The Arlington PDF Model

This is a DaeDaLus speicfication of the grammar for the Arlington PDF Model. The most recent developments are available from Github:

`https://github.com/pdf-association/arlington-pdf-model`

The model describes a collection of datatypes, each in a separate file. Each file

```
def Main =
  block
    Many $[! $recordTerminator]; $recordTerminator -- Skip header
    $$ = Many Field
    END

def $fieldSeparator   = '\t'
def $recordTerminator = '\n'

def Field =
  block
    key               = FreeText;                 $fieldSeparator
    type              = FieldType;                $fieldSeparator
    sinceVersion      = Version;                  $fieldSeparator
    deprecatedIn      = Optional Version;         $fieldSeparator
    required          = IsRequired;               $fieldSeparator
    indirectReference = Alts IsIndirect;          $fieldSeparator
    inheritable       = IsInheritable;            $fieldSeparator
    defaultValue      = Alts DefaultValue;        $fieldSeparator
    possibleValues    = MultiAlts PossibleValue;  $fieldSeparator
    specialCase       = MultiAlts SpecialCase;    $fieldSeparator
    link              = MultiAlts Link;           $fieldSeparator
    note              = FreeText;                 $recordTerminator

def FreeText  = Many $[! ($fieldSeparator | $recordTerminator)]

def Bracketed P =
  block
    KW "["
    $$ = P
    KW "]"

def Alts P =
  First
    SepBy (KW ";") (Bracketed (Optional P)) -- must be first
    [ Optional P ]

def MultiAlts P =
  Optional (SepBy (KW ";") (Bracketed (Optional (SepBy (KW ",") P))))
```

Field Type

```
def FieldType = SepBy (KW ";") FieldTypeExpression

def FieldTypeExpression =
  First
    TType       = PrimitiveType
    TSince      = FnSinceVersion PrimitiveType
    TDeprecated = FnDeprecated PrimitiveType

def PrimitiveType =
  First
    TArray       = @Match "array"
    TBitmask     = @Match "bitmask"
    TBoolean     = @Match "boolean"
    TDate        = @Match "date"
    TDictionary  = @Match "dictionary"
    TInteger     = @Match "integer"
    TMap         = @Match "matrix"
    TNameTree    = @Match "name-tree"
    TName        = @Match "name"
    TNull        = @Match "null"
    TNumberTree  = @Match "number-tree"
    TNumber      = @Match "number"
    TRectangle   = @Match "rectangle"
    TStream      = @Match "stream"
    TStringASCII = @Match "string-ascii"
    TStringByte  = @Match "string-byte"
    TStringText  = @Match "string-text"
    TString      = @Match "string"
```

Required Fields

```
def IsRequired : BoolExpr =
  First
    Fun1 "IsRequired" BoolExpr
    BoolExpr
```

Direct Fields

This encode 3 possible values:

- direct-only
- indirect-only
- either

```
def IsIndirect =
  First
    IndirectIf = BoolExpr
    DirectIf   = {| TRUE = Fun0 "MustBeDirect" |}
    DirectIf   = Fun1 "MustBeDirect" BoolExpr
    IndirectIf = Fun1 "MustBeIndirect" BoolExpr
    IndirectIf = {| TRUE = KW "IndirectReference" |}
```

Inheritable Fields

```
def IsInheritable : BoolExpr =
  First
    {| TRUE = KW "Inheritable" |}
    BoolExpr
```

Default Values

```
def DefaultValue =
  First
    ImplementationDependent = Fun0 "ImplementationDependent"
    Conditional             = ConditionalDefaultCases
    Value                   = Term

def ConditionalDefaultCases =
  First
    FnEval (SepBy (KW "||") ConditionalDefault)
    [ ConditionalDefault ]

def ConditionalDefault =
  First

    block
      let f = Fun2 "IsPresent" FieldName Term
      condition = {| IsPresent = f.arg1 |} : BoolExpr
      value     = f.arg2

    block
      let f = Fun2 "DefaultValue" BoolExpr Term
      condition = f.arg1
      value     = f.arg2
```

Possible Values

```
def PossibleValue =
  First
    Deprecated    = FnDeprecated PossibleValue
    SinceVersion  = FnSinceVersion PossibleValue
    BeforeVersion = FnBeforeVersion PossibleValue
    Conditional   = ConditionalValue
    Eval          = FnEval (SepBy (KW "&&") PossibleValue)
    Constraint    = BoolExpr
    Value         = Term
    Wild          = KW "*"

def ConditionalValue =
  block
    let f = Fun2 "RequiredValue" BoolExpr Term
    condition = f.arg1
    value     = f.arg2
```

