The CSSSave package

Extending the built-in `HTMLSave` function with style sheets

Reinhold Kainhofer

Abstract

I present a package called CSSSave[1], which extends this internal `HTMLSave` function so that is uses Cascading Style Sheets (CSS) to let the converted HTML pages look exactly like the Mathematica notebooks. The style definitions from the Mathematica Style Sheet are converted into a .css file, which is then referenced by the actual HTML page and used by the web browser when viewing the HTML page. After this style conversion, the built-in `HTMLSave` function is used with several customized options to convert the notebook itself to HTML.

Using the CSSSave package, I will show how one can extend the built-in functionality of Mathematica in a very simple way for the developer. To the user the extended functionality is completely transparent, and runs without any noticeable difference (except for the better result) to the way it worked before installing the package.

1 Introduction

With the definition [4, 5] of Cascading Style Sheets (CSS in short), the markup language HTML - ubiquitous as the markup language of the WWW - has turned from a pure markup language for providing simple information into a tool for presenting that very information in a visually styled and consistent manner.
Mathematica [7] also provides similar features through its Style Sheets, which contain style definitions such a fonts, colors, spacings, etc. It also provides an export function (`HTMLSave`, available through the menu item "File - Save as Special... - HTML") to convert notebooks into HTML pages, however this export function does not make use of cascading style sheets.

Here, we will take a look at the CSSSave package [2] written by the author, where the built-in `HTMLSave` function is extended to provide such a conversion of the Mathematica styles in addition to the conversion of the pure notebook contents. To this end, we have to intercept the internal call to the HTML export function in Mathematica (section 2 for all HTML export functionality. Then we describe the style definitions both in Mathematica and in cascading style sheet, where we will see in section 3 that there exists a good correspondence between them. This fact is exploited in the `CSSSave` function, which is described in section 4. After the styles are converted, the built-in `HTMLSave` function can be used to convert the notebook contents (section 5). Additionally, the package adds some other nice new

[1]The code of this Mathematica package is licensed as free and open software under the GNU public license and can be downloaded freely from `http://csssave.sourceforge.net`.

features to Mathematica's HTML export, which are described in section 6. Finally, we will present an example of the high quality of the converted notebook using our CSSSave package and compare it to the results of the built-in `HTMLSave` function.

2 How to intercept the menu-item for HTML export

First, let us take a look at how Mathematica calls the HTML export function that is built into the application. In particular, one can call the menuitem "File -> Save As Special ... -> HTML" to convert the current notebook to HTML. Additionally, Mathematica provides the function `HTMLSave` [6] which takes the following arguments:

```
In[1]:= ?HTMLSave

 HTMLSave["file.html", notebook, options]
     converts the Notebook object notebook into an HTML document.
```

In fact, if we look at Mathematica's menu definition in the file `$TopDirectory/SystemFiles/FrontEnd/$InterfaceEnvironment/MenuSetup.tr`, we see that the HTML export is actually a call to the `HTMLSave` function:

```
Item["HTML",
  KernelExecute[ToExpression["FrontEnd‘DoHTMLSave[]"]], MenuEvaluator->Automatic],
Item["HTML+MathML",
  KernelExecute[ToExpression["HTMLSave[ Experimental‘FileBrowse[],  \
    InputNotebook[], ConversionOptions->{\"MathOutput\"->\"MathML\"} ]" ]],
  MenuEvaluator->Automatic],
```

`FrontEnd‘DoHTMLSave[]` is just a wrapper for the `HTMLSave` function. The `KernelExecute` function and the `MenuEvaluator->Automatic` option tell Mathematica not to interpret the second argument of `Item` as a FrontEndToken, but rather as code that is to be executed by the Kernel. If the user selects the menu item, the kernel is started (if it is not already running), and the expression is evaluated by the kernel.

