns to its previous value. Thus the definition of `\y` is performed in the outer namespace.

```
\begingroup
  \namespace{tex}
  \gdef\x{123}
\endgroup
\def\y{123}
```

One question which arises is, how the macro `\import` interacts with the grouping. The answer to this question is that the import should influence the current group only. Similar to the definition of `\let` the prefix command `\global` can be used to indicate that the imports should be applied globally instead of locally in the current group:

```
\global\import{tex.latex.dtk}
```

The macro `\import` has to take into account the `\global` prefix.

Another inference of namespaces and groups can be seen in the following example:

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

Note that the `\export` declaration is preceded by a `\global` modifier. Consider the case that this modifier would not be there. Then the end of the group would revert the meaning of `\export` the previous binding. In general this would destroy the intended meaning. The `\global` modifier ensures that the intended meaning of `\export` survives the end of the group and can be used in subsequent imports.

7 Namespaces and Expansion

Let us consider the following example where a macro with an argument is exported:

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

The intuitive meaning of the last expression `\x\y` is that `\y` is taken from the namespace `two` and `\y` is taken from the outer namespace `one`. Now we follow the expansion of `\y`. It leads to

```
-\y␣\y-
```

where the first `\y` comes from the argument. As such it is rooted in the outer namespace `one`. Whereas the second `\y` is defined in the namespace `two`.

As we can conclude that the namespace has to be attached to the token. Then the two tokens can be expanded in their original namespaces. To illustrate this we use the namespace as subscript to the macro name. With this convention we have the following tokens in the situation above:

-\y$_{one}$␣\y$_{two}$-

With this refined understanding we can come to the conclusion that the expansion will indeed lead to the expected result:

```
-two in one-
```

8 Explicit Expansion without Import

In several programming languages there is the possibility to invoke a method in another namespace. With the means we have depicted so far we can achieve the same effect:

```
\begingroup\namespace{tex}\expandafter\abc\endgroup
```

This construct can even be used to expand a macro which is not exported. Since it is in general not a good idea to use this feature no provision is made to provide a shortcut for such an expansion.

In this document a minimalistic approach has been proposed. To provide some syntactic sugar would require some more adaptions to the basic machinery. Since all basic requirements can be fulfilled within the minimalistic approach it has net been considered worthwhile to explore these adaptions any further.

9 Namespaces and the Basic Definitions

Usually nobody actually starts with iniTeX since nearly no definitions and settings are defined in it. At least something like the `plain` definitions are used. With the means given in the previous sections each new namespace would start out like a new iniTeX instance.

To solve this problem, we define a fallback strategy for the resolution of control sequences and active characters. If a definition is not found in the current namespace then the definition is taken from the default namespace.

The default namespace is denoted by the empty token list. Initially the namespace is set to the default namespace.

With this definition it is possible to load the some macros into the default namespace – e.g. the `plain` definitions. Then they are availlable in any namespace unless redefined in it.

10 Implementation in ε𝒳TeX

ε𝒳TeX (`http://www.extex.org`) is a project to provide an implementation of a typesetting system based on the ideas of TeX. It is designed to be highly configurable and should provide a base for extensions and experimentation. As a starting point a TeX compatibility mode is provided.

According to the considerations in the previous sections we need the following additions to ε𝒳TeX:

- The definition of tokens and their factory have to be extended to carry the namespace.
- The group has to be extended to provide means to store the current namespace.
- The binding mechanisms for control sequences and active characters needs to be extended to take into account the fallback to the default namespace.
- The primitive `\namespace` has to be provided which allows the reading and writing access to the namespace stored in the group.
- The primitive `\export` has to be provided to allow access to a special tokens register in the current namespace.
- The primitive `\import` has to be provided which is similar to the implementation of `\let`. New bindings for the exported control sequences or active characters have to be introduced to reference the definitions in the defining namespace.

11 Namespaces for Registers

In the current extension of ε𝒳TeX the registers are not affected by the namespaces. Nevertheless it might be desirable to extend namespaces to registers.

For instance count registers can be made aware of namespaces. Then each namespace can have its own incarnations of count registers. The extensions into this direction is straight forward. Experiments into this directions have been performed within ε𝒳TeXṄevertheless they where not really convincing. The plain format makes use of several count registers. The adaption of the visibility of count registers would require adaptions on the macro level as well.

The restrictions to a limited number of count registers has already been relaxed in εTeX and in ε𝒳TeX even further. Thus with the use of the allocation macros it is no problem to have separate registers in separate modules – even when namespaces are used.

12 Conclusion

In the previous sections we have seen how a basic namespace support has be integrated into ε𝒳TeX. The changes required for this extension are restricted to very few modifications in the core components. These modifications provide a base upon which the externally visible extensions can act. The extensions are purely optional – to be enabled in a configuration file or even loaded dynamically within ε𝒳TeX. Without the definitions of the three new primitives the behaviour of ε𝒳TeX has not been changed.

The base mechanism for the use of namespaces is provided. Now it is up to the macro level to make use of it.