UDC 35.060
L 78

中华人民共和国国家标准

GB/T 16648—1996

信息技术　文本通信
标准页面描述语言(SPDL)

Information technology—Text communication
—Standard of Page Description Language(SPDL)

1996-12-17发布　　1997-07-01实施

国家技术监督局 发布

目　　次

前　言

本标准与国际标准草案 ISO/IEC DIS 10180《信息技术——文本通信——标准页面描述语言(SPDL)》在技术上是一致的。

本标准为适合中文处理，增加了“16.2.1.1.1　FontType 4 字库词典”条。

通过制定这项国家标准，有利于我国出版印刷业中标准页面描述语的推广、使用。

GB/T 16648《信息技术　文本通信　标准页面描述语言(SPDL)》，目前包括 26 章(详见目次)。

本标准的附录 A、B、C、D 和 E 都是标准的附录。

本标准由中华人民共和国电子工业部提出。

本标准由电子工业部标准化研究所归口。

本标准起草单位：南京大学。

本标准主要起草人：徐福培、高健。

ISO/IEC 前言

ISO(国际标准化组织)是由各个国家标准机构(ISO 的成员体)联合组成的一个世界性组织。该组织通过其各个技术委员会进行国际标准的制定工作。凡是对于已设有技术委员会的某一专业感兴趣的每一个成员体,都有权参加该技术委员会。与 ISO 有联系的官方和非官方国际组织也可参与国际标准的制定工作。ISO 与国际电工委员会(IEC)在电子技术标准化的所有方都进行密切合作。

各个技术委员会提出国际标准草案,须先分发给各成员体表决通过后,再由 ISO 理事会批准为国际标准。根据 ISO 工作导则,国际标准至少需要投票成员体的 75%赞成。

国际标准 ISO/IEC DIS 10180 是由"联合技术委员会"JTC1 的 SC18 分委员会制定的。

用户应随时注意所有引用的国际标准的修订,以及引用其他国际标准的最新版本,除非另有声明。

ISO/IEC DIS 10180 在《信息技术——文本通信——标准面页描述语言(SPDL)》,目前包括 26 章,详见目次。

附录 A、B、C、D 和 E 都是标准的附录。

中华人民共和国国家标准

信息技术　文本通信 标准页面描述语言(SPDL)

GB/T 16648—1996

Information technology —Text communication —Standard of Page Description Language(SPDL)

0　引言

70年代初诞生的激光打印机提供了低成本打印文件的良好机会，这种文件不仅包含有字符正文，而且还可以包含一般的图形。这就导致人们对标准打印接口不断产生兴趣，通过这种接口可以把这些打印机用作为文件创作系统。提供这样一种接口的最新尝试有两个方面：增强现有打印机接口功能，以充分发挥增加打印机能力的优越性；开发新的文件描述技术，使它更适合于增加图像功能的新打印技术。这些尝试中最成功的地方是并没有采用当时已经存在的正文打印技术和标准，而是采用了基于现代计算机语言的技术，这些打印机接口已经变成了人们熟悉的“页面描述语言(Page Description Language)”或者“PDL”。

这一标准吸取了早期有关页面描述语言方面的工作成果，特别是采用了编程语言的特性。

SPDL定义了由正文和图形组成文件的描述手段，且这种描述与设备无关，并可以在纸或其他介质上显现出来。SPDL还允许去完成如选择页面或SPDL部分文件等功能。

本标准构成如下：

1）定义了SPDL的主要组成和语义。

2）指定了两种交换格式。

3）提供了文件的结构和内容的两类语句。

4）附录中给出了一些例子和其他说明信息。

1　主题内容与适用范围

1.1　主题内容

本标准定义了一种用来描述电子文件的语言，该文件可以由黑白、灰度级或者全彩色的正文、图像和几何图形组成，并以适当的格式表示出来(打印或在其他合适的介质上显现)。

本标准计划是可扩充的，以便适应图像技术的进一步发展。

本标准计划使用多种结构，以满足多种连接的需要，特别是与使用OSI网络兼容。

除了指出如何表示文件图像外，本标准还指出了文件生成指令如何影响文件的显现。

1.2　适用范围

本标准可适用于多种打印和出版环境，包括：

——电子出版；

——办公系统；

——信息网络；

国家技术监督局1996-12-17批准　　1997-07-01实施

——需求打印。

符合这一标准的文件称为 SPDL 文件，可以用于：

——交换；

——通过局部连接的显现设备进行处理；

——发送给与 OSI 或非 OSI 网络连接的显现设备；

——存储，以便今后进行显现。

1.3 与其他标准的关系

符合本标准文件结构的文件编码，若是采用二进制编码，则它应符合 GB/T 16262 和 GB/T 16263；若采用纯正文编码，则它应符合 GB/T 14814。

在表示字符正文时，若字体符合 ISO/IEC 9541 结构，本标准将提供所得结果的详细说明。本标准中用于字体识别的结构名与 ISO/IEC 9541.2 和 ISO 9070 中定义的完全一样。

本标准提供了一种简单而有效地表示由 ODA 系统产生打印文件的方法，它也提供了表示由 SGML 应用产生的文件（其格式由 ISO/IEC 10179(DSSSL)描述）的能力。

本标准提供了符合 ISO/IEC 10175 和 SPDL 文件表示过程要求的打印服务器这一有效工具。本标准还指定了适用于文件生成指令的特定语义。

2 引用标准

2.1 引用的标准

下列标准所包含的条文，通过在本标准中引用而构成为本标准的条文。本标准出版时，所示版本均为有效。所有标准都会被修订，使用本标准的各方应探讨使用下列标准最新版本的可能性。

GB 1988—89 信息处理用于信息交换的 7 位代码字符集(equ ISO 646:1984)

GB 13000.1—1995 通用多八位编码字符集(idt ISO 10646—1:1993)

GB/T 14814—1993 信息处理 文本和办公系统 标准通用置标语言(SGML)(idt ISO 8879:1986)

GB/T 16262—1996 信息处理系统 开放系统互连 抽象语法记法一(ASN.1)的说明(idt ISO 8824:1990)

GB/T 16263—1996 信息处理系统 开放系统互连 抽象语法记法一(ASN.1)的基本编码规范(idt ISO 9925:1990)

ISO/IEC 9541-1:1991 信息处理 字体信息交换 第 1 部分：结构

ISO 9070:1991 信息处理 SGML 支撑工具 用于公共正文拥有者标识符登记过程

IEEE 854—1987 与基数无关的浮点算术数据的 IEEE 标准

2.2 参考的标准

本标准有关条文的补充或说明信息包含了下面的标准或标准工作草案，它们在将来可能变成标准引用。

GB/T 15936—1996 信息处理 文本和办公系统 办公文件结构(ODA)和交换格式，部分 1、2 和 4 到 8(idt ISO/IEC 8613)

ISO/IEC 9541-2:1991 信息处理 字体信息交换 第 2 部分：交换格式

ISO/IEC 9541-3:1991 信息处理 字体信息交换 第 3 部分：字形形状表示

ISO/IEC DIS 10179 信息处理 文本和办公系统 文件类型语义和描述语言(DSSSL)

ISO/IEC DP 10175 信息处理 文本和办公系统 文件打印应用

3 定义

本标准采用下列定义。

3.1 附加 DPI additional DPI

其名字不是由本标准定义的文件生成指令(DPI—document production instructions)(见 8.1.2)

3.2 复制区域 assured reproduction area

某种媒体上可以描述页面图像的区域。

3.3 基字型 base font

一种直接包含或者指定字形形状的字型;在复合字型中,它是复合字型树结构中的一个叶结点。

3.4 基本结构元素 base struction element

一种不能再分小的结构元素。

3.5 赋值 bind

用符号表达式来表示一个特定对象。

3.6 位 bit

信息单位,其值可以是 0 或 1。

3.7 黑色生成 black generation

计算黑色颜料的数量,以得到一种特殊的颜色。

3.8 布尔(值) Boolean(value)

布尔类型的值。

3.9 布尔类型 Boolean type

其值可以是真(true)或假(false)的一种类型。

3.10 基数(值) cardinal(value)

基数类型的值,一种非负的整数。

3.11 基数类型 cardinal type

整型数的一个分支,其值为非负整数。

3.12 字符正文 character text

由一或多串字形图像表示的文本。

3.13 裁剪 clip

消除某个图像或图形符号的一部分,它是通过不画这一部分来实现的。

3.14 裁剪区 clipping region

一种限制页面图像生成区域的基本成像模型。在区域内部,油墨可以成像(见 10.2.3);而在裁剪区的外部,任何图像将被裁剪掉。

3.15 复合字型 composite font

一种层次结构(树型结构)的字型集合。

3.16 构图和布局过程 composition and layout process

文件生成模型(见 4.1)的基本组成部分之一。

3.17 上下文词典 context dictionary

由 DICTIONARY GENERATOR 结构元素定义的词典。

3.18 复合结构元素 composite structure element

一种可以再分成更小结构的结构元素。

3.19 (复合结构元素的)内容 content(of a composite structure element)

隶属于复合结构元素的所有 TOKENSEQUENCE 结构元素的内容。

3.20 (TOKENSEQUENCE 的)内容 content(of a TOKENSEQUENCE)

八位字节串,它是 TOKENSEQUENCE 结构元素的值。

3.21 内容处理器 content processor

SPDL 文件处理模型中负责处理 SPDL 内容的部分(见 4.3.4)。

3.22 解释上下文 context of interpretation

由内容处理器解释的 TOKENSEQUENCE 内容的上下文(见第 9 章)。

3.23 当前解释上下文 current context of interpretation

与一个特定 BLOOK 有关的解释上下文,并且它用来处理下属 TOKENSEQUENCE 结构元素的内容(见 9.2)。

3.24 当前页面图像 current page image

SPDL 文件处理模型中生成页面图像的部分(见 4.3.3 和 9.2)。

3.25 子字型 descendant font

在复合字型中,层次结构比根字型低的字型。

3.26 设备色域 device gamut

组成输出设备可能产生彩色的各种颜色的子集。

3.27 词典 dictionary

由内容处理所创建的数据结构,它由名字一值对组成(见 10.1.4)。

3.28 文件 document

当没有其他限制时,它是以可见格式表述的电子信息。

3.29 文件创建程序 document creator

创建最终版式文件的过程。如果没有其他限制,最终版式文件以 SPDL 形式出现。

3.30 文件 DPI document DPI

文件内部指定的文件生成指令集。

3.31 文件生成指令 document production instructions

影响文件输出的指令。

3.32 文件结构 document structure

用来构造 SPDL 文件和它们之间关系的一组构成部分。

3.33 编辑过程 editing process

文件处理模型的基本组成部分之一(见 4.1)。

3.34 环境 environment

适用于输出过程的所有外部信息的集合。

3.35 外部信息 external information

其本身是在文件外部的资源和结构元素,但是在文件的输出过程中可以通过引用来使用这些信息。

3.36 外部结构元素 external structure element

其本身是文件外部的结构元素,但是通过引用可以使之包含在文件内部。

3.37 最终版式文件 final form document

以适合于表示过程描绘的版式给出的文件,在应用本标准时,这一术语就意味着所有的构图和布局处理均已完成(也可以见"可修改的版式文件")。

注:最终版式一般不会比刚开始编辑的文件更长。当然可修改版式和最终版式彼此之间无排它概念;某些数据,如 GB/T 15936(ODA),在单个文件中允许可修改版式和最终版式同时存在。