Thus, in order to obtain our goal of intersecting the built-in HTML conversion with our own adapted function, all we have to do is to make sure that when Mathematica calls `HTMLSave` it does not use the built-in `HTMLSave` function, but an extended version provided by us. At first thought, providing our own conversion function instead of the built-in version might seem like an extremely laborious task. However, we do not have to replace the whole conversion functionality with our own code. All we need to do is to convert the styles from the Mathematica style sheet to a cascading style sheet, and then the notebook can be converted with the built-in `HTMLSave` function. Since Mathematica is rule-based, all function definitions are internally stored as rules of the form `functionname[args___]:>functioncode` and are sorted according to their generality. A function can have different definitions with different function arguments and other conditions in the pattern on the left-hand side of the `:>`. If Mathematica encounters an expression like

```
In[2]:= HTMLSave["/home/kainhofer/Notebook.html", InputNotebook[]]
```

it searches all rules defined for `HTMLSave` starting from the most specific pattern going to more general patterns, and uses the first one that matches the expression. Since the built-in `HTMLSave` function is defined in the form (see e.g. [8])

```
In[3]:= HTMLSave[destinPath : (_String|_FileName|_FrontEnd‘FileName), nb_Notebook,
          convOpts___?OptionQ] /; (Head[$FrontEnd] === FrontEndObject) := Block[...];
```

```
In[4]:= HTMLSave[destinPath : (_String|_FileName|_FrontEnd‘FileName), nb_Notebook,
           convOpts___?OptionQ]/; ($UseCSSInternal) /;
           ( Head[$FrontEnd] === FrontEndObject) :=
        Module[{res},
          (* Convert the styles to css*)
          CSSSave[destinPath, nb, convOpts];
          (* Prepare the arguments for the internal HTMLSave function *)
          Block[{$UseCSSInternal},
            res = HTMLSave[destinPath, nb, newConvOpts]
          ];
          (* Clean up if necessary *)
          res
        ];
```

Figure 1: General structure of the `HTMLSave` definition to intercept Mathematica's call to it.

we need to write our own definition of `HTMLSave`, which does not overwrite the existing one, and is slightly more specific so that it will be called instead of the built-in one. One way to achieve this is to use an additional condition `/; $UseCSSInternal` in the `HTMLSave` pattern, but leave it untouched otherwise. If we want to call the original `HTMLSave` function from within our own code, we need to make sure that Mathematica does not again use our own definition. This can be solved using a function structure including `Block[{varname=localvalue}, ...]` as shown in figure 1:

Once this definition is loaded by the kernel, every call to `HTMLSave` will go to our own code, no matter if it was called in the form of the menuitem, or by explicitly executing `HTMLSave`.

3 Overview of CSS and Mathematica's style sheets

In Mathematica the style definitions for a notebook opened in the FrontEnd are made in a separate notebook that contains one cell for each style (and environment), which stores all cell options like fonts, borders, spacing, evaluation options, etc. An example of this is the definition of the Section style in the PastelColor style sheet:

```
Cell[StyleData["Section"],
  CellFrame->{{0, 0}, {0, 0.25}},
  CellMargins->{{36, 20}, {10, 20}},
  CellGroupingRules->{"SectionGrouping", 30},
  PageBreakBelow->False,
  CellFrameMargins->{{10, 4}, {6, 2}},
  CounterIncrements->"Section",
  CounterAssignments->{{"Subsection", 0}, {"Subsubsection", 0}},
  FontSize->14,
  FontWeight->"Bold",
  FontColor->RGBColor[0.771908, 0.399634, 0.262867]]
```

Unless explicitly overridden in a cell, every cell of style "Section" will inherit these options.

CSS in turn, provides a similar structure to apply style definitions to the contents of HTML tags. If a HTML file links to a .css file via a tag

```
<link REL="StyleSheet" HREF="Own.css" TYPE="text/css" TITLE="Mathematica Styles">
```

in its `<head>...</head>` section, and the linked style sheet contains a definition like (corresponding to the above example of the Section style of Mathematica's PastelColor style sheet)

```
H3 {    /* "Section" style */
  border-color : rgb(0,0,0);
  border-style :  solid none none none;
  border-width :  1px 0px 0px 0px;
  margin :  20px 20px 10px 36px;
  padding :  2px 4px 6px 10px;
  font-size : 14px;
  font-weight : 700;
  color : rgb(77%, 40%, 26%);
}
```

then all third-level headings (`<h3>...</h3>`) in the HTML file will apply these styles to its contents. As one can see, there exists approximately a one-to-one correspondence between Mathematica's style options and the style settings in the sections of a cascading style sheet for HTML files. Tables 1 to 3 show these correspondences in detail, which are crucial for the conversion to CSS.

