Exploring Mathematica

This pages points to ways of exploring Mathematica's implementation of Wolfram Language.

To access the definition of a built-in function, simply use `PrintDefinitions` from the `GeneralUtilities` package. (You don't need the Spelunking Tools anymore.)

```
In[1]:=
GeneralUtilities`PrintDefinitions[GeneralUtilities`Pri


Out[1]=
NotebookObject[GeneralUtilities`PrintDefinitions]
```

Parsing Expressions

The FE creates a box representation of your input as you type. The box representation then gets reinterpreted as an expression by the kernel. The FE often interprets input differently from the kernel. In particular, there are many cases of inconsistencies in operator precedence between the FE and `ToExpression` / command line. Therefore, we need ways of explicitly choosing which component parses an expression even if we are working in a notebook.

Using the kernel to parse an expression

Using `ToExpression` or the command line interface uses the kernel directly to parse the expression, circumventing the FE's parser. To make sure your expression is parsed by the kernel even if you are working in a notebook, do this:

```
In[10]:= FullForm[ToExpression["Hold[a|b*c]"]]

Out[10]//FullForm= Hold[Alternatives[a,Times[b,c]]]
```

It matters how you use `ToExpression`. Compare:

```
In[1]:= ToExpression["a;;\[Intersection]a\
[SquareIntersection];;a", StandardForm,
Hold]//FullForm

Syntax::sntxf: "a;; ∩ a ⊓" cannot be followed by
";;a".

ToExpression::sntx:
   Invalid syntax in or before "\!\
(StandardForm\`a;;\:22c2a\:2293;;a\) ".

^
Out[1]//FullForm= $Failed
```

gives a different result from

```
In[1]:= ToExpression["Hold[a;;\[Intersection]a\
[SquareIntersection];;a]"]//FullForm

Out[1]//FullForm=
Hold[Span[SquareIntersection[Intersection[Span[a,
All], a], System`Private`DummyId], a]]
```

in both the FE and command line.

Using the Frontend to parse an expression

To ask the FE to interpret an expression given as a `String`, do this:

```
FEToExpression[s_String] :=

MakeExpression@FrontEndExecute@FrontEnd`ReparseBoxStru
```

Alternatively, you can use the `UndocumentedTestFEParserPacket` function:

```
FEToExpression[s_String] := MakeExpression[
    MathLink`CallFrontEnd[
        FrontEnd`UndocumentedTestFEParserPacket[s,
False]
        ][[1]] (* Do not need form annotation. *)
    ]
```

Note that `UndocumentedTestFEParserPacket` returns a list of the form `{boxexpression, form}`, where `form` is usually `StandardForm`.