3.38 结束指令 finishing instruction

继描绘文件页面图像之后影响文件生成过程的文件生成指令。

3.39 字型索引映射表 font index map

复合字型中,在字型的映射变换过程中把字型索引与字型选择符联系起来的向量。

3.40 字型对象 font object

字型和用来表示字符正文的字形索引表的结合。

3.41 字型资源 font resource

字形描述集合再加上与字形描述集合有关的总体说明和字的尺度等信息。

3.42 伽马修正 Gamma correction

描绘函数，用来补偿输出设备和/或人眼的非线性效应。

3.43 色域变换 gamut transfer mapping

从源色域到设备色域这一变换颜色的过程，尽可能地维持由文件生成程序所指定的颜色外表和视觉之间的反差。

3.44 字形 glyph

抽象的标识图形的符号，它独立于任何实际图像。

3.45 字形图像 glyph image

从字形描述中求得字形图像。

3.46 字形索引映射表 glyph index map

字形索引值和字形标识符之间的映射关系，该标识符可以用来识别字型中的字形描述。

3.47 字形描述 glyph representation

把字形形状和字形尺度信息(glyph metrics)与字型中的指定字形联系起来。

3.48 字形形状 glyph shape

字形描述中用来定义该字形形状的一组信息。

3.49 图形元素 graphical element

一可见的信息单位。

3.50 半色调单元 halftone cell

一组像素，借助像素图案可以使该组像素具有近似连续色调的颜色；半色调屏幕上极小的单位。

3.51 半色调屏幕 halftone screen

定义了一种映射变换，即均匀的半色调单元矩形网格覆盖了设备的像素阵列。

3.52 半色调屏幕角 halftone screen angle

半色调屏幕网格子线在由设备像素阵列所定义的坐标系中的走向。

3.53 半色调屏幕频率 halftone screen frequency

在由设备像素阵列所定义的坐标系中，Y 方向上每一厘米半色调单元的数目。

3.54 半色调技术 halftoning

借助像素图案以得到所希望的近似颜色；该技术适用于不可能生成连续色调的输出设备，仅能用生成的某些不连续的颜色来代替。

3.55 高层结构元素 high level structure element

三种类型的结构元素中的一种，它们是可以出现在最高结构层的：PICTURE，PAGESET 和 RESOURCE DEFINITION。

3.56 最高结构层 highest structure level

一个不从属于任何其他结构元素的结构元素，则说它位于最高结构层。

3.57 图像元素 image element

成像模型的基元；单个成像动作所得的结果(见 10.2.4)。

3.58 标识符(值) identifier(value)

标识符类型的值。

3.59 标识符类型 identifier type

标识值的类型是表示一序列字符的八位字节串。如果表示的八位字节串相等，则两个标识符相等。

3.60 标识符值 identifier value

标识 SPDL 例客体的值

3.61 成像装置 imager

SPDL 文件处理模型的组成部分，其职责是作某些交换以得到当前图像(见 4.3.3)。

3.62 直接下属 immediate subordinate

复合结构元素已被再分细的部分。

3.63 实现 implementation(注：原文无定义)

3.64 油墨 ink

成像模型的基元，用来在输出设备上指定显示色和可见纹理。

3.65 内含数据 in-line data

物理上包含在 SPDL 文件中的光栅图形图像数据(对应于逻辑上包含在文件内部的外部数据)。

3.66 SPDL 实例 instance of SPDL

一个 SPDL 文件或用本标准中所定义的标准页面描述语言所表示的资源。

3.67 整数(值) integer(value)

数学上的整型数。

3.68 整数类型 intege type

一种类型，其识别值为数学上的整型数。

3.69 管理指令 management instructions

管理文件生成过程的文件生成指令。

3.70 掩模 mask

成像模型的基元，它提供了图像元素的形状(见 10.2.2)。

3.71 媒体 medium

输出设备上可以用可见格式描绘文件的物理界面，例如纸和视频显示终端的显示屏幕。

3.72 媒体说明 medium declarations

用来指定文件输出介质的文件生成指令。

3.73 名字(值) name(value)

类型为名字的值。

3.74 名字类型 name type

一种类型，其识别值为 # 中定义的字符序列。

3.75 空类型 null type

识别值为 GB/T 16262 所定义的“NULL”类型。

3.76 对象名 object name

用来标识一个信息对象的名字，本标准指定了如 GB/T 16262 中定义的对象标识符值和如 ISO 9070中定义的公用标识符(见 6.3)。

3.77 对象类型 object type

一种标识，与 SPDL 虚拟机对象有关的对象类型(见第 10 章)。

3.78 操作符 operator

一种能使 SPDL 虚拟机执行某种操作的解释标记。

3.79 八位字节 octet

一个有序的八位序列。

3.80 八位字节串(值) octet string(value)

八位字节串类型的值。

3.81 八位字节串类型 octet string type

识别值是 0 个或者多个八位字节序列的类型。

3.82 页面描述语言 page description language

基于编程语言概念的一组控制打印格式的命令。

3.83 页面图像 page image

成像模型的一类基元，页面图像是由不断地在原来空白的页面上增加一序列图像来实现的。

3.84 父(上级)字型 parent font

在一个复合字型中，父(上级)字型有给定的子(下级)字型。父字型是根结点，而子字型是根结点下的子叶。

3.85 路径 path

引用坐标系中一系列(0个或者多个)点、线和/或曲线所定义的几何图形。

3.86 同级(结构元素) peer(structure elements)

直接隶属于同一结构元素的结构元素。

3.87 图 picture

一幅独立于其他图像且可以进行单独处理的图像，例如一个页面或页面上的一个图。

3.88 预处理指令 pre-processing instruction

在描绘文件页面图像以前，影响文件生成过程的文件生成指令。

3.89 显现设备 presentation device

3.90 显现指令 presentation instruction

影响文件页面图像的描绘过程和它们在介质上放置的文件生成指令。

3.91 显现过程 presentation process

在合适的介质上描绘一个文件或部分文件的过程。

3.92 显现过程 presentation process

文件生成模型的基本组成部分(见4.1)。

3.93 显现次序 presentation order

一系列SPDL文件结构元素的次序。

3.94 可打印字符串(值) printatble string(value)

一个可打印字符串类型的值。

3.95 可打印字符串类型 printatble string type

一种数据类型，其识别值为GB/T 16262中定义的可打印字符串。

3.96 结构元素 prologue structure elements

PROLOGUE结构元素或者从属于PROLOGUE的结构元素。

3.97 公用对象标识符类型 public object identifier type

一种类型，根据所用的结构表示其识别值可以是SGML格式的公用标识符或者ASN.1对象标识符。

3.98 公用对象标识符值 public object identifier value

公用对象标识符类型的值。

3.99 实数类型 real number type

一种数据类型，其识别值由IEEE 854所定义。

3.100 引用坐标系(RCS) reference coordinate system(RCS)

一笛卡尔坐标系，其两根轴的单位是毫米，该坐标系在媒体上提供了明确的位置说明(见10.3.1)。

3.101 引用名(值) reference name(value)

引用名类型的值。

3.102 引用名类型 reference name type

一种类型，其识别值是名字值和公用对象标识符值。

3.103 描绘 render

建立一个可见图像(字形或图形元素)，它作为表示过程的一部分。

3.104 描绘过程 rendering

控制光栅输出设备的过程。

3.105 资源 resource

可在表示过程环境下应用的信息对象，且可在表示一个 SPDL 文件过程中加以引用。

3.106 可修改版式文件 revisable form document

以便于某些设备进行编辑和修改版式出现的文件。本术语一般意味着除了它的显式内容以外，文件还可以包含有标识其逻辑结构的信息(也可以见“最终版式文件”)。

3.107 根字型 root font

在复合字型中，层次结构中最高层的字型。

3.108 扫描转换 scan conversion

描绘过程的最后一步；把页面图像转换成指定设备的像素。

3.109 源色域 source gamut

在成像操作时，组成 SPDL 文件颜色的所有可能颜色的子集。

3.110 SPDL 文件 SPDL document

以本标准中定义的标准页面描述语言表示的(最终版式)文件。

3.111 点函数 spot function

半色调单元中的像素将按照 SPDL 文件中所指定次序改变值的一种方法，以控制正方形半色调单元的使用。

3.112 (SPDL)结构 (SPDL)structure

用来构造 SPDL 实例和它们之间关系的一组成分。

3.113 结构元素 structure element

借助文件结构来区分的任何 SPDL 文件元素。

3.114 结构处理 structure processing

对构造 SPDL 文件组成部分进行的处理。

3.115 结构处理器 structure processor

SPDL 文件处理模型的组成部分，用来处理 SPDL 结构(见 4.3.3)。

3.116 下属 subordinate

通过进一步再分某一结构元素所得到的结构元素，我们说它是某结构元素的下属。

3.117 上级 superior

如果 B 下属于 A，则我们说 A 结构元素是 B 结构元素的上级。

3.118 辅助 DPI supplementary DPI

与一个特殊文件表示有联系但又不包含在该文件内部的文件生成指令。

3.119 阈阵列 threshold array

一种方法，通过该方法半色调单元中的像素将按照 SPDL 文件所指定的次序改变值；对正方形半色调单元的使用没有什么限制，但它更多地与设备有关。

3.120 标记 token

可以由 SPDL 虚拟机解释的信息的逻辑单位。

3.121 填图案 tiling

在填充区域内铺设图案的过程。

3.122 变换函数 transfer function

一种方法，通过该方法 SPDL 文件执行指定的伽马修正。

3.123 类型 type

一组值的标识。

3.124 类型(按内容分) type(in content)

与 SPDL 虚拟机对象(见第 10 章)有关的一个标识对象类型。

3.125 底色去除 undercolour removal

计算减少青色、品红和黄色颜料的数量,以补偿由黑色生成(black generation)所决定的黑色颜料的数量。

3.126 用户坐标系 user coordinate system(UCS)

一个无单位的笛卡尔坐标系,把该坐标系映射到由 current transformation 所定义的引用坐标系(见 10.3.2),则可以在介质上指定位置。

3.127 虚拟机 virtual machine

通过定义一个理想化的装置来实现的过程描述。某个理想化的描述仅仅服务于指定过程,而并不支配实际的装置。

4 SPDL 一般结构

本章叙述标准页面描述语言的一般结构。基本的文件处理模型在 4.1 叙述,本模型重要之点在于信息必须能用 SPDL 文件来描述。用来描述这一信息的一般结构在 4.2 介绍,交换格式在 4.3 加以叙述。SPDL 表示过程与打印服务器和文件打印应用之间的关系将在 4.4 叙述。

4.1 文件处理模型

基本的文件处理模型由三类处理过程和两种版式文件组成,如图 4-1 所示。编辑过程形成可修改版式文件的内容和逻辑结构,构图和布局过程版式化可修改版式文件中的表示信息,并得到最终版式文件。显现过程以一种可见版式描绘最终版式文件,这些过程下面将进一步叙述。

注:这一模型进行了高度的简化,而实际的文件处理过程一般会多次重复,可修改的版式文件会被进一步版式化以便进行校对。构图和布局过程的结果必须用定值表示出来。许多应用还包括多种过程,文件编辑系统经常包括构图、布局和描述过程,以便对文件进行预观察,而某些打印机还可能把构图和布局过程与显现过程结合起来。