Special Checks

```
def SpecialCase =
  First
    SinceVersion  = FnSinceVersion  SpecialCase
    BeforeVersion = FnBeforeVersion SpecialCase
    AtVersion     = FnIsPDFVersion  SpecialCase
    Eval          = FnEval (SepBy (KW "&&") SpecialCase)
    Ignore0       = Fun0 "Ignore"
    Ignore        = Fun1 "Ignore" BoolExpr
    IgnoreF       = Fun1 "Ignore" FieldName
    Meaningful    = Fun1 "IsMeaningful" BoolExpr
    NoCycle       = Fun0 "NoCycle"
    NotPresentIf  = Fun1 "Not" (Fun1 "IsPresent" BoolExpr)
    NotRequiredIf = Fun1 "Not" (Fun1 "IsRequired" BoolExpr)
    MustbeDirect  = Fun1 "MustBeDirect" FieldName
    Constraint    = BoolExpr
```

Links

```
def Link =
  First
    Deprecated    = FnDeprecated Link
    SinceVersion  = FnSinceVersion TypeName
    InVersion     = FnIsPDFVersion TypeName
    Link          = TypeName

def TypeName  = Many (1..) $[ $alpha, $digit, '_' ]
```

Boolean Expressions

```
def BoolExpr =
  block
    First
      {| OR  = SepBy2 (KW "||") BoolAtomExpr |}
      {| AND = SepBy2 (KW "&&") BoolAtomExpr |}
      BoolAtomExpr

def BoolAtomExpr : BoolExpr =
  First

    block
      KW "("
      $$ = BoolExpr
      KW ")"

    {| TRUE                       = KW "TRUE"  |}
    {| FALSE                      = KW "FALSE" |}
    {| NOT                        = Fun1 "Not" BoolExpr |}

    {| FontHasLatinChars          = Fun0 "FontHasLatinChars" |}
    {| NotStandard14Font          = Fun0 "NotStandard14Font" |}
    {| KeyNameIsColorant          = Fun0 "KeyNameIsColorant" |}

    {| IsAssociatedFile           = Fun0 "IsAssociatedFile" |}
    {| IsPDFTagged                = Fun0 "IsPDFTagged" |}
    {| IsEncryptedWrapper         = Fun0 "IsEncryptedWrapper" |}
    {| PageContainsStructContentItems = Fun0 "PageContainsStructContentItems" |}
    {| ImageIsStructContentItem   = Fun0 "ImageIsStructContentItem" |}

    {| IsPresent                  = Fun1 "IsPresent" FieldName |}

    {| InMap                      = Fun1 "InMap" FieldName |}
    {| Contains                   = Fun2 "Contains" Term Term |}
    {| ArraySortAscending         = Fun2 "ArraySortAscending" Term Term |}

    {| SinceVersion               = FnSinceVersion BoolExpr |}
    {| BeforeVersion              = FnBeforeVersion BoolExpr |}
    {| AtVersion                  = FnIsPDFVersion BoolExpr |}

    {| IsAtLeastVersion           = Fun1 "SinceVersion" Version |}
    {| IsBeforeVersion            = Fun1 "BeforeVersion" Version |}
    {| IsPDFVersion               = Fun1 "IsPDFVersion" Version |}

    {| BitClear                   = Fun1 "BitClear" Term |}
    {| BitsClear                  = Fun2 "BitsClear" Term Term |}
    {| BitSet                     = Fun1 "BitSet" Term |}
    {| BitsSet                    = Fun2 "BitsSet" Term Term |}
    {| IsLastInNumberFormatArray  = Fun1 "IsLastInNumberFormatArray" Term |}

    {| EQ                         = BinOp "==" Term |}
    {| NEQ                        = BinOp "!=" Term |}
    {| LT                         = BinOp "<"  Term |}
    {| GT                         = BinOp ">"  Term |}
    {| LEQ                        = BinOp "<=" Term |}
    {| GEQ                        = BinOp ">=" Term |}
```