Mathematica	Cascading Style Sheet (CSS)

`Cell[ StyleData["Stylename"],`
`optionname->optionvalue,`
`...`
`]`
	`stylename, .style {`
	`(* Comment:  Definition for "style" *)`
	`propertyname:  propertyvalue;`
	`...`
	`}`

`Optionname->Value`	`propertyname:  "Value";`

Table 1: Corresponding structures of the Mathematica style sheets and cascading style sheets

Mathematica	Cascading Style Sheet (CSS)	*Mathematica*	Cascading Style Sheet (CSS)
"Notebook"	BODY, HTML	"Text"	P
"Title"	H1	"Input"	.Input
"Section"	H3	"Output"	.Output
...		"Graphics"	.Graphics

Table 2: Corresponding style names of the Mathematica style sheets and HTML with cascading style sheets. Styles like Input and Output are converted to `<P class="Input">...</P>` in HTML.

4 Reading and processing the Mathematica style sheet

Now that we know how the output of our conversion needs to look like, we can finally start writing a `CSSSave` function

Mathematica	Cascading Style Sheet (CSS)
Background->RGBColor[0, 1, 0]	background-color : rgb(0%, 100%, 0%);
FontColor->GrayLevel[1]	color : rgb(100%, 100%, 100%);
FontFamily->"Lucida"	font-family : "Lucida";
CellFrame->{{0, 0}, {0, 0.25}}	border-style : solid none none none;
	border-width : 1px 0px 0px 0px;
CellMargins->{{36, 20}, {10, 20}}	margin : 20px 20px 10px 36px;
CellFrameMargins->{{10, 4}, {6, 2}}	padding : 2px 4px 6px 10px;
TextAlignment->Left	text-align : Left;
FontVariations->{"Underline"->True}	text-decoration : underline;
CellDingbat->"\[FilledCircle]"	display: list-item; list-style-type: square;

Table 3: Some corresponding style attributes and properties.

```
In[5]:= CSSSave[ToFile_String, nb_Notebook, opts___Rule]
```

which takes the style definitions from the given notebook and generates a cascading style sheet form it. In this section, I will only outline the process of creating a cascading style sheet from a Mathematica style sheet and ignore the details. The interested reader is referred to the CSSSave code [2] for the exact implementation and the correspondence tables.

In Mathematica, the style definitions are stored inside the notebook expression either by a link to a standard Mathematica style sheet (e.g. `StyleDefinitions->"Report.nb"`), or the whole stylesheet is included as a notebook expression, i.e. the StyleDefinitions option of the notebook looks like

```
StyleDefinitions->Notebook[{
 Cell[CellGroupData[{Cell["Style Definitions","Subtitle"],
   ...,
   Cell[StyleData["Title"],CellFrame->{{0,0},{0,0.25}}, FontSize->36, ...],
   Cell[StyleData["Subtitle"],LineSpacing->{1,0}, FontSize->24, ...],
   ...
 }], Open]
}]
```

Either way, given the notebook expression (e.g. by calling `NotebookGet`) of the notebook scheduled for conversion to HTML, it is easy to get the notebook expression of the Mathematica style sheet it uses. There, only the cells that match the pattern `Cell[StyleData[_String, ___String], opts___?OptionQ]` are of interest to us, because only they contain style definitions. Thus, using `Cases`, we can extract all style definitions and then apply a function to this list of cells that converts a StyleData cell to a string containing the css representation of that style settings as outlined in the previous section.