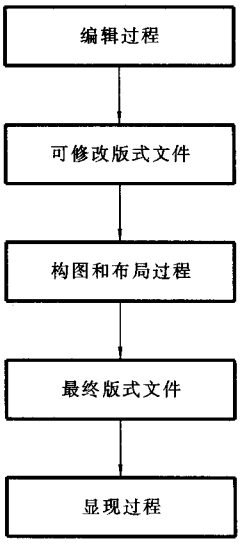


图 4-1 文件处理模型

4.1.1 编辑过程

编辑过程负责建立文件内容和逻辑结构,它也可能产生用来指定如何进行构图和布局处理的信息,以便版式化用于显现的文件内容。

4.1.2 可修改的版式文件

可修改的版式文件表示是由编辑过程所建立的文件内容和逻辑结构，它也包括了由编辑过程产生的、用来指出文件内容如何版式化以便显现的其他信息。

4.1.3 构图和布局过程

构图和布局过程组织文件内容，并对所显现的内容进行布局处理。所有构图、版式化和定位等的决定是由构图和布局过程实现的，这些决定详细包括：

——选择显现字符正文时所用的字形；

——定位每一个字形，隐含地或显式地查找字型；

——选择线的权值和末端，并加入相应特性；

——选择几何对象的填充图案。

4.1.4 最终的版式文件

最终的版式文件表示的是由构图和布局过程所产生的文件内容和版式化所决定的结果，它并不要求去表示文件的逻辑结构、版式说明或这些决定的目的。

4.1.5 显现过程

显现过程就是在合适的介质上以可见格式去描绘一个文件，这一过程还包括使所描绘的文件能适应显现设备的要求，以便提供由构图和布局过程所指定的最合适的描述。

显现过程还执行一些结束操作（如整理、装订或捆扎），它们是文件表示的一部分。

4.2 SPDL 文件

SPDL 是一种最终版式文件，它提供了有效地表示由构图和布局过程所得结果的能力。SPDL 并不提供表示文件逻辑结构的功能。

4.2.1 源 SPDL 文件

主要的源 SPDL 文件是应用构图和布局过程所得到的可修改的版式文件。此外，SPDL 文本也可以由以下一些方法来建立：

——转换一个已完全版式化且以某种替补的最终版式结构表示的文件；

——一种强制过程或其他构图和布局过程，它们适用于最终版式文件或最终版式和可修改版式文件的混合物。

4.2.2 SPDL 文件显现

SPDL 文件描述了由构图和布局过程所指定的理想图像。SPDL 显现过程的责任是使得所给出的文件能适应于显现设备的要求，以便提供由构图和布局过程所指定的最合适的描述。

SPDL 显现过程也执行某些结束操作（如收集、装订或捆扎），它们是文件表示的一部分。

一个 SPDL 文件可以在不同时间和/或不同地点显现多次。由于这一原因，文件显现必须提供特定显现的有关参数，这些参数叫做文件生成指令（Document Production Instructions）。

文件生成指令可以作为部分 SPDL 文件传送给显现过程或用于一台独立机器。通过某些方法，可以使不是文件部分的文件生成指令用于显现过程，而这种方法已超出了本标准的范围。

4.2.3 SPDL 文件的使用

SPDL 文件的基本用法是输入给 SPDL 显现过程。此外，SPDL 文件可以用来表示一个已版式化的文件内容。例如，先前已版式化的文件内容可以用来作为另外一个文件中构图和布局过程的输入。

4.3 SPDL 文件结构

SPDL 文件分成结构和内容两部分。文件结构独立于内容，且可以不依赖于内容单独处理，内容以页面描述语言的格式出现，它的处理依赖于文件结构。

当结构处理用于显现这一目的时，结构处理为文件内容的每一个元素建立上下文说明。

注：当结构处理用于非文件显现的其他目的时，进行内容处理一般就不需要了。

显现过程还包含有适用于它的各种源附加数据，例如字型或 SPDL 文件的存储部分，它们可以由 SPDL 文件所引用。所有这些数据的总和叫显现过程环境。

4.3.1 SPDL 文件结构

SPDL 文件结构是把一个 SPDL 文件不断地划分成更小部分的结果，这较小部分叫结构元素。

SPDL 文件采用分层结构，最高层结构元素是 DOCUMENT。一个 DOUMENT 可以包含多个叫 PAGESET 和/或 PAGE 结构元素的下级结构元素。

一个 PAGESET 可以逐级包含多个下级 PAGESET 和/或 PAGE。每个 PAGE 包含一个文件描述部分，它们可以在介质上显现出来。

一个 PAGE 可以包含多个叫 PICTURE 和/或 TOKENSEQUEN 结构元素的下级结构元素，一个 PICTURE 也可以包含多个子结构 PICTURE 和/或 TOKENSEQUENCE。TOKENSEQUENCE 是一个包含文本内容的结构元素。

DOCUMENT 和 PAGE 结构元素是语义上的差别，而语法上没有什么不同。DOCUMENT 是分层结构中位于最高层的简单 PAGESET 或 PICTURE。同样地，PAGE 是在 PICTURE 分层结构中位于最高层的简单 PICTURE。

SPDL 文件结构的例子如图 4-2 所示。

SPDL 文件除了可以分成更小的部分以外，SPDL 文件结构还支持

——按照正确表示次序给出的一系列 SPDL 文件页面；

——应用文件生成指令；

——建立 SPDL 内容元素的上下文说明；

——解决对 SPDL 文件外的结构元素的引用；

——解决对 SPDL 文件外的资源的引用；

——资源的说明；

——由其他过程来选择页面或页面的一部分。

注：用于文件显现以外目的的 SPDL 结构处理（如提取包含在其他文件中 PICTURE 结构元素的构图和布局处理，或者强制处理）已超出了本标准的范围。

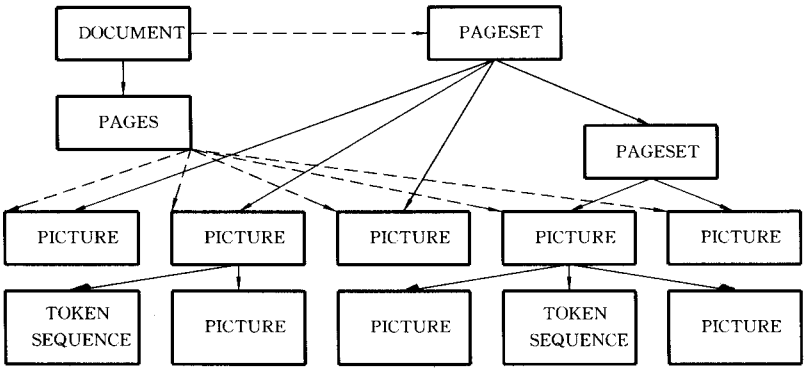


图 4-2 SPDL 文件结构的例子

4.3.2 文件内容

SPDL 文件内容是包含在 TOKENSEQUENCE 结构元素中的数据。在每一个 TOKENSEQUENCE 内的文本内容由一系列标记组成。文件内容的处理，即去解释每一个 TOKENSEQUENCE 中的一系列标记。定义这一复杂的动作需要一系列子模型。

纵观标记解释过程类似于虚拟状态机模型，定义为成像模型的成像动作包括坐标系统、油墨和裁剪区域。

此外，有三种类型的画图操作：字符正文、光栅图形（图像）和几何图形。这三种类型中的每一种都有

自己的子模型。

注：这些模型仅仅是用来定义文件内容语义的一种手段，它们并不控制内容处理机的具体实现，也不限止文件内容语义以一种特殊方式实现的方法。

4.3.3 外部信息和资源

外部信息是指用来显现 SPDL 文件时所需要的、但其本身又并不包含在内的信息。SPDL 文本中可引用的外部信息有两种类型：外部结构元素和资源。

外部结构元素是包含在一个引用该元素的 SPDL 文件内的 SPDL 结构元素。引用外部元素的 SPDL 文件将以外部说明结构元素来说明它。

资源是一个可以在显现过程环境下使用的信息对象。资源可以按本标准所指定的方式进行定义，或者通过某种处理使得资源可在显现环境下应用，当然这些处理已超出了本标准的范围。在一个 SPDL 文件中，可以通过引用来使用资源，引用资源的 SPDL 文件将以 RESOURCE DECLARTION 结构元素来说明它。

4.3.4 文件处理

由于 SPDL 文件是由文件结构和文件内容组成的，且由于这些数据是可分开的，所以显现过程就类似分别处理结构和内容的子过程这一模型。在本标准中这些处理分别称为结构处理器和内容处理器。显现过程还包含一个成像装置(Imager)，用来在介质上生成页面图像，见图 4-3。

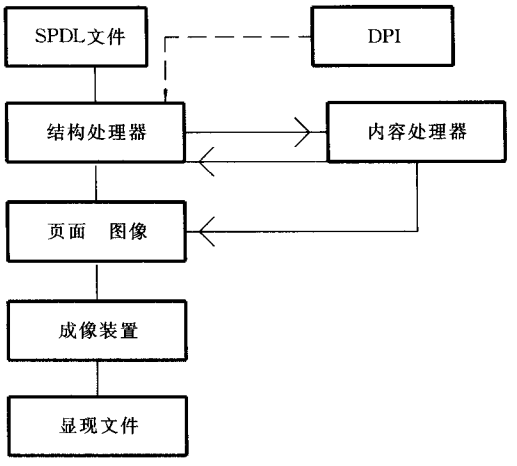


图 4-3 SPDL 显现过程

注：结构处理和内容处理逻辑上是分别进行的，而在显现过程的实现时分别处理又并不存在，仅需要 SPDL 文件表示以本标准所指定的结果出现。

文件结构处理包括分析 SPDL 文件的结构元素、它们的类型、层次关系和表示的次序。表示结构处理还

——解决文件生成指令之间的相互作用；

——使用文件生成指令生成相应 SPDL 文件的 PAGESETS 和 PAGES；

——建立每一个 TOKENSEQUENCE 的上下文说明。

4.3.5 交换格式

SPDL 文件按第 25 章定义的交换格式进行交换。交换格式有两种表示方法：纯正文表示和二进制格式表示；并分成两个部分：结构交换格式和内容交换格式。文件内容的二进制表示与文件结构的二进制表示一起使用；文件内容的纯正文表示与结构的纯正文表示一起使用。

二进制表示的结构交换格式(或二进制结构交换格式)使用 GB/T 16262 或 GB/T 16263 中定义的抽象语法表示 1(ASN.1)。纯正文表示的结构交换格式(纯正文结构交换格式使用 GB/T 14814 中定义

的标准通用置标语言(SGML))。

内容交换格式使用专门的优化表示法,以便使每一个二进制或纯正文表示紧凑而有效。

4.4 与打印服务器之间的关系

能打印 SPDL 文件的打印服务器内含一个 SPDL 显现过程,该过程作为打印服务器的一部分。本节讨论某打印服务器与它内含的 SPDL 显现过程之间的关系。

4.4.1 文件生成指令

当通过打印服务器打印一个 SPDL 文件时,打印操作可以包括影响文件成像和显现的那些参数。某些参数用来指定名为辅助 DPI(Supplementary)DPI 的文件生成指令。打印服务工具负责使 SPDL 文件和辅助 DPI 可适用于 SPDL 显现过程。

SPDL 显现过程解决了与一个特殊显现实例相关的辅助 DPI 所提供的文件生成指令和包含在 SPDL 文件中的任何文件生成指令之间的相互作用。这一关系可由图 4-4 来说明。

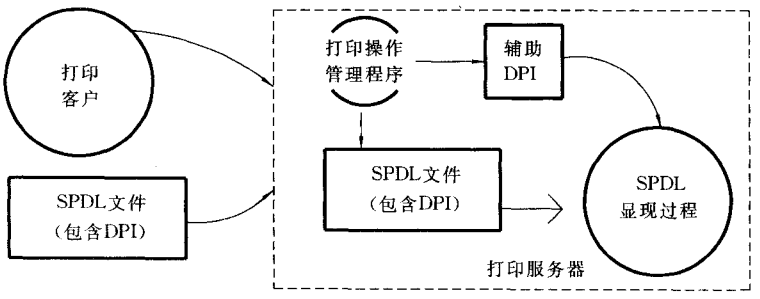


图 4-4

4.4.2 与 ISO/IEC DP 10175 的关系

ISO/IEC DP 10175 定义了一个提供标准(抽象)打印服务的文件打印应用的标准。文件打印是通过调用打印操作以及相应文件的一组打印操作参数来完成的。

ISO/IEC DP 10175 所定义的打印操作参数由打印作业管理指令和文件生成指令组成。

打印作业管理指令并不影响 SPDL 文件的表示。某些(并不是所有的)由 ISO/IEC DP 10175 所定义的文件生成指令适用于 SPDL 显现过程。

ISO/IEC DP 10175 打印作业管理指令的语义,和可以作为 ISO/IEC DP 10175 打印操作参数出现的这些文件生成指令的一般语义均由 ISO/IEC DP 10175 来定义。可以作为 ISO/IEC DP 10175 打印操作参数出现,且可适用于 SPDL 文件显现的文件生成指令的特定语义由本标准定义。

