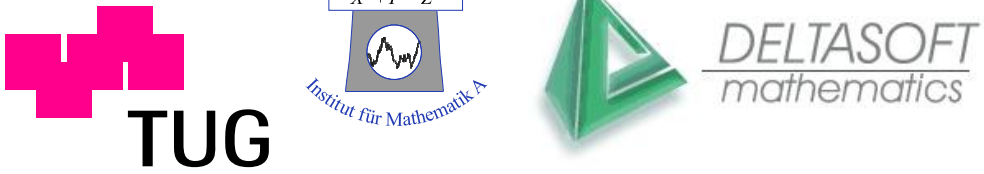
Reinhold Kainhofer

The CSSSave‘ Package for Mathematica

Extending the built-in `HTMLSave` function with (cascading) style sheets



Graz University of Technology
Deltasoft mathematics

Zagreb, September 26, 2003

Motivation

- Style sheets give Mathematica a stylish look
- But converted HTML pages look boring (black text on white bg., no colors, borders, spacing, layout)
  Only some (more or less advanced) HTML markup, hardcoded in Mathematica's Style Sheet
- HTML supports style sheets as well: Cascading Style Sheets (CSS) as defined by the W3C

Problem:
Try to extend the existing `HTMLSave` function of Mathematica
No duplication, expansion of the function

Solution: **CSSSave` Package** (http://csssave.sf.net)

How to work with the package

- Loading: `<<CSSSave`` `
- Converting to html: HTMLSave will automatically use the package
- Generating a Cascading style sheet from a notebook: CSSSave function (similar to HTMLSave)
  `CSSSave[toFile_String, nb_Notebook, opts___?OptionQ]`
- In Mathematic 5.0 this functionality is already included. The extra package is not needed.

Style Sheets

- Mathematica uses it's own style sheets in Mathematica syntax, no inheritance
- HTML uses Cascading Style Sheets, even inheritance is possible

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;

How to intercept the HTML export (menu item)

- Menuitem calls `FrontEnd`DoHTMLSave[]`, which is a wrapper for `HTMLSave`
- ⇒ Write our own HTMLSave function, force Mathematica to use that one
- This can be done using a more specific pattern with a condition:

```
$UseCSSInternal = True;
HTMLSave[ destinPath:(_String|...), ...,
   convOpts___?OptionQ] /; $UseCSSInternal :=
Block[{$UseCSSInternal=False},
  (* Here comes our own conversion code.
  We can even call HTMLSave here, and the internal
  one will be called *)
]
```

General structure of our own HTMLSave function

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

Converting the style sheets

- Extract all style cells by **Cases**, then generate a string from each.
- Convert each `Cell[StyleData[style_String], opts___?OptionQ]` to the form

```
stylename, .style {
  propertyname: propertyvalue;
  ...
}
```