Values

```
def Term =
  First
    {| add = BinOp "+" TermProduct |}
    {| sub = BinOp "- " TermProduct |}
        -- the space is to avoid conflict with names like a-b
    TermProduct

def TermProduct : Term =
  First
    {| mul = BinOp "*"   TermAtom |}
    {| mod = BinOp "mod" TermAtom |}
    TermAtom

def TermAtom : Term =
  First
    block
      KW "("
      $$ = Term
      KW ")"

    {| ValueOf           = ValueOf |}

    {| Float             = FloatValue |}
    {| Integer           = IntegerValue |}    -- Must be after Float
    {| String            = StringValue |}
    {| Bool              = BoolValue |}
    {| Null              = NullValue |}
    {| Array             = ArrayValue |}

    {| RectWidth         = Fun1 "RectWidth"  Term |}
    {| RectHeight        = Fun1 "RectHeight" Term |}
    {| FileSize          = Fun0 "FileSize" |}
    {| ArrayLengthField  = Fun1 "ArrayLength" FieldName |}  -- ?
    {| ArrayLength       = Fun1 "ArrayLength" Term |}
    {| PageProperty      = Fun2 "PageProperty" Term FieldName |}
    {| StringLength      = Fun1 "StringLength" Term |}
    {| StreamLength      = Fun1 "StreamLength" Term |}
    {| NumberOfPages     = Fun0 "NumberOfPages" |}

    {| Name              = NameValue |}
    -- Needs to be after the functions, float, integer, bool, null


def Natural =
  block
    $$ = many (s = Digit) (10 * s + Digit)
    -- can't be followed by a letter
    case Optional $alpha of
      just    -> Fail "Expected number, found identifier"
      nothing -> Accept

def IntegerValue =
  Token
    First
      { $['-']; - Natural }
      Natural

{- Leave as text for now.  We could parse this as double
but some of the literals (e.g. 1.2) are not exactly representable,
so they might print funny, although likely not due to rounding. -}
def FloatValue =
  Token
    block
      whole = Many (1..) Digit
      $['.']
      frac  = Many (1..) Digit

def StringValue =
  block
    $['\'']
    $$ = Many $[!'\'']
    KW "'"

def BoolValue =
  First
    { KW "true";  true }
    { KW "false"; false }

def NullValue = KW "null"
def ArrayValue =
  block
    KW "["
    $$ = Many Term
    KW "]"

def NameValue = Token (Many (1 .. ) $[ $alpha, $digit, '.', '_', '-'])

def Version =
  block
    major   = Natural
    $['.']
    minor   = Natural
```

Field Names

```
def SimpleFieldName =
  First
    Text   = NameValue
    Wild   = $['*']

def FieldName = SepBy (Match "::") SimpleFieldName

def ValueOf =
  block
    qualifier = Optional { $$ = FieldName; Match "::" }
    $['@']
    field  = SimpleFieldName
```

Functions

```
def Fun0 f =
  block
    Match "fn:"
    Match f
    KW "("
    KW ")"
    Accept

def Fun1 f Arg =
  block
    Match "fn:"
    Match f
    KW "("
    $$ = Arg
    KW ")"
```

```
def Fun2 f Arg1 Arg2 =
  block
    Match "fn:"
    Match f
    KW "("
    arg1 = Arg1
    KW ","
    arg2 = Arg2
    KW ")"

def versioned (f : Fun2)  = { version = f.arg1, value = f.arg2 }

def FnSinceVersion Arg    = versioned (Fun2 "SinceVersion"  Version Arg)
def FnBeforeVersion Arg   = versioned (Fun2 "BeforeVersion" Version Arg)
def FnDeprecated Arg      = versioned (Fun2 "Deprecated"    Version Arg)
def FnIsPDFVersion Arg    = versioned (Fun2 "IsPDFVersion"  Version Arg)

def FnEval Arg            = Fun1 "Eval" Arg
```

Lexical and Utilities

```
def $alpha            = 'a' .. 'z' | 'A' .. 'Z'
def $digit            = '0' .. '9'

def Digit             = $digit - '0' as int

-- P followed by some optional space
def Token P =
  block
    $$ = P
    Many $[' ']

-- Match this string, followed by optional space
def KW x = Token (@Match x)

-- One or more Q, separated by P
def SepBy Sep Thing =
  build (many (s = emit builder Thing) { Sep; emit s Thing })

-- Two or more Q, separated by P
def SepBy2 Sep Thing =
  block
    let first  = emit builder Thing
    let second = emit first { Sep; Thing }
    let rest   = many (s = second) (emit s { Sep; Thing })
    build rest

-- | Binary infix operator
def BinOp op P =
  block
    lhs = P
    KW op
    rhs = P
```