5 记法

本章叙述本标准文件中所用到的印刷上表示各类信息的惯例。

5.1 结构元素

5.1.1 结构元素类型

结构元素类型由完全是大写字形的名字来表示。

例如:PAGESET

PICTURE

5.1.2 特殊类型的结构元素

特殊类型的结构元素由结构元素类型名来指出。

例如:一个 PAGESET

一个或多个 PICTURE 结构元素

5.1.3 公用对象标识符值

如本标准附录中所指出的那样，与对象名(object name)相对应的一个公用标识符值，可以作为一个“对应于对象名为(object name)的一个公用对象标识符值”来引用。在这类语句中，对象名将用斜体字印刷。

5.2 对象

SPDL 虚拟机所处理的对象包含有类型和值。一个类型为“Type”和值为“Value”的对象可用“〈Value:Type〉”来表示。

5.2.1 对象类型

对象类型由开头为大写字形的斜体字来表示，例如：

Number

Vector

5.2.2 特定类型的对象

特定类型的对象用纯正文的对象类型名来表示，例如：

Cardinal

Vector

5.2.3 对象值

对象的值可以由名字或者其值本身来表示，值所在的地方由名字来表示，名字(可以是大写和/或小写字形)用斜体字，例如：

〈*n*:*Number*〉

〈*fred*:*Boolean*〉

〈*T*:*Transformatcon*〉

带有特定值的对象在其名字的地方用对象值来表示。例如：

〈2：*Integer*〉

〈0.5：*Number*〉

5.2.4 缩写格式

当一个对象的值由其名字来表示时，它在同一子句中可以引用多次，并且上下文提供了足够的信息以避免二意性，对象可以用值的名字来简单表示。

例如：“n”和“T”可以用来重复引用“〈n:Number〉”和“〈T:Transformation〉”。

5.2.5 对象序列

对象序列可以由包含在括号内的用逗号分开的一串对象来表示。例如：有三个对象的串，其每一个的类型是整数，就可以用(整数，整数，整数)来表示。

这种记法也可以用来表示序列长度不定的由许多对象或值组成的序列，例如：一个有 n 个对象的序列，其头、尾的值分别为 Valuel 和 Value n，则序列可以表示为(Value1，Value12，…，Value n)。

5.3 字面值 Literals

字面值采用与对象相同的方法来表示。

5.4 操作符

操作符用它们的名字来表示。名字是一个复合词，粗体，且其每个组成部分的第一个字符为大写字形，例如：

FindFont

FillPath

5.5 状态变量

状态变量用它们的名字来表示。名字是个复合词，斜体，其每个组成部分的第一个字符为大写字形，所有的状态变量名均是用“Current”开头，如：

CurrentFont

CurrentStrokejoin

5.6 操作数栈

当必须表示操作数栈状态信息的时候，可用以下的方法进行。操作数栈顶的对象用一垂直列表来表示，对象后面可以有选择地紧接与对象特征有关的注释。如下所示：

〈object1：Any〉object at top of operand stack

〈object2：Any〉Second object on operand stack

这一列表就意味着栈中的状态 object1 是紧跟着 object2 被压入堆栈的（因此 object1 实际上是栈顶对象）。

这种列表仅表明操作数栈顶的少数几个对象，操作数栈可以保存任意多个附加对象，这一切也是可以理解的。

5.7 操作符描述

由本标准定义的每个操作符其描述内容如下：

——操作符开始解释执行时弹出操作栈的操作数的数值和类型；

——操作符解释执行结束时压入操作数栈的结果的数值和类型；

——操作数与结果之间的关系；

——操作符解释执行时对虚拟机状态产生的任何其他结果。

操作数表和结果表均以操作数栈的描述格式给出。

结果的描述以正文方式给出，或者可以包括一串 SPDL 标记（tokens），解释这些标记将得到相同的结果，记法为：

{sequence of tokens}

这就表示解释一串花括号内的标记所得的结果。在这种情况下，标记就表示成如上面所说的字面值（literals）或操作符，串中的标记由空格将它们分开。

6 文档结构和结构处理

将 SPDL 的文档进行划分，并重复这一过程可以产生越来越小的部分，这就形成了 SPDL 的文档结构。划分的过程导致了层次结构。

描述 SPDL 结构的方法是自然的，这不同于对其内容的描述所采用的过程式语言的形式。构成文档的结构元素的集合是有序的。

每个结构元素有一个类型和一个值，复合结构元素的值是下属结构元素的集合，基本结构元素的值是基本（primitive）类型。构成基本结构元素的类型以及基本元素的值的类型在第 6 章中定义。用于标识文档中用到的资源和用于描述文档生成指令的结构元素的类型在第 7 和第 8 章中定义，文档结构与内容处理之间的接口在第 9 章中讨论。

6.1 层次结构

下面的概念和术语与层次结构及它们之间的关系有关。

6.1.1 结构元素 Structure element

借助文档结构来区别的 SPDL 的文档元素称为结构元素（Structure element）。

6.1.2 复合结构元素与基本结构元素

划分 SPDL 文档而得到的结构元素本身可能还可继续划分成更小的结构元素。可以划分为更小结构元素的结构元素称为复合（composite）结构元素。不可再分的结构元素称为基本（base）结构元素。

复合结构元素的值由其下属（或称子结构）组成。对于各复合结构元素类型，其定义的子句指出了该类型的一个结构元素的直接下属及该结构元素的处理方法。

6.1.3 下属和直接下属 Subordinates and immediate Subordinates

由复合结构元素划分成的部分称为其直接下属（或称直接子结构）。由于直接下属可能是复合结构

元素，因此可以对它进行再划分，这时划分成的部分就不是原复合结构元素的直接下属。所有通过这一步步划分而得到的结构元素，无论是复合结构元素还是基本结构元素均称为原结构元素的下属或称从属于该结构元素。

对于“直接下属”常常采用“直接”两字加以说明，不带这一定语时称“下属”，即指划分至任何层次的下属。

6.1.4 上级和直接上级 Superiors and immediate Superiors

经常需要指出一个结构元素是从哪里划分而来的。当 B 是 A 的下属时，称结构元素 A 为结构元素 B 的上级。当 B 是 A 的直接下属时，称 A 是 B 的直接上级。

对于“直接上级”，常用“直接”一词加以说明，不带这一定语的，“上级”是指任何层次的上级结构元素。

6.1.5 最直接上级结构元素 Most immediately superior structure elements

类型为 T(简称“T”)的结构元素是 A 的最直接上级，这一术语用于说明结构元素 A、B 和结构元素类型 T 三者之间的关系。结构元素 B 为“A 的 T 最直接上级”是指：

——B 是类型为 T 的结构元素；

——B 是 A 的上级；

——不存在具有类型 T 的结构元素，它既是 A 的上级又是 B 的下属。

6.1.6 最高结构层 Highest structure level

不可以作为任何其他结构元素的下属的结构元素称为具有最高结构层。

类似地，由于 PICTURE 结构元素的下属可能包括其他 PICTURE 结构元素，但不可能包括 PAGESET 结构元素，因此不属于任何其他 PICTURE 的 PICTURE，可以称该 PICTUFE 在层次结构中处于最高层。

6.1.7 同级 peer

某结构元素的各个直接下属称为同级结构元素。结构元素之间既不为下属又不为上级，相互之间称为无关(unrelated)。

6.1.8 结构元素类型 Structure element types

SPDL 文档的每个结构元素都有一特定的类型，直接下属的类型是其上级结构类型定义的一部分。

6.1.9 作用域 Scope

结构元素有“作用域”。结构元素 A 的作用域是一个结构元素的集合，其中的结构元素可以使用 A 中定义和说明的信息(对象或资源)。定义或说明信息的结构元素的作用域是定义该结构元素的一部分。

若结构元素 B 在结构元素 A 的作用域内，则在 A 中定义或说明 的对象或资源在 B 的上下文中可以被定义或说明。

6.2 序列次序 Sequential order

除了层次结构以外，SPDL 结构对每个复合结构元素的直接下属引入了序列次序的概念。

同级结构元素的序列次序和层次结构的结合，给出了组成文档结构元素集合的一个序列次序。序列次序中处在后面的结构元素称为紧跟(follow)前面的结构元素，同样地前面的结构元素称为先于(precede)后面的结构元素。

6.2.1 序列次序与层次结构 Sequential Order and Hierarchical Structure

序列次序与层次结构的关系有两条规则：

——SPDL 文档的每个结构元素后面紧跟着它的所有下属。

——先于某同级结构元素的任何结构元素，那其下属也必然先于该同级结构元素。

于是，序列次序相对于层次结构具有“前序深度优先(pre-order depth-first)”性质。

6.2.2 表示次序

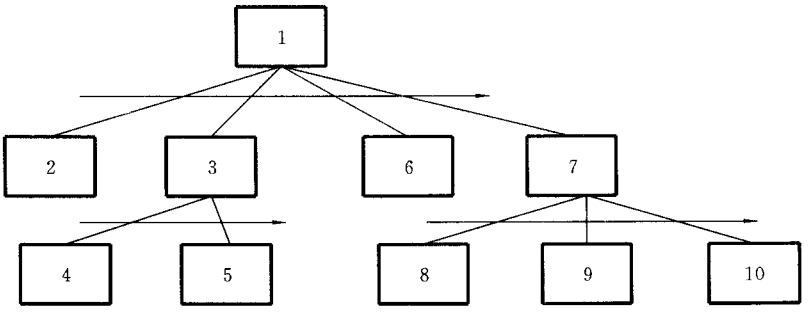
表示次序是又一个用于 SPDL 文档内部结构元素子集的术语。例如，SPDL 文档中“PAGE 结构元

素的表示次序"是指 PAGE 结构元素出现在 SPDL 文档中的表示次序。PAGE 结构元素的描述次序就是文档的页面提供给用户的次序。

结构元素的序列次序也是 SPDL 文档的表示次序，它是在文档表示过程中结构处理器处理结构元素的次序。

6.2.3 例子

图 6-1 给出了序列次序与层次结构相互关系的一个例子。图中结构元素以方框表示，用连线表示其下属关系。每个结构元素的直接下属的次序从左至右用箭号表示。构成文档的结构元素集合，其序列次序用方框中的数字来表示。



注

1 方框中的数字表示序列次序。

2 箭号表示直接下属的次序。

图 6-1 层次结构与序列次序的关系

6.3 基本结构元素值的类型

基本结构元素的值具有如下基本类型之一。

6.3.1 布尔型 Boolean type

布尔型的值只有 True 和 False。

6.3.2 整型 Integer type

整型的值为整数。

6.3.3 实型 Real type

实型的值属于 IEEE 854 标准所规定的集合。

一个整数作为实型值和作为整型值是有区别的，虽然数值上是相等的。

6.3.4 数值型 Number type

数值型的值包括整型和实型全部数值。

6.3.5 名字型 Name type

名字型的值是 11.3.5 中所定义的伪类型(Pseudo type)的值。

6.3.6 可打印串型 Printable String type

可打印串型的值为 GB/T 16262 标准中所定义的 PrintablString 的值。

6.3.7 八位字节串型 Octet String type

八位字节串型的值是零个或多个八位字节的序列，每个八位字节是一个八位(bit)的序列，定义见 GB/T 16262。

6.3.8 公用对象标识符型 Public Object Identifier type

公用对象标识符型的值取决于所采用的结构表示。

——采用二进制结构表示时，公用对象标识符类型的值为 GB/T 16262 中所定义的"object identifier value"。

——采用纯文本结构表示时，公用对象标识符类型的值为 ISO 9070 中所定义的"public identifier"，

其中的“owner name”应为“registed owner name”。

注

1 公用对象标识符的值通常与一信息对象相联系，其目的是为了标识其使用范围内的确定性。

2 在 ASN.1 和 SGML 环境下使用的信息对象，通常赋予一个 ASN.1 对象标识符值和一个已登记的公用标识符。

3 不存在一个通用的机制来将 ASN.1 对象标识符值转化为一个公用标识符，使其能标识同一个对象。类似地，不存在一个通用的机制来将一个已登记的公用标识符转换成一个 ASN.1 对象标识符值，使其能标识同一个对象。