There are, however, a few pitfalls, which make the conversion process not so straight-forward as to just map a function on all options and write the resulting string out to a file. In particular, the reasons for this are

- HTML and CSS know several different types of tags [1], block-level elements, container elements (`<DIV>`) and other elements. Block-level elements like `<P>`, `<PRE>`, `<UL>`, or `<H1>` define paragraph-like structures, which must not be nested. In Mathematica, however, one can always have e.g. one sentence of style "Section" inside a text cell. So, to convert such a cell, we need to provide both a block-level element `<ht>` and another non-block-level element (e.g. `<div style="Section">`) in the cascading style sheet with the necessary style attributes. Additionally, non-block-elements (and non-container elements) in the css file cannot have block-level attributes like `border` or `text-align`.

- Several Mathematica options like `FontVariations` have suboptions, which need to be converted as separate CSS properties.
- Several options (like `CellFrame` and `CellFrameColor`) depend on each other. If only one of these options is available in the Mathematica style definition, in the CSS file we need to provide sensitive defaults (e.g. if no CellFrameColor option is given, the border-color attributes should default to black, while it needs to give the color of CellFrameColor if that option is specified in the Mathematica style sheet).
- Mathematica has two options that control how text in a paragraph is aligned, `TextAlignment` and `TextJustification`, while CSS defines only `text-align`, so the two Mathematica options need to be combined.
- Some Mathematica options apply only to special tags, e.g. `GridBoxOptions` apply only to tags like `<TABLE>`, `<TR>`, etc.

Nevertheless, the `CSSSave` function is able to create a sensitive CSS file that can be linked to by the HTML page of the converted notebook.

5 Preparing defaults for `HTMLSave`

The only thing that remains to be done is to actually convert the notebook to HTML. Since Mathematica already provides a good and very customizable [6] way to convert the notebook to HTML via its `HTMLSave` function, there is no need to write our own conversion function for HTML. Instead we will call `HTMLSave` (see figure 1) which converts the notebook for us. However, to fully exhaust the potential of `HTMLSave` and our function `CSSSave`, we need to call it with the appropriate option values. In particular, `HTMLSave` has the option `ConversionOptions` with the following suboptions that are of interest in our case:

- `"MarkupRules"` ... The value of this option is a list of rules, where each entry gives the rules for the conversion of a specific style. For example, the default conversion rules for the styles "Title" and "Text" are as follows:

  ```
  "Title" -> {
    {"<strong><big><span class=\"Title\">", "</span></big></strong>"},
    {"<h1 class=\"Title\">", "</h1>"}},
  "Text" -> {{"<span class=\"Text\">", "</span>"},
    {"<p class=\"Text\">", "</p>"}}
  ```

  The first sublist of the right hand side of each rule gives the tags before and after each element with that style, if the element is an inline element (e.g. a StyleBox of style "Title" within a cell of a different style). The second sublist, in turn, gives the tags that are used if the element to be converted is a block-level element in Mathematica. In particular, this is the case for cells of that style. The distinction between these two cases is necessary since block-level elements in HTML are always paragraphs, i.e. they have a linebreak before and after their contents. So, inline elements cannot be converted the same way as whole cells.

- `"HeadElements"` ... The `HTMLSave` function inserts the value of this option in the HTML page in the `<HEAD>....</HEAD>` section. This is where we can and need to in insert the link to the css file: `<link REL="StyleSheet" HREF="Notebook.css" TYPE="text/css" TITLE="Mathematica Styles">`.

- `"MathOutput"` ... This option controls how complex typeset mathematical box structures (such as fractions, subscripts, superscripts, roots, etc) are converted. By default, the option value is set to "GIF", which causes almost all input and output to be converted to gif graphics. While the converted structure then looks exactly like the corresponding part in Mathematica, one cannot further use the code displayed in the browser. Instead, we would like to convert most of the input to a pure text representation, so that Mathematica code can be copied out of a converted HTML page and immediately evaluated in Mathematica. This is possible by setting the value of `"MathOutput"` to `InputForm`.

