Preprocessing Source Files

Manipulating Graphic Resources

Hans Hagen
Pragma ADE

June 2005

Preprocessing Source Files

This manual describes two mechanisms of (semi) run-time processing or (re)source files. Let's start with source files.

Although in principle it is possible to process files during a TEX run, in practice this proved to be rather error prone. The main reason is that there is no robust way of determining which files need preprocessing. In TEX mode, only real source files may need a treatment, and styles and auxiliary files need to remain untouched. In XML mode, again only the resources need some treatment, and TEX files have to remain untouched.

Examples of preprocessing are 'reencoding a file' or 'applying an XSLT script to a document source'. In all cases, the original is to be untouched, and TEX needs to take the converted file as replacement. Also, when converting files —there can be more than one involved— we need to make sure that only files that did change are preprocessed.

The preprocessing is controlled by a small XML definition file, which is read by TEXEXEC. There can be a file for each (main) source file with suffix `ctx`. Alternatively one can have a file `jobname.ctx`. The next example demonstrates how such a file is constructed:

```
<?xml version='1.0' standalone='yes'?>

<ctx:job>
    <ctx:message>xmldemo</ctx:message>
    <ctx:preprocess>
        <ctx:processors>
            <ctx:processor name='demo'>texmfstart --direct xsltproc --output
                <ctx:value name='new'/> kpse:demo.xsl <ctx:value name='old'/></ctx:processor>
        </ctx:processors>
        <ctx:files>
            <ctx:file processor='demo'><ctx:value name='job' method='nosuffix'/>.xml</ctx:file>
            <!-- ctx:file processor='demo'>demo-*.xml</ctx:file -->
        </ctx:files>
    </ctx:preprocess>
    <ctx:process>
        <ctx:resources>
            <ctx:environment>o-m4all.tex</ctx:environment>
        </ctx:resources>
    </ctx:process>
    <ctx:postprocess>
        <!-- Postprocessing is not yet supported. -->
    </ctx:postprocess>
</ctx:job>
```

You can apply one or more processors to a file, in our case we have only one: demo. Multiple processors are separated by a comma. The processors are defined in their own section. The old and new names will be resolved at the right moment.

```
<ctx:preprocess>
    <ctx:processors>
        <ctx:processor name='step-1'>texmfstart do_step_one <ctx:value name='old'/> steps.tmp</ctx:processor>
        <ctx:processor name='step-2'>texmfstart do_step_two steps.tmp <ctx:value name='new'/></ctx:processor>
    </ctx:processors>
    <ctx:files>
        <ctx:file processor='step-1,step-2'><ctx:value name='job' method='nosuffix'/>.tex</ctx:file>
    </ctx:files>
</ctx:preprocess>
```

A file element can have a wildcard specification, which makes it possible to process a bunch of files. Only files that are changed will be processed. The processed file gets the *extra* suffix prep.

The main proces section can contain a resource section that specifies one or more environment and/or module sections. This information will be passed onto CONTEXT.

You can share definitions by using the include feature. In that case each job has an associated file that looks as follows:

```
<?xml version='1.0' standalone='yes'?>

<ctx:job>
    <ctx:include name='common.ctx'/>
</ctx:job>
```

Inclusion can happen at each level, so the following is legal:

```
<?xml version='1.0' standalone='yes'?>

<ctx:job>
    <ctx:preprocess>
        <ctx:include name='om2mmc.ctx'/>
        <ctx:files>
```

```
            <ctx:file processor='demo'><ctx:value name='job' method='nosuffix'/>.xml</ctx:file>
        </ctx:files>
    </ctx:preprocess>
</ctx:job>
```

This assumes a file `om2mmc.ctx` that looks like:

```
<?xml version='1.0' standalone='yes'?>

<ctx:processors>
    <ctx:processor name='demo'>texmfstart --direct xsltproc --output
        <ctx:value name='new'/> kpse:om2cml.xsl <ctx:value name='old'/></ctx:processor>
</ctx:processors>
```

When everything went well, a log file will be produced. Depending on how CONTEXT is configured, this log file will be consulted for replacement names. Consulting happens by default which means that loading unprocessed files will not take place, which prevents errors. The log file looks like:

```
<?xml version='1.0' standalone='yes'?>

<ctx:preplist>
       <ctx:prepfile done='yes'>myfile.xml</ctx:prepfile>
</ctx:preplist>
```

There can be multiple entries. Here the preprocessed file gets the name `myfile.xml.prep` and when deleted afterwards it will be regenerated a next time that it is needed.

Manipulating Graphic Resources

Including graphic is only possible when the format in which they come are accepted by the backend. If not, they need to be converted. Also, sometimes graphics need to be manipulated before they can be used, think of downsampling a huge 24 MB image into a more handsome 2 MB one. Especially in automated workflows you want this to be done without user intervention and as efficient as possible, i.e. only once.

The manipulating mechanism is just another variant of managing resource libraries and resource logs. It is hooked into the graphic inclusion macros (thereby complicating this mechanisms even more).

A simple conversion is configured (and applied) as follows:

```
\usemodule[res-08] \setups[rl:manipulate]

\setupexternalfigures[location=local,directory=.,conversion=pdf]

\starttext
  \externalfigure[svg/example.svg]
\stoptext
```

Sometimes it makes sense to collect files in a different spot:

```
\usemodule[res-08] \setups[rl:manipulate]

\setupexternalfigures[location=local,directory=.,conversion=lowres,prefix=lowres/]

\starttext
  \externalfigure[whatever.pdf]
\stoptext
```

The manipulations themselves are defined in a file called `rlxtools.rlx`. This file should be placed in a path known to TEX (or in the local path). The RLXTOOLS program takes care of the conversions between the first and successive runs. Graphics are only converted when their timestamp has changes.