6.3.9 引用名类型 Reference Name type

引用名类型的值是名字类型的值加上公用对象标识符类型的值。

6.3.10 字形标识符类型 Glyph Identifier type

字形标识符类型的值是名字类型的值加上结构化名字类型(Structured Name type)的值，定义见 ISO/IEC 9541.2。

注

1 结构化名字可以表示成一个由 ISO/IEC 9541.2 定义的 ASN.1 结构。

2 结构化名字可以作为一个由 ISO/IEC 9541.2 定义的 SGML 结构。

3 上述两种表示与公用标识符(定义见 ISO 9070)经典的字符串形式语义上是相同的。

4 字形标识符的值在 SPDL 内容中用一个类型为“标识符”的对象来表示。

6.4 公用对象标识符的值与结构化名字

公用对象标识符的值用于标识文档表示中用到的信息对象，其中一些信息对象在本标准中定义，另一些定义在别处。结构化名字正如在 ISO/IEC 9541 中所述，用于标识字型属性以及字型资源中的字形(glyphs)。公用对象标识符的值和结构化名字都可以在 SPDL 结构和内容中出现。

6.4.1 公用对象标识符的值

本标准定义公用对象标识符值以标识多个信息对象。对于每个这样的信息对象，本标准用以下两个说明来标识它：

——ASN.1 对象标识符的值，定义见 GB/T 16262；

——公用标识符定义见 ISO 9070。

由本标准定义的公用对象标识符值的各信息对象，在本标准中用一个对象名(object name)来识别。如下一段陈述：

“X 的值具有公用对象标识符类型。公用对象标识符的值所定义的对象名和这些值的含义是……”。其意思是本标准为各个指定的对象名定义了一个对应的公用对象标识符的值，这些值的含义将在上段陈述的余下部分里给出。

ASN.1 对象标识符的值和每个对象名所对应的公用标识符的定义在附录 B 中给出。

6.4.2 公用对象标识符值的表示

选择哪种格式的公用对象标识符值来表示一个在 SPDL 结构的实例中出现的值，取决于所采用的结构交换格式(Structure Interchange Format)。一个赋予对象名的信息对象应如下标识：

——当使用二进制结构交换格式时，一个 ASN.1 对象标识符的值即为指定的 ASN.1 对象标识符的值；

——当采用纯文本结构交换格式时，一个 SGML 规范的公用标识符，语义上等同于由规范正文表示所指定的公用标识符。

用于表示 SPDL 内容实例中的公用对象标识符值的形式依赖于所采用的内容交换格式(Content Interchange Format)一个赋予对象名的信息对象被标识如下：

——当采用二进内容交换格式时，使用 Convert To Identifier 的结果，它带有一个八位字节串的参数，用于表示 ASN.1 对象标识符的值，详见 GB/T 16263 中的定义；

——当采用纯文本内容交换格式时，使用 Convert To Identifier 的结果，带有一个八位字节串的参数，用于表示公用标识符的规范文本，详见 ISO 9070 中的定义。

6.4.3 结构化名字 Structured Names

本标准没有定义结构化名字。

6.4.4 结构化名的表示

用于表示 SPDL 结构实例中的结构化名的形式取决于所采用的结构交换格式。结构化名在结构中的表示为：

——当采用二进制结构交换格式时，使用 ASN.1 结构，定义见 ISO 9070；

——当采用纯文本结构交换格式时，使用 SGML 规范公用标识符，定义见 ISO 9070。

在 SPDL 内容实例中出现的结构化名，可使用 Convert To Identifier 的结果，它带有一个八位字节串的参数，用于表示与结构名相对应的规范文本表示，定义见 ISO 9070。

6.5 结构小结

下表总结了基本的 SPDL 结构元素及其相互关系，每个结构元素的下属(或子结构)用缩进格式列在结构元素下面。

每个结构元素类型的下属只列出了那些最常见的类型。同一结构元素类型的其他出现形式可能有相同的下属。资源定义结构元素和文档生成指令结构元素的下属在定义它们的章条中以类似的表给出。

RESOURCE DEFINITION
DOCUMENT (Top Level PAGESET or PICTURE)
PAGESET
  SPDL IDENTIFIER
  PROLOGUE
    (see subordinates under PICTURE)
PAGE (Top Level PICTURE)
PICTURE
  SPDL IDENTIFIER
  CONTENT TYPE IDENTIFIER
  PROLOGUE
    EXTERNAL DECLARATION
      EXTERNAL IDENTIFIER
      STRUCTURE TYPE IDENTIFER
      STRUCTURE IDENTIFIER
    INFORMATIVE DECLARATION
      HINT
    RESOURCE DEFINITION
      SPDL IDENTIFIER
      REFERENCE NAME
      RESOURCE TYPE IDENTIFIER
      FUNCTION IDENTIFIER
      RESOURCE SPECIFICATION
    RESOURCE DECLARATION
      INTERNAL IDENTIFIER
      RESOURCE TYPE IDENTIFIER
      REFERENCE NAME
    DPI DECLARATION

DPI NAME
DPI VALUE
CONTEXT DECLARATION
DICTIONARY IDENTIFIER
DICTIONARY GENERATIOR
DICTIONARY IDENTIFIER
DICTIONARY SIZE SPECIFIER
TOKENSEQUENCE
SETUP PROCEDURE
TOKENSEQUENCE
PICTURE
(see under PAGESET)
TOKENSEQUENCE
COMMENT(May be subordinate to any structure element)

6.6 高层结构元素

SPDL 文档由三类基本结构元素构成，它们是：

——PICTURE

——PAGESET

——RESOURCE DEFINITION

这些结构元素统称高层结构元素(High Level Structure Element)。

PICTURE 描述一幅可以作为一个单独实体的图像，图像可以是一整页，也可以是整页的一部分。PAGESET 描述零个或多个页面。RESOURCE DEFINITION 定义了可被表示过程引用的资源。

另有两类语义上有区别的高层结构元素类型。

——PAGE，它是一个 PICTURE，且不作为任何其他 PICTURE 的下属。

——DOCUMENT，它是一 PAGESET 或 PICTURE，且不作为任何其他结构元素的下属。

这些结构元素类型语法上与 PICTURE 和 PAGESET 没有区别，但可以通过它们在层次结构中的位置来加以区分。它们也被称为高层结构元素。

还有一个高层结构元素伪类型(pseudo-type)：

——BLOCK，它可以是任何 DOCUMENT、PAGESET、PAGE 或 PICTURE。

每个 BLOCK 可以是一个 DOCUMENT，也可以是其直接上级 BLOCK 的一个直接下属。

每个 BLOCK 有一个当前解释上下文与之相联系，它用于处理下属 TOKENSEQUENCE 结构元素。当前解释上下文与 BLOCK 处理之间的关系在第 9 章定义。

6.6.1 文档

文档不是下属于任何其他结构元素的 PAGESET 或 PAGE。对文档的处理：

——为用户重新设置完成文档的表示机制；

——初始化文档生成参数；

——如果文档是一个 PAGESET，用 6.6.2 中所述方法处理；

——如果文档是一个 PAGE，用 6.6.3 中所述方法处理；

——处理文档中所有结束操作；

——将完成的文档表示提交给用户。

6.6.2 页集

页集(PAGESET)是一个复合结构元素，它具有如下的直接下属：

——一个 SPDL IDENTIFIER 结构元素；

——零个或一个 PROLOGUE 结构元素；

——零个或多个 PAGESET 或 PAGE 结构元素以任意次序构成的序列。

SPDL IDENTIFIER 的值为公用标识符的值，它与对象名字有关。PROLOGUE 结构元素在 6.7 中给出，PAGE 结构元素见 6.6.3。

对 PAGESET 的处理包括：

——用第 9 章中指出的初始值建立一个 PAGESET 的当前解释上下文；

——如果有直接下属 PROLOGUE，则处理之；

——按序列次序处理各个直接下属 PAGESET 和 PAGE；

——撤消 PAGESET 的当前解释上下文。

若 PAGESET 没有下属 PAGE 结构元素，处理的结果将不会产生任何页面图像。

直接下属 PROLOGUE 结构元素的处理见 6.7，直接下属 PAGESET 结构元素的处理见 6.6.2，直接下属 PAGE 结构元素的处理见 6.6.3。

6.6.3 页

页是一个 PICTURE 结构元素，它不作为任何其他 PICTURE 的下属。

注：PAGE 可以简单看作是充满整个页面的一个 PICTURE，在刚开始处理 PAGE 的内容时，内容处理器将从空页开始。

页的处理包括：

——建立一个当前页面图像，它的初值是空页，见第 10 章中所述；

——按 6.6.4 中所述方法处理 PICTURE 结构元素；

——将与 PAGE 对应的当前页面图像描绘到媒体实体上。

若 PAGE 没有直接下属 PICTURE 或 TOKENSEQUENCE 结构元素，媒体实体上将描绘一个空页。

6.6.4 图画

图画是一个复合结构元素，它有以下几个直接下属：

——一个 SPDL IDENTIFIER 结构元素；

——一个 CONTENT TYPE IDENTIFIER 结构元素；

——零个或一个 PROLOGUE 结构元素；

——任意次序的零个或多个 PICTURE 或 TOKENSEQUENCE 结构元素序列。

SPDL IDENTIFIER 的值应是公用对象标识符值，它对应于对象名 SPDL(Object Name SPDL)。PROLOGUE 结构元素见 6.7。

CONTENT TYPE IDENTIFIER 的值具有公用对象标识符类型，公用对象标识符所定义的对象名以及这些值的含义是：

对象名	含　义
Content Type/SPDLBinary	PICTURE 的内容是以二进制方式表示的 SPDL 内容
Content Type/SPDL ClearText	PICTURE 的内容是以纯文本方式表示的 SPDL 内容

每个 TOKENSEQUENCE 结构元素的值具有八位字节串(Octet String)类型，八位字节串被称为是 TOKENSEQUENCE 的内容。

对 PICTURE 的处理包括：

——接第 9 章中给出的初始值建立一个与 PICTURE 相对应的当前解释上下文。

——如果有的话，处理直接下属 PROLOGUE。

——按其序列次序处理每个直接下属 PICTURE 或 TOKENSEQUENCE 结构元素。

——撤消 PICTURE 的当前解释上下文。

对直接下属 PROLOGUE 的处理见 6.7，对直接下属 PICTURE 的处理见本节，对 TOKENSEQUENCE 结构元素的处理见第 9 章。

6.6.5 注释

注释可以出现在 SPDL 实例的任何地方。

注释结构元素的值的类型是可打印字符串(Printable String)，注释对结构处理和内容处理没有任何作用，对注释的处理就是丢弃其值。

6.7 序言结构元素 Prologue Structure Elements

每个 BLOCK 的第一个直接下属都是序言，PROLOGUE 及其所有下属均称为序言结构元素。

每个序言结构元素有一个作用域(scope)，它包括：

——所有在序列次序中紧跟序言结构元素的同级(peer)结构元素；

——所有这些同级结构元素的下属。

除了 PROLOGUE 的直接下属之外，另有几类序言结构元素，它们可以作为下述结构元素的下属：RESOURCE DEFINITION，RESOURCE DECLARATION，DPI DECLARATION 和 INFORMATIVE DECLARATION。