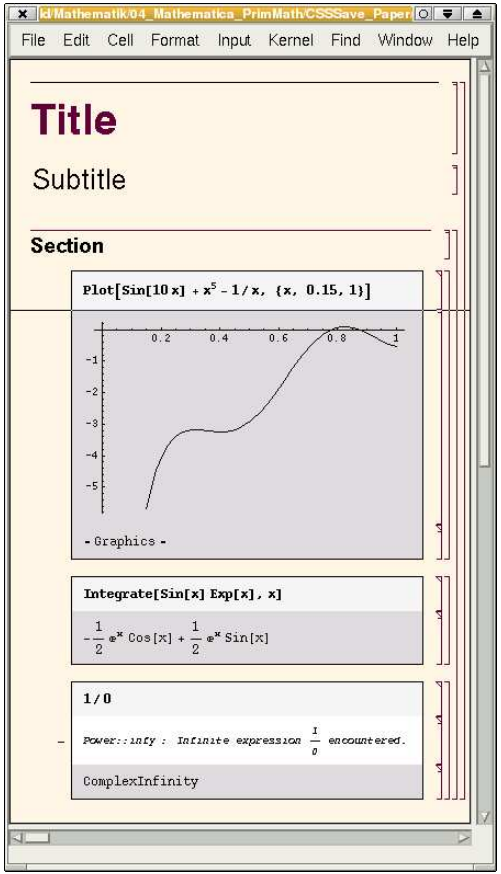
Generate arguments for HTMLSave

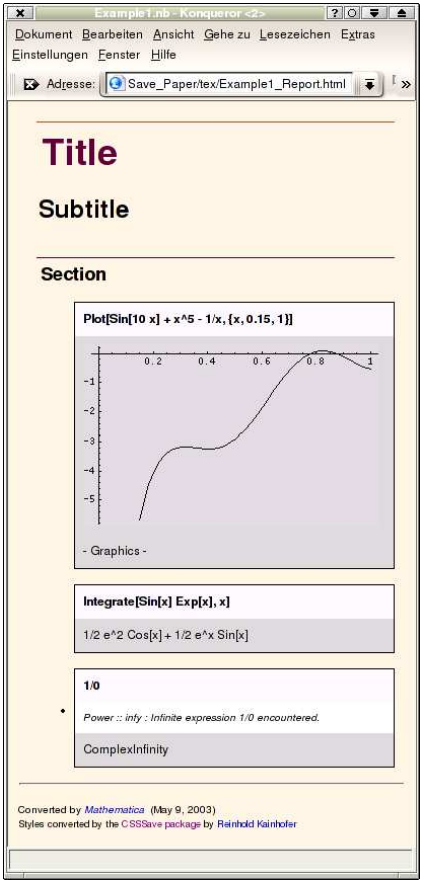
- Use the **MarkupRules** option of HTMLSave. Gives conversion rules for each style.
- Need rules for inline text and block level elements, e.g.:

```
"Text" -> {{"<span class=\"Text\">", "</span>"},
           {"<p class=\"Text\">", "</p>"}}
```

The rest is done by the internal HTMLSave

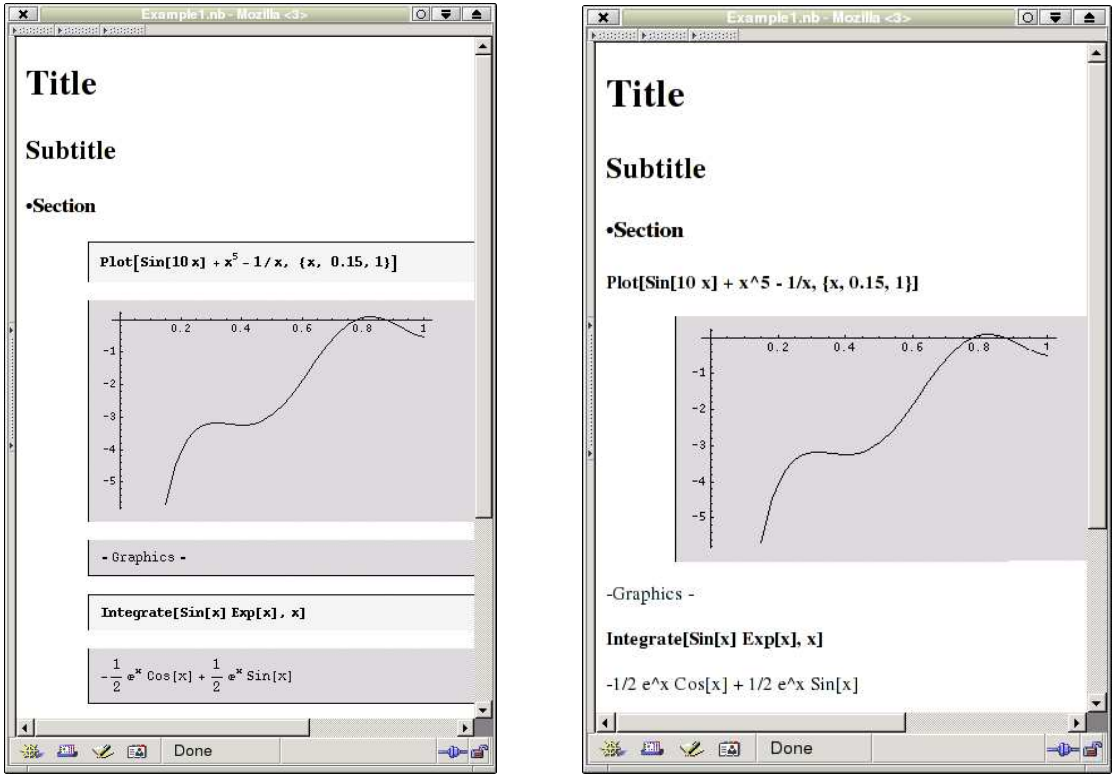
Results: Using the CSSSave package





The same notebook in Mathematica and converted HTML using the CSSSave package.

Results: The old HTMLSave results



Conversion result with the built-in HTMLSave.

Either only graphics are created (no copying), or all formatting is lost.