In the following examples, the values are expanded when needed. We will not cover each small detail of this setup, which is rather verbose anyway.

```
<?xml version='1.0 standalone='yes'?>

<rl:manipulators>

    <rl:manipulator name='pdf' suffix='svg'>
        <rl:old>
            <rl:value name='path'/>/<rl:value name='file'/>
        </rl:old>
        <rl:new>
            <rl:value name='path'/>/<rl:value name='prefix'/><rl:value name='file' method='nosuffix'/>.pdf
        </rl:new>
        <rl:step>
            inkscape
            --without-gui
            --print="&gt;<rl:value name='path'/>/<rl:value
                name='prefix'/><rl:value name='file' method='nosuffix'/>.ps"
            <rl:value name='path'/>/<rl:value name='file' method='nosuffix'/>.svg
        </rl:step>
        <rl:step>
            texmfstart pstopdf
            <rl:value name='path'/>/<rl:value name='prefix'/><rl:value name='file' method='nosuffix'/>.ps
            <rl:value name='path'/>/<rl:value name='prefix'/><rl:value name='file' method='nosuffix'/>.pdf
        </rl:step>
    </rl:manipulator>

    <rl:manipulator name='pdf' suffix='svgz'>
        <rl:old>
            <rl:value name='path'/>/<rl:value name='file'/>
        </rl:old>
        <rl:new>
            <rl:value name='path'/>/<rl:value name='prefix'/><rl:value name='file' method='nosuffix'/>.pdf
        </rl:new>
        <rl:step>
            inkscape
            --without-gui
            --print="&gt;<rl:value name='path'/>/<rl:value
                name='prefix'/><rl:value name='file' method='nosuffix'/>.ps"
            <rl:value name='path'/>/<rl:value name='file' method='nosuffix'/>.svgz
        </rl:step>
        <rl:step>
```

```
            texmfstart pstopdf
            <rl:value name='path'/>/<rl:value name='prefix'/><rl:value name='file' method='nosuffix'/>.ps
            <rl:value name='path'/>/<rl:value name='prefix'/><rl:value name='file' method='nosuffix'/>.pdf
        </rl:step>
    </rl:manipulator>

</rl:manipulators>
```

The next setup defines conversion as well as downsampling manipulators.

```
<?xml version='1.0 standalone='yes'?>

<rl:manipulators>

    <rl:manipulator name='pdf' suffix='svg'>
        <rl:old>
            <rl:value name='path'/>/<rl:value name='file' method='nosuffix'/>.svg
        </rl:old>
        <rl:new>
            <rl:value name='path'/>/<rl:value name='prefix'/><rl:value name='file' method='nosuffix'/>.pdf
        </rl:new>
        <rl:step>
            texmfstart newpstopdf --convert
            <rl:value name='old'/>
            <rl:value name='new'/>
        </rl:step>
    </rl:manipulator>

    <rl:manipulator name='pdf' suffix='svgz'>
        <rl:old>
            <rl:value name='path'/>/<rl:value name='file' method='nosuffix'/>.svgz
        </rl:old>
        <rl:new>
            <rl:value name='path'/>/<rl:value name='prefix'/><rl:value name='file' method='nosuffix'/>.pdf
        </rl:new>
        <rl:step>
            texmfstart newpstopdf --convert
            <rl:value name='old'/>
            <rl:value name='new'/>
        </rl:step>
    </rl:manipulator>
    <rl:manipulator name='lowres' suffix='pdf'>
        <rl:old>
            <rl:value name='path'/>/<rl:value name='file'/>
        </rl:old>
        <rl:new>
            <rl:value name='path'/>/<rl:value name='prefix'/><rl:value name='file'/>
        </rl:new>
        <rl:step>
            texmfstart newpstopdf --convert --method=4
            --inputpath=<rl:value name='path'/>
            --outputpath=<rl:value name='path'/>/<rl:value name='prefix'/>
            <rl:value name='file'/>
        </rl:step>
    </rl:manipulator>

    <rl:manipulator name='medres' suffix='pdf'>
        <rl:old>
            <rl:value name='path'/>/<rl:value name='file'/>
        </rl:old>
        <rl:new>
            <rl:value name='path'/>/<rl:value name='prefix'/><rl:value name='file'/>
        </rl:new>
        <rl:step>
            texmfstart newpstopdf --convert  --method=4
            --inputpath=<rl:value name='path'/>
            --outputpath=<rl:value name='path'/>/<rl:value name='prefix'/>
            <rl:value name='file'/>
        </rl:step>
   </rl:manipulator>

</rl:manipulators>
```

After the first run of TEXEXEC, a resource log file is produced. It's this file that tells RLXTOOLS what files need to be manipulated.

```
<?xml version='1.0' standalone='yes'?>

<rl:library>
    <rl:usage>
        <rl:type>figure</rl:type>
        <rl:state>missing</rl:state>
        <rl:file>svg/example</rl:file>
        <rl:suffix>svg</rl:suffix>
        <rl:conversion>pdf</rl:conversion>
        <rl:width>115.73352pt</rl:width>
        <rl:height>86.80014pt</rl:height>
    </rl:usage>
</rl:library>
```

Depending on the request, there can be more info in this file.