6.7.1 序言

PROLOGUE 是一个复合结构元素，它具有以下直接下属：

——零个或多个 EXTERNAL DECLARATION 结构元素；

——零个或一个 INFORMATIVE DECLARATION 结构元素；

——零个或多个 RESOURCE DEFINITION 结构元素；

——零个或多个 RESOURCE DECLARATION 结构元素；

——零个或多个 DPI DECLARATION 结构元素；

——零个或一个 CONTEXT DECLARATION 结构元素；

——零个或一个 DICTIONARY GENERATOR 结构元素；

——零个或多个 SETUP PROCEDURE 结构元素。

这些结构元素应按指定次序出现。

对 PROLOGOE 结构元素的处理，即按出现的次序处理每个直接下属。

6.7.2 信息说明

INFORMATIVE DECLARATION 结构元素提供关于 BLOCK 的信息数据。这种信息是以提示(hint)的方式给出的。提示并不要求准确和完整，但一定要是有用的信息。提示的例子如(1)PICTURE 处理过程中所画图像的最大范围(如边框 bounding box)。(2)字型或其他所用资源列表。

INFORMATIVE DECLARATION 是一个复合结构元素，它的直接下属是：

——零个或多个 HINT 结构元素。

HINT 是一个复合结构元素，它的直接下属是：

——一个 HINT NAME 结构元素；

——一个 HINT VALUE 结构元素。

这些结构元素按指定次序出现。

HINT NAME 的值具有引用名类型(Reference Name type)，HINT VALUE 的值依赖于 HINT NAME 值所标识的提示(hint)。

INFORMATIVE DECLARATION 结构元素不影响文档表示。它不需要任何处理。

6.7.3 上下文说明 CONTEXT DECLARATION

CONTEXT DECLARATION 结构元素是 EXTERNAL IDENTIFIER 结构元素的一个有序列表，表中各元素与 CONTEXT DECLARATION 的上下文中定义的上下文词典(Context Dictionary)密切相关。CONTEXT DECLARATION 改变最直接上级 BLOCK 的当前解释上下文中的上下文栈的状态。上

下文栈见第 9 章中的定义。

CONTEXT DECLARATION 是一个复合结构元素，它具有以下直接下属：

——零个或多个 DICTIONARY IDENTIFIER 结构元素。

DICTIONARY IDENTIFIER 的值具有名字(Name)类型。

设 CONTEXT DECLARATION 由 n 个 DICTIONARY IDENTIFIER 结构元素组成，其值分别为 name 1. ……name n，若对每个 i，名字 name i 与 CONTEXT DECLARATION 上下文中的上下文词典 Dicti 相对应，且 SystemDict 和 UserDict 为第 10 章中定义的上下文词典，则对 CONTEXT IDENTIFIER 的处理就是将最直接上级 BLOCK 的当前解释上下文的上下文栈替换成：

〈Dictn：Dictionary〉

……

〈Dict1：Drctionary〉

〈UserDict：Dictionary〉

〈SystemDict：Dictionary〉

若没有一个 DICTIONARY IDENTIFIER 的值与 CONTEXT DECLARATION 上下文中的一个词典相对应，则结构处理器将产生一个异常。

6.7.4 词典生成器 DICTIONARY GENERATOR

每个 DICTIONARY GENERATOR 结构元素定义了单个上下文词典，并且赋予这个词典一个名字。上下文词典在 DICTIONARY GENERATOR 定义以后是不可修改的。

注：术语“词典”(Dictionary)定义见 10.1.4，术语“上下文词典”(Contex Dictionary)不是泛指上下文栈中的任一词典，而是显式地指向一个由 DICTIONARY GENERATOR 建立的词典。需要区别时，术语“非上下文词典”(non-context Dictionary)用以指明上下文词典以外的任何词典。

DICTIONARY GENERATOR 为一个复合结构元素，其直接下属为：

——一个 DICTIONARY IDENTIFIER 结构元素；

——一个 DICTIONARY SIZE SPECIFIER 结构元素；

——零个或多个 TOKENSEQUENCE 结构元素。

这些结构元素按指定次序出现。

DICTIONARYIDENTIFIER 的值为名字类型，DICTIONARYSIZE 结构元素值为整型，TOKENSEQUENCE 结构元素定义见第 9 章。

对 DICTIONARY GENERATOR 的处理包括：

——为处理下属 TOKENSEQUENCE 结构元素建立一个解释上下文。解释上下文的值包括：

——最直接上级 BLOCK 的当前解释上下文的资源的集合；

——最直接上级 BLOCK 的当前解释上下文的上下文栈；

——一个包括单个 Dictionary Reference 对象的操作数栈，而该对象指向一个大小为 DICTIONARY SIZE SPECIFIER 值的空词典；

——最直接上级 BLOCK 的当前解释上下文的状态变量的值。

——调用内容处理器依次处理每个 TOKENSEQUENCE：

——处理第一个 TOKENSEQUENCE 内容的解释上下文就是在第一步中所建立的解释上下文；

——处理每一个后继 TOKENSEQUENCE 内容的解释上下文就是内容处理器在完成了前一个 TOKENSEQUENCE 处理以后所返回的解释上下文。

在所有的直接下属 TOKENSEQUENCE 结构元素处理完以后：

——由内容处理器返回的虚拟机最后状态的上下文栈与送给内容处理器的解释上下文的上下文栈完全相同。

——由内容处理器返回的虚拟机最后状态的操作数栈由单个 DictionaryReference 组成，它与第一次开始处理 TOKENSEQUENCE 时操作数栈中由 DictionaryReference 所引用的词典相同。

这词典指向一个由 DICTIONARY GENERATOR 所定义的上下文词典。

——将由 DICTIONARY GENERATOR 所定义的上下文词典压入上下文栈，从而改变最直接上级 BLOCK 的当前解释上下文。

——将由 DICTIONARY GENERATOR 所定义的上下文词典与 DICTIONARY IDENTIFIER 的值对应起来。

处理 DICTIONARY GENERATOR 的结果将不会改变当前页面图像。

注：词典的大小就是词典可以容纳的名字-值对的数目。

6.7.5 SETUP PROCEDURE

SETUP PROCEDURE 结构元素通过初始化其最直接上级 BLOCK 的当前解释上下文中的状态变量，来对其作用域内的内容解释上下文起作用，它也可改变词典内容。

SETUP PROCEDURE 是一个复合结构元素，其直接下属包括：

——零个或多个 TOKENSEQUENCE 结构元素。

TOKENSEQUENCE 结构元素的定义见第 9 章。

对 SETUP PROCEDURE 的处理包括：

——为处理下属 TOKENSEQUENCE 结构元素建立一个解释上下文，其初始值由最直接上级 BLOCK 的当前解释上下文组成。

——调用内容处理器，依次处理每一个 TOKENSEQUENCE 结构元素的内容。

——与第一个 TOKENSEQUENCE 相对应的解释上下文，就是在第一步中所建立的解释上下文。

——与每一个后继 TOKENSEQUENCE 相对应的解释上下文，就是内容处理器在处理完前一个 TOKENSEQUENCE 以后所返回的解释上下文。

——通过以下一些步骤来改变最直接上级 BLOCK 的当前解释上下文：

——用内容处理器在完成了最后一个直接下属 TOKENSEQUENCE 的处理以后所返回的解释上下文的状态变量的值来取代状态变量；

——用新值来取代当前解释上下文中的任何非-上下文(non-context)词典，而该字典在处理直接下属 TOKENSEQUENCE 结构元素的过程中已经被修改过。

处理 SETUP PROCEDURE 的结果并不改变当前页面图像。

注

1 对 SETUP PROCEDURE 的处理可能改变用于解释直接下属 TOKENSEQUENCE 结构元素的解释上下文的上下文栈的内容。这些改变不影响非直接下属于 SETUP PROCEDURE 的 TOKENSEQUENCE 结构元素处理过程中所用的解释上下文的上下文栈的内容。

2 由于高层结构边界的状态重新设定这一特性，PAGESET 的 PROLOGURE 中的一个 SETUP PROCEDURE，可以为直接下属于 PAGESET 的所有 PAGE 结构元素的解释上下文设定初始状态。

6.8 外部结构元素 External Structure Elements

外部结构元素是 SPDL 文档的一部分，它由单个结构元素组成，在表示过程的环境中使用。外部结构元素可以在任何 SPDL 文档结构中被引用，引用外部结构元素的一个 SPDL 文档必须以一个 EXTERNAL DECLARATION 对它加以说明。

在 EXTERNAL DECLARATION 的作用域内，由 STRUCTURE IDENTIFIER 的值所标识的外部结构元素，可以通过引用从而在词法上使它包含在结构内部，其方法是将 EXTERNAL IDENTIFIER 放置在结构中相应外部结构元素可以访问的任何位置上，结构处理器通过用外部结构元素来替换 EX-

TERNAL IDENTIFIER 的方法实现引用。

本标准的结构元素和它们下属的定义并没有进一步注意到这样一个事实，即适当说明的 EXTERNAL IDENTIFIER 可以替换任何指定的结构元素。结构处理机将接受这种用法，并执行上面所指定的替换。

6.8.1 外部说明 EXTERNAL DECLARATION

每一个 EXTERNAL DECLARATION 把一个 EXTERNAL IDENTIFIER 与一个外部结构元素联编起来。

EXTERNAL DECLARATION 是复合结构元素，它包括如下一些直接下属：

——一个 EXTERNAL IDENTIFIER 结构元素；

——一个 STRUCTURE TYPE IDENTIFIER 结构元素；

——一个 STRUCTURE IDENTIFIER 结构元素。

这些结构元素按照指定的次序出现。

EXTERNAL IDENTIFIER 的值具有名字类型。STRUCTURE TYPE IDENTIFIER 定义见 6.8.1.1，STRUCTURE IDENTIFIER 结构元素定义见 6.8.1.2。

对 EXTERNAL DECLARATION 的处理包括在其作用域内将 EXTERNAL IDENTIFIER 与 STRUCTURE TYPE IDENTIFIER 的值和 STRUCTURE IDENTIFIER 进行联编(binding)。

一个 EXTERNAL DECLARATION 出现下述情况之一时：

——由 STRUCTURE IDENTIFIER 的值所标识的外部结构元素在表示过程环境中不能使用；

——STRUCTURE IDENTIFIER 值的语法并不为结构处理器所理解。

并不一定产生结构异常。结构异常仅仅在当 EXTERNAL IDENTIFIER 被用在其他地方去引用一个外部结构元素时才产生。

若一个 EXTERNAL DECLARATION 刚好在它前一个 EXTERNAL DECLARATION 的作用域内，且 EXTERNAL IDENTIFIER 的值相同，则后一个 EXTERNAL DECLARATION 将前一个置于自己的作用域内。

6.8.1.1 结构类型标识符 STRUCTURE TYPE IDENTIFIER

STRUCTURE TYPE IDENTIFIER 是一个整型数，它标识由 EXTERNAL DECLARATION 所引用的一个外部结构元素的类型。STRUCTURE TYPE IDENTIFIER 的值和结构元素类型对应关系如下：

值	结构元素类型	值	结构元素类型
1	PAGESET	11	INFORMATIVE DECLARATION
2	PICTURE	12	STRUCTURE IDENTIFIER
4	PROLOGUE	13	RESOURCE REFERENCE
5	RESOURCE DECLARATION	14	RESOURCE SPECIFICATION
6	DPI DECLARATION	15	FONT OBJECT REFERENCE
7	CONTEXT DECLARATION	16	FONT REFERENCE
8	DICTIONARY GENERATOR	17	PROPERTY LIST
9	SETUP PROCEDURE	18	PROPERTY
10	TOKENSEQUENCE	19	RUN VECTOR