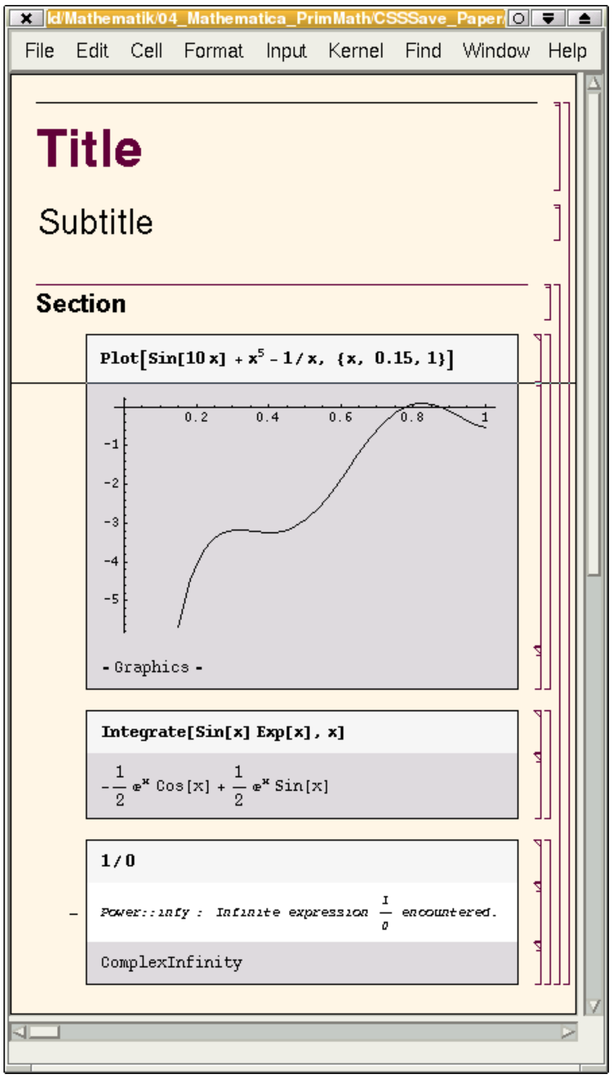
When walking through all styles in the Mathematica style sheet, the `CSSSave` function automatically generates the appropriate MarkupRules for all styles it encounters in the style sheet. It returns these rules so that our implementation of `HTMLSave` can just pass them on to the built-in `HTMLSave` function.

6 Adapting `HTMLSave` to our needs

The discussion in the previous sections showed how one can convert the Mathematica styles to a cascading style sheet and let `HTMLSave` use these styles. However, there are several cases, where this simple procedure is not sufficient to generate acceptable output. For example, if style settings (like fonts, frames, borders, colors, etc.) are not done in the style sheet, but are explicitly set for a cell or a Box expression in the Mathematica-notebook, `HTMLSave` does not take advantage of the additional layout features provided by CSS. Thus, we have to provide it with an extended set of rules, which describe how many of the Cell and Text options in Mathematica can be converted to HTML and include CSS tags. The built-in `HTMLSave` function uses `ConvertFormatRule[a, inline]` to format any part of the text. Like with `HTMLSave`, we override the built-in behavior with our own function, which calls the built-in one and then inserts the appropriate CSS code into the HTML tags according to the options of `a`. This way, it is possible, to convert e.g. a `CellMargins->{{10, 50}, {20, 70}}` option set for a Title cell into `<H1 style="margin:  70px 50px 20px 10px;">`, where the `<H1>` tag comes from `HTMLSave`, and the `style` option is inserted by the CSSSave package.

Similarly, not all options can be in the cascading style sheet. For example, the CellFrameLabels option in Mathematica does not have a corresponding attribute in CSS, and thus a CellFrameLabels setting for a style would be completely ignored during the conversion. The CSSSave package, however, adds an additional option `CopySSOptionsToNB->{CellFrameLabels, CellDingbat}` for `HTMLSave`. If a style contains any of the options listed there, the option and its value will be copied directly into the cell before the conversion takes place. This has the effect that the procedure outlined in the previous paragraph will take care of that option and convert e.g. CellFrameLabels into a table.