6.8.1.2 结构标识符 STRUCTURE IDENTIFIER

STRUCTURE IDENTIFIER 的值为公用对象标识符类型或八位字节串类型。

STRUCTURE IDENTIFIER 为结构处理器标识一个外部结构元素。一个公用对象标识符类型的值标识一个特定的外部结构元素。八位字节串类型的值必须为结构处理器访问外部结构元素提供需要

的任何信息，且可以包括与系统有关的信息。这些信息的语法已超出了本标准的范围，并且与系统有关。

注：若 STRUCTURE IDENTIFIER 的值由 专用数据组成，则为了进一步的交换必须替换某些数据，COMMENT 可以用来帮助用户或处理过程执行这种替换。

6.8.2 对一个不是 EXTERNAL DECLARATION 下属的 EXTERNAL IDENTIFIER 的处理。

在一个 EXTERNAL DECLARATION 中说明的 EXTERNAL IDENTIFIER 可以在文档结构的任何地方使用，只要在该处由 EXTERNAL DECLARATION 所指定类型的外部结构元素是有效的。如果在一个 EXTERNAL DECLARATION 的作用域内，把一个 EXTERNAL IDENTIFIER 赋给一个外部结构元素，则 EXTERNAL IDENTIFIER 在该作用域内的结构元素中出现的效果就是用该外部结构元素来取代 EXTERNAL IDENTIFIER。结构处理器通过用外部结构元素替换 EXTERNAL IDENTIFIER 来实现这一引用。外部结构元素将被结构处理器当作一个 SPDL 文档的原有部分进行处理。

对一个非 EXTERNAL IDENTIFIER 的直接隶属于一个结构元素的 EXTERNAL IDENTIFIER 的处理过程包括：

——标识最直接上级 EXTERNAL DECLARATION，其中的 EXTERNAL IDENTIFIER 的值与当前 EXTERNAL IDENTIFIER 的值相同；

——取得由 STRUCTURE TYPE IDENTIFIER 的值所标识的外部结构元素；

——确认外部结构元素具有由 STRUCTURE TYPE IDENTIFIER 指定的类型；

——用外部结构元素取代 EXTERNAL IDENTIFIER。

以上任何一步失败时，结构处理器将产生一个异常。

7 资源

7.1 概述

资源(Resources)是可以在表示过程环境中使用的一类信息对象。资源可以用本标准所给的方法进行定义，也可以用本标准范围之外的方法来使之在表示过程中有效。

本章定义了用来说明和定义资源结构元素的结构和处理过程。资源本身的语义在说明资源使用方法的相应章条中介绍。

7.1.1 资源类型

本标准定义的资源类型有：

——FontObject （字型对象）
——GlyphIndexMap （字形索引映射表）
——FontIndexMap （字型索引映射表）
——ColorSpace （彩色空间）
——DataSource （数据源）
——Filter （过滤器）
——Pattern （网纹）
——Form （图案）

对每一个类型本标准给出了使用的方法以及产生的结果。

7.1.2 资源说明

使用资源的 SPDL 文件必须在 RESOURCE DECLARATION 中加以说明。RESOURCE DECLARATION 使得资源可用于内容处理器，使之成为 RESOURCE DECLARATION 作用域中的任何 TOKENSEQUENCE 解释上下文的一部分。

RESOURCE DECLARATION 将 Name 与资源联编(bind)，以使资源能在 SPDL 内容处理过程中被引用。这类资源在文件内容中可通过 Find Resource 操作符来引用。其参数就是解释上下文中与资源联编的名字。

7.1.3 资源定义

资源可以用 RESOURCE DEFINATION 结构元素来定义。它可以出现在层次结构的如下两处：

——DOCUMENT 的下属层；

——最高结构层。

由隶属于 DOCUMENT 的 RESOURCE DEFINITION 定义资源的作用域即是 RESOURCE DEFINITION 定义的作用域。由 RESOURCE DEFINITION 定义的且出现在最高结构层的资源的作用域与系统有关。

资源由名字或者公用对象标识符值来标识。由 RESOURCE DEFINITION 定义的且属于一个 DOCUMENT 的资源，可以由名字或者公用对象标识符值来标识。在表示过程环境下的资源将由公用对象标识符值来标识。

7.1.4 资源结构元素小结

下表列出了用于资源说明与资源定义的结构元素。

RESOURCE DECLARATION(资源说明)
  INTERNAL IDENTIFIER
  RESOURCE TYPE IDENTIFIER
  RESOURCE IDENTIFIER
RESOURCE DEFINITION(资源定义)
  SPDL IDENTIFIER
  RESOURCE IDENTIFIER
  RESOURCE TYPE IDENTIFIER
  FUNCTION IDENTIFIER
  RESOURCE SPECIFICATION
  —FONT OBJECT RESOURCE SPECIFICATION
    —FONT REFERENCE BASE FONT SPECIFICATION
      FONT REFERENCE
      GLYPH INDEX MAP IDENTIFIER
    —FONT OBJECT BASE FONT SPECIFICATION
    —NAMED BASE FONT SPECIFICATION
      FONT RESOURCE IDENTIFIER
      GLYPH INDEX MAP IDENTIFIER
    —COMPOSITE FONT SPECIFICATION
      FMAPTYPE
      FMAPTYPE PARAMETERS LIST
      FONT INDEX MAPIDENTIFIER
      FONT OBJECT LIST
        FONT OBJECT RESOURCE SPECIFICATION
    —CONSTRUCTED FONT SPECIFICATION
      TOKENSEQUENCE
    —GLYPH INDEX MAP RESOURCE SPECIFICATION
      MAP SIZE
      GLYPH IDENTIFIER LIST
    —FONT INDEX MAP RESOURCE SPECIFICATION

MAP SIZE

INDEX LIST
—COLOR SPACE RESOURCE SPECIFICATION
COLOR SPACE TYPE
PRIMARY SET IDENTIFIER
COLOR SPACE SPECIFICATION
—PATTERN RESOURCE SPECIFICATION
TOKENSEQUENCE
—FORM RESOURCE SPECIFICATION
TOKENSEQUENCE

7.2 资源说明 RESOURCE DECLARATION

7.2.1 结构定义

RESOURCE DECLARATION 是一个复合结构元素，它可有如下直接下属：

——一个 INTERNAL IDENTIFIER 结构元素；

——一个 RESOURCE TYPE IDENTIFIER 结构元素；

——一个 RESOURCE IDENTIFIER 结构元素。

它们将以指定次序出现。

7.2.1.1 内部标识符

INTERNAL IDENTIFIER 结构元素的值为 Name 类型。

7.2.1.2 资源类型标识符

RESOURCE TYPE IDENTIFIER 的值是 Integer 类型。其值的含义如下：

值	资源类型	
1	Font Object	(字型对象)
2	Glyph Index Map	(字形索引映射表)
3	Font Index Map	(字型索引映射表)
4	Color Space	(彩色空间)
5	Data Source	(数据源)
6	Filter	(过滤器)
7	Pattern	(网纹)
8	Form	(图案)

7.2.1.3 资源标识符

RESOURCE IDENTIFIER 的值是引用名(Reference Name)类型。

7.2.2 处理

对 RESOURCE DECLARATION 的处理步骤：

——在表示过程环境中标识的资源：

——其类型由 RESOURCE TYPE IDENTIFIER 的值所指定；

——由 RESOURCE IDENTIFIER 的值来标识。

——将直接下属 INTERNAL IDENTIFIER 的值与 RESOURCE DECLARATION 作用域内的资源相联编。

——使资源在最直接上级 BLOCK 的当前解释上下文中有效。

若在 RESOURCE DECLARATION 的上下文中有一个 RESOURCE DEFINITION，它定义了一个

给定类型的资源，并赋以 RESOURCE IDENTIFIER 的值作为标识符，则 INTERNAL NAME 的值将与最直接的上级（如 RESOURCE DEFINITION）所定义的资源联编。

若某 RESOURCE DECLARATION 在其前一个 RESOURCE DECLARATION 的作用域内，且两者对同一类型的不同资源给出了相同的名（Name），则后者在它自己的作用域内将取代前者。

一个 RESOURCE DECLARATION 如果不存在可满足第一步中条件的资源时，资源说明不会产生一个结构处理错误。但是，对 RESOURCE DECLARATION 的处理就不能使得 INERNAL NAME 的值与一个资源联编。因此以后在处理一个 TOKENSEQUENCE 的过程中，任何通过这一名字对资源的引用将产生一个内容处理异常。

注：RESOURCE DECLARATION 只将在 RESOURCE DECLARATION 作用域内的资源与一个名字联编。TOKENSEQUENCE 结构元素中，处在 RESOURCE DECLARATION 作用域之外的内容不可使用这一联编。任何 TOKENSEQUENCE 中的内容用一个未与一个合适资源联编的名来引用资源时，将产生一个内容处理异常。

7.3 资源定义 RESOURCE DEFINITION

7.3.1 结构定义

RESOURCE DEFINITION（资源定义）是一个复合结构元素，它有如下一些下属：

——一个 SPDL IDENTIFIER 结构元素；

——一个 RESOURCE IDENTIFIER 结构元素；

——一个 RESOURCE TYPE IDENTIFIER 结构元素；

——零个或一个 FUNCTION IDENTIFIER 结构元素；

——零个或一个 RESOURCE SPECIFICATION 结构元素。

这些结构元素以指定的次序出现。

7.3.1.1 SPDL 标识符

SPDL IDENTIFIER（SPDL 标识符）的值是对应于对象名 SPDL 的公用对象标识符的值。

7.3.1.2 资源标识符

RESOURCE IDENTIFIER（资源标识符）的值的类型是引用名（Reference Name）。

7.3.1.3 资源类型标识符

RESOURCE TYPE IDENTIFIER（资源类型标识符）的值和所标识资源的类型见 7.2.1 中的定义。

7.3.1.4 功能标识符

FUNCTION IDENTIFIER（功能标识符）在当 RESOURCE DEFINITION（资源定义）出现在最高结构层时才出现；当 RESOURCE DEFINITION 是文件的下属时，它就不再出现。出现时，FUNCTION IDENTIFIER 的值是 Integer 型，它表示由 RESOURCE DEFINITION 所执行的功能。FUNCTION IDENTIFIER 的值与它们标识的功能如下：

值（Value）	功能（Function）
0	定义资源
1	不定义资源

若 FUNCTION IDENTIFIER 不出现，其默认值为 0。若 FUNCTION IDENTIFIER 出现，其值为 1，则在 RESOURCE DEFINITION 中将不存在 RESOURCE SPECIFICATION（资源描述）。

7.3.1.5 资源描述

RESOURCE SPECIFICATION（资源描述）结构元素的类型依赖于所定义的资源的类型。资源描述结构元素应是如下之一：

——FONT OBJECT RESOURCE SPECIFICATION 结构元素；

——GLYPH INDEX RESOURCE SPECICFIATION 结构元素；

——FONT INDEX MAP RESOURCE SPECIFICATION 结构元素；

——COLOR SPACE RESOURCE SPECIFICATION 结构元素；

——PATTERN RESOURCE SPECIFICATION 结构元素；

——FORM RESOURCE SPECIFICATION 结构元素；

资源描述结构元素在 7.4 中给出。

7.3.2 处理一个属于 DOCUMENT 的 RESOURCE DEFINITION

这一处理过程包括：

——解释直接下属 RESOURCE SPECIFICATION，以定义一个由 RESOURCE IDENTIFIER 给出了类型的资源；

——将直接下属 RESOURCE IDENTIFIER 的值赋给资源作为标识符；

——将资源放置在 RESOURCE DEFINITION 作用域的表示过程环境中。

结构处理器在如下情况发生时将产生一个异常：即当 RESOURCE SPECIFICATION 的类型与由 RESOURCE TYPE IDENTIFIER 的值给出的资源类型不相一致时，或当 FUNCTION IDENTIFIER 的值为 1，且又给出了 RESOURCE SPECIFICATION 结构元素时。