Another nice feature that `HTMLSave` misses is an option to delete certain cells before the conversion is done. When preparing some Mathematica notebook for the web, one might insert some cells with notes for oneself, which should not be converted and included in the final web page. The CSSSave package adds such an option `DeleteCells->{}`, where the list can contain either patterns (like `Cell[s_/;MemberQ[s, "TODO", Infinity],___]` or `_ButtonBox`), where every part of the notebook that matches any of the patterns is deleted prior to the conversion. Alternatively, one can also give strings in the list, which are interpreted as patterns for cells of this style. Additionally, a `PreConvertReplacements` option is added, which can contain several replacement rules that will be applied to the notebooks expression using `ReplaceAll`.



Title

Subtitle

Section

```
Plot[Sin[10 x] + x^5 - 1/x, {x, 0.15, 1}]
```

```
- Graphics -
```

```
Integrate[Sin[x] Exp[x], x]
```

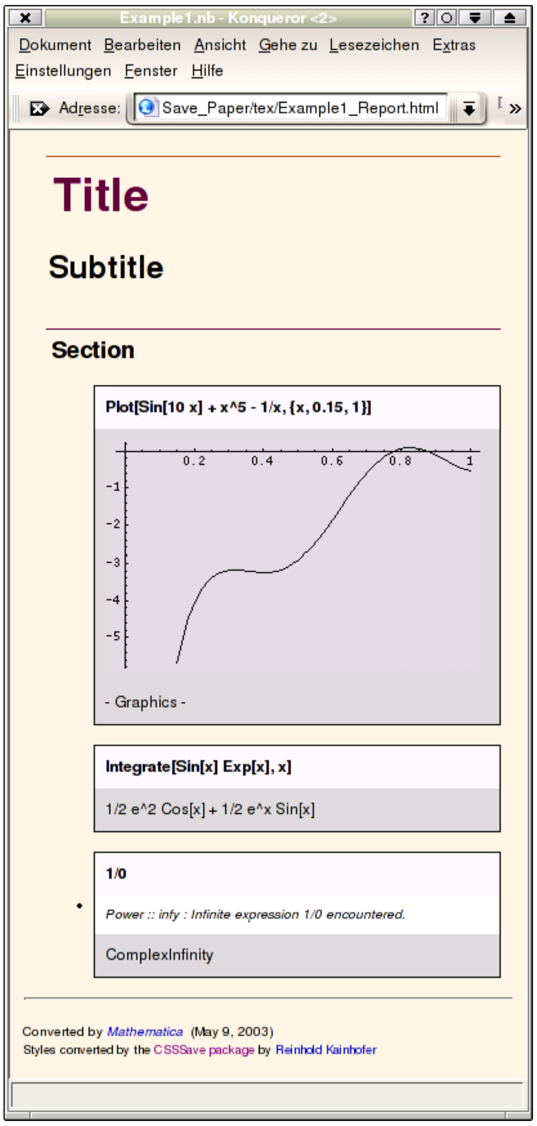
$$-\frac{1}{2} e^x \operatorname{Cos}[x] + \frac{1}{2} e^x \operatorname{Sin}[x]$$

```
1/0
```

```
Power::infy : Infinite expression 1/0 encountered.
```

```
ComplexInfinity
```

Figure 2: Notebook in Mathematica



Title

Subtitle

Section

```
Plot[Sin[10 x] + x^5 - 1/x, {x, 0.15, 1}]
```

```
- Graphics -
```

```
Integrate[Sin[x] Exp[x], x]
```

```
1/2 e^2 Cos[x] + 1/2 e^x Sin[x]
```

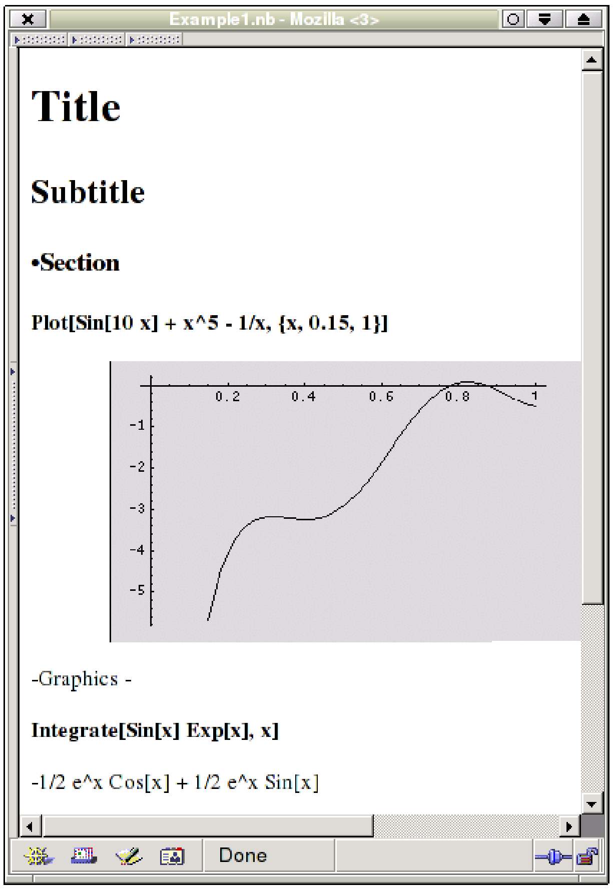
```
1/0
```

```
Power :: infy : Infinite expression 1/0 encountered.
```

```
ComplexInfinity
```

Converted by *Mathematica* (May 9, 2003)
Styles converted by the CSSSave package by Reinhold Kainhofer

Figure 3: The same notebook in HTML



Title

Subtitle

•Section

Plot[Sin[10 x] + x^5 - 1/x, {x, 0.15, 1}]

-Graphics -

Integrate[Sin[x] Exp[x], x]

-1/2 e^x Cos[x] + 1/2 e^x Sin[x]

Figure 4: The same notebook converted without CSSSave, using "MathOutput" -> InputForm. All expressions are converted as text and can be copied. The graphical layout of the notebook is lost.

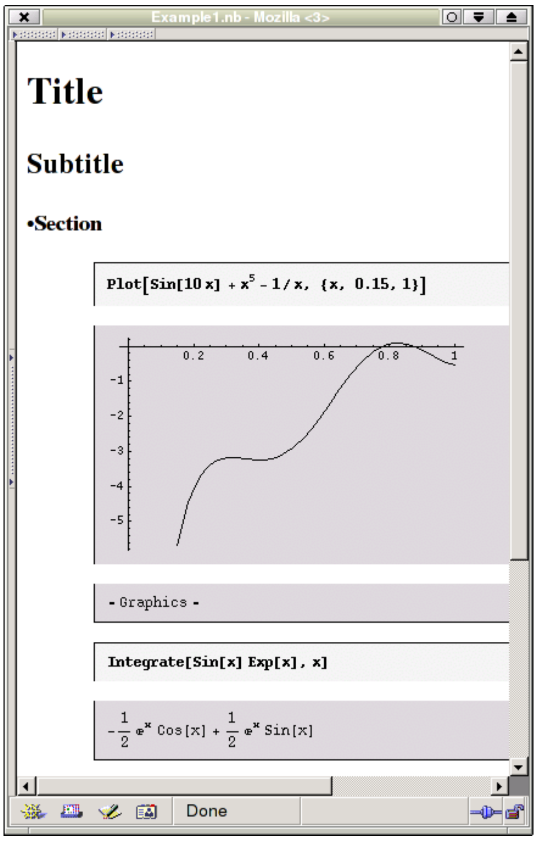

Example1.nb - Mozilla <3>
Title
Subtitle
•Section
Plot[Sin[10 x] + x^5 - 1/x, {x, 0.15, 1}]
- Graphics -
Integrate[Sin[x] Exp[x], x]
$-\frac{1}{2} e^x \text{Cos}[x] + \frac{1}{2} e^x \text{Sin}[x]$
Done

Figure 5: The notebook converted using the plain `HTMLSave` function (using the default "MathOutput"->"GIF"). Copying the text is no longer possible.

Last, but not least, `HTMLSave` is provided with the new options `MainPageLinks->True` as well as `MainPageLinkText->{"<center><p>Back to ","Main Page","...</p></center>"}`, which cause a link to the main HTML page of the conversion to be added at the bottom of each subpage (closed subgroups can be converted to separate HTML pages, which are linked from the visible first cell of the closed group).

7 Results of the CSSSave conversion

Finally, after the discussion of the techniques and programming tricks used in the CSSSave package, let us take a look at the results produced by the HTML export when the CSSSave package is loaded and active. Figure 2 shows a sample notebook in Mathematica using the "Report.nb" stylesheet that is provided with Mathematica. If we convert this notebook to HTML using the CSSSave package, the resulting HTML page looks almost exactly the same in the web browser (figure 3) as it does in Mathematica. Additionally, the mathematical expressions in the HTML page are not graphics, but pure text so that the person viewing the HTML page can easily copy the code to a Mathematica notebook and immediately evaluate it there.

In contrast, figures 4 and 5 show the same notebook converted with the pure built-in `HTMLSave` function as provided by Mathematica (without the CSSSave package). While figure 4 uses the ConversionOption `"MathOutput"->InputForm`, so that parts of the code can be selected in the HTML page and copied, it completely looses all graphical layout of the notebook. Figure 5 cures this to some extent by using the default ConversionOption `"MathOutput"->"GIF"`, but at the expense that every typeset expression in the notebook is converted to a gif graphics, so that copying is not possible.

8 Conclusion

In this paper and the CSSSave package we showed how the built-in HTML export function can be extended to use CSS, so that the resulting HTML page completely retains the layout and the design of the original notebook. This was possible due to the high customizability of the built-in `HTMLSave` function, which does not display its full potential with the default values set, but also due to the fact that in Mathematica one can access almost every aspect of the FrontEnd or the kernel using Mathematica's own powerful programming language. This allows us to intercept the call to the HTML export function with our own extended version, which in turns converts the styles, but then calls the built-in function for the actual HTML conversion. To the user, the package works completely transparent, so except for the better result, the user will not see any difference in the HTML export function when the CSSSave package is loaded or not.

References

[1] D. Goodman. *Dynamic HTML: The Definitive Reference*. O'Reilly & Associates, 1998.

[2] R. Kainhofer. CSSSave package. Homepage: `http://csssave.sourceforge.net/`.

[3] R. Maeder. *Programming in Mathematica*. Addison Wesley Longman, Inc., 3rd edition, 1997.

[4] W3 Consortium. Cascading Style Sheets, level 1. Published online: `http://www.w3.org/TR/REC-CSS1`, 17 Dec 1996, revised 11 Jan 1999.

[5] W3 Consortium. Cascading Style Sheets, level 2, CSS2 Specification. Published online: `http://www.w3.org/TR/REC-CSS2`, May 1998.

[6] S. Wolfram. *HTMLSave, Mathematicareference guide*. Wolfram Researcht, Inc. Available online: `http://documents.wolfram.com/v4/RefGuide/HTMLSave.html`.

[7] S. Wolfram. *The Mathematica Book*. Wolfram Media, Champaign, IL, USA and Cambridge University Press, Cambridge, UK, fourth edition, 1999.

[8] Wolfram Research, Inc. Todo: Html and latex conversion code. Available for download at `http://support.wolfram.com/mathematica/interface/export/`.

Reinhold Kainhofer (email: `reinhold@kainhofer.com`)

Graz University of Technology, Department of mathematics, Steyrergasse 30, A-8010 Graz, *and*
Deltasoft mathematics, Lead developer, Elisabethstraße 42, A-8010 Graz