7.3.3 处理一个出现在最高结构层的 RESOURCE DEFINITION

这一处理过程依赖于系统有关的管理策略和资源的有效性。它可能包括：

——按后面几段所述的方法去处理 RESOURCE DEFINITION；

——忽略 RESOURCE DEFINITION。

这一处理可能因 RESOURCE DEFINITION 的不同而不同。

出现在最高结构层的 RESOURCE DEFINITION 的处理是否被忽略则依赖于 FUNCTION IDENTIFIER 的值。

——若 FUNCTION IDENTIFIER 的值为 0，则处理 RESOURCE DEFINITION 的过程为：

——解释 RESOURCE SPECIFICATION，以定义由 RESOURCE TYPE IDENTIFIER 给出类型的资源；

——将 RESOURCE IDENTIFIER 的值赋给资源以作为一个全局标识符；

——将此资源加入到表示过程的环境中去。

——若在表示环境中存在另一个具有相同类型的资源并且用同一引用名来标识，那么这一资源将由 RESOURCE SPECIFICATION 所定义的资源来代替。

——若 FUNCTION IDENTIFIER 的值是 1，则 RESOURCE DEFINITION 的作用是在表示过程的环境中删除满足下列条件的任何资源：

——资源类型是由 RESOURCE TYPE IDENTIFIER 的值所给出的类型；

——资源由引用名来标识，而该引用名为 RESOURCE IDENTIFIER 的值。

7.4 RESOURCE SPECIFICATION 结构元素

本条定义 RESOURCE SPECIFICATION（资源描述）结构元素的类型，适用于由本标准所定义的每一类型的资源，并给出相应的处理。

7.4.1 字型对象资源描述 Font Object Resource Specification

Font Object 提供了在一个 SPDL 表示过程中来自字型资源（Font Resource）的用来进行字母描述时所需要的信息（见第 16 章），一个 FONT OBJECT RESOURCE SPECIFICATION（字型对象资源描述）应是如下之一：

——FONT REFERENCE BASE FONT SPECIFICATION 结构元素；

——FONT OBJECT BASE FONT SPECIFICATION 结构元素；

——NAMED BASE FONT SPECIFICATION 结构元素；

——COMPOSITE FONT SPECIFICATION 结构元素；

——CONSTRUCTED FONT SPECIFICATION 结构元素。

注：FONT OBJECT RESOURCE SPECIFICATION 的定义是递归的。见 7.4.1.4。

7.4.1.1 字型引用基字型描述 Font Reference Base Font Specification

FONT REFERENCE BASE FONT SPECIFICATION 是一个复合结构元素，它有两个直接下属：

——一个 FONT REFERENCE 结构元素；

——一个 GLYPH INDEX MAP IDENTIFIER 结构元素。

FONT REFERENCE 的值是 ISO/IEC 9541-2 中所定义的字型引用。GLYPH INDEX MAP IDENTIFIER 具有类型为引用名的值。对 FONT REFERENCE FONT SPECIFICATION 的处理包括：

——标识出表示过程环境中最能满足字型引用(它是 FONT REFERENCE 的值)的字型资源；

——标识出表示过程环境中由 GLYPH INDEX MAP IDENTIFIER 标识的字形索引映射；

——定义一个对应于基本字型词典的 Font Object 词典。它与第一步中指出的字型资源和第二步中指出的字形索引映射完全一致，正如第 16 章所指出的那样。

标识表示过程环境中最能满足字型引用的字型资源的过程依赖于字型引用是否为字型资源名给定一个值。字型资源名即是 FONTNAME 的特性值，见 ISO/IEC 9541.1 中的定义。

注：FONTNAME 特性用结构化名来命名，其值是：

ISO(1)Standard(0)9541(9541)/FONTNAME。这规范字符串表示的公用标识符，语义上它等于另一个结构化名“ISO/IEC 9541.1/FONTNAME”。

若字型引用给出了字型资源名的一个值，且在表示过程环境中存在一个字型资源，其中字型资源名的值与由字型引用所指出的字型资源名的值相同，则此字型资源就可以认为是最能满足字型引用要求的字型资源。

若字型引用没有给出字型资源名的值，或者在表示过程环境中不存在这样一个字型资源，其字型资源名的值与由字型引用所指出的字型资源名的值相同，则选择最能满足字型引用要求的字型资源与系统有关。

若表示过程环境中不存在由 GLYPH INDEX MAP IDENTIFIER 所标识的字形索引映射，则将产生一个异常，并且将使用系统有关的字形索引映射。

注：若想要的字型资源或者字形索引映射未找到，那么选中的字型资源就可能不包括在字形索引映射表中由字形标识符所标识的那些字形，或者所产生的字形图像并不是文件创建者想要的内容。

7.4.1.2 字型对象基字型描述 Font Object Base Font Specification

FONT OBJECT BASE FONT SPECIFICATION 的值是一个八位字节串，它表示一个 SPDL 字体对象(Font Object)，见本标准附录 C 中的定义。对它的处理就是去解释一个定义 FontObject 的八位字节串。

7.4.1.3 命名基字型描述 Named Base Font Specification

NAMED BASE FONT SPECIFICATION 是一个复合结构元素，它有如下下属：

——一个 FONTOBJECT IDENTIFIER 结构元素；

——零个或一个 GLYPH INDEX MAP IDENTIFIER 结构元素。

FONT OBJECT IDENTIFIER 和 GLYPH INDEX MAP IDENTIFIER 的值是引用名类型。

如果存在一个 GLYPH INDEX MAP IDENTIFIER，那么对 NAMED BASE FONT SPECIFICATION 的处理如下：

——标识一个在表示过程环境中由 FONT OBJECT IDENTIFIER 的值指出的基本 Font Object；

——标识一个在表示过程环境中由 GLYPH INDEX MAP IDENTIFIER 的值指出的字形索引映射表；

——定义一个对应于基字型词典的 FontObject 词典，它可以通过将由第二步中所标识的 FontObject 的字形索引映射来完成。

如果 GLYPH INDEX MAP IDENTIFIER 并不存在，则对 NAMED BASE FONT SPECIFICATION 的处理，就是在表示过程环境中标识出由 FONT OBJECT IDENTIFIER 所标识的 FontObject。

如果在表示过程环境中，并不存在由 FONT OBJECT IDENTIFIER 标识的 FontObject，则 FontObject 的选择将与系统有关。如果被标识的 FontObject 是一个复合的 FontObject 而不是一个基本的 FontObject，且 GLYPH INDEX MAP IDENTIFER 被忽略。如果表示过程环境中并不存在由 GLYPH INDEX MAP IDENTIFER 的值所标识的字形索引映射，则将出现一个异常，且使用与系统有关的字形索引映射。

注：如果想要的 FontObject 或字形索引映射未找到，则选中的 FontObject 就可能不包含在所用的字形索引映射中由字形标识符所标识的所有字形，或者生成的字形图像并不是文件创建者想要的内容。

7.4.1.4 复合字型描述 Composite Font Specification

COMPOSITE FONT OBJECT SPECIFICATION 是一个复合结构元素，它有以下一些直接下属：

——一个 FMAPTYPE 结构元素；

——零个或一个 FMAPTYPE PARAMETERS LIST 结构元素；

——一个 FONT INDEX MAP IDENTIFIER 结构元素；

——一个 FONT OBJECT LIST 结构元素。

直接下属的值的类型和结构元素的处理在以下一些条中叙述。

7.4.1.4.1 FMAPTYPE

FMAPTYPE 的值是一个 INTEGER 类型，它的值是标识复合字型字形索引映射算法的一组值中的一个。

7.4.1.4.2 FMAPTYPE 参数表

FMAPTYPE PARAMETERS LIST 的值是一个复合结构元素，它有以下一些直接下属：

——零个或一个 ESCCHAR 结构元素；

——零个或一个 SHIFTOUT 结构元素；

——零个或一个 SHIFTIN 结构元素；

——零个或一个 SUBSVECTOR 结构元素。

ESCCHAR 结构元素的值是一个在 0～255 之间的基数。如果不存在 FMAPTYPE PARAMETER LIST，或者不存在有直接下属 ESCCHAR 结构元素，则默认值为 14。

SHIFTIN 结构元素的值是一个在 0～255 之间的基数。如果不存在 FMAPTYPE PARAMETER LIST，或者不存在直接下属 SHIFTIN 结构元素，则默认值为 15。

SUBSVECTOR 结构元素的值是八位字节串(Octetstring)类型，其语义在 16.2.1.2.2.4 中定义。如果 FMAPTYPE 的值为 6，则 FMAPTYPE PARAMETER LIST 和 SUBSVECTOR 结构元素是必须的。如果两者中有一个不存在，则会出现非法的描述结构异常。如果 FMAPTYPE 的值不是 6，则 SUBSVECTOR 结构元素是可选的，且当它存在时也可以不管它。

7.4.1.4.3 字型索引映射标识符 Font Index Map Identifier

FONT INDEX MAP IDENTIFIER 的值是引用名类型。

7.4.1.4.4 字型对象表 Font Object List

FONT OBJECT LIST 是一个复合结构元素，它有下列直接下属：

——零个或多个 FONT OBJECT RESOURCE SPECIFICATION 结构元素。

FONT OBJECT RESOURCE SPECIFICATION 结构元素在 7.4 中定义。

7.4.1.4.5 处理 Processing

COMPOSITE FONT SPECIFICATION 的处理包括以下内容：

——在表示过程环境中标识一个由 FONT INDEX MAP IDENTIFIER 的值指出的字型索引映射表；

——建立一个对应于复合字型词典的复合 FontObject 词典，见第 16 章中的定义，有以下一些内容：

——FontType 的值是〈0:Integer〉；

——Fontmatrix 的值是恒等变换；

——FMapType 的值是 FMAPTYPE 结构元素的值；

——EscChar，ShiftOut，ShiftIn 和 SubsVector 等的值由 FMAPTYPE PARAMETER LIST 结构元素中对应的值来指定；

——编码值是对由 FONT INDEX MAP IDENTIFIER 的值所标识的字型索引映射表的引用；

——FDepVector 的值是对 FontObject 的向量的引用，这里的 FontObjects 是通过轮流地处理每一个 FONT OBJECT RESOURCE SPECIFICATION 结构元素来产生的，而这些结构元素又直接隶属于 FONT OBJECT LIST。

如果在表示过程环境中不存在由 FONT INDEX MAP IDENTIFIER 所标识的字型索引映射表，则选择字型索引映射将与系统有关。

注：若想要的字型索引映射没有找到，FontObject 生成的编码和 FDepVector 两者之间的关系是不相宜的，且使用该 FontObject 表示字符文本时所生成的字形图像也并不是文件创建者希望要的内容。若下属于 FONT OBJECT LIST 的 FONT OBJECT RESOURCE SPECIFICATION 结构元素的数目小于字型索引映射表中最大的索引值，且如果在使用复合 FontObject 过程中应用了一个较大的字型索引值，则会引发内容处理异常，正如第 16 章所述。

7.4.1.5 构造字型描述 Constructed Font Specification

CONSTRUCTED FONT SPECIFICATION 的值是一个复合结构元素，它有以下一些直接下属：

——一个或者多个 TOKENSEQUENCE 结构元素。

TOKENSEQUENCE 结构元素在第 9 章中定义。处理 CONSTRUCTED FONT SPECIFICATION 有如下几步：

——建立一个处理 TOKENSEQUENCE 子结构元素的解释上下文。

——若最直接的上级结构元素 RESOURCE DEFINITION 是在最高结构层，则解释上下文包括以下一些内容：

(1) 一个可用资源的空集；

(2) 由 9.3.2 定义的缺省上下文栈；

(3) 由第 12 章定义的状态变量的初始值；

(4) 一个空的操作数栈。

——若最直接的上级结构元素 RESOURCE DEFINITION 隶属于一个 DOCUMENT，则解释上下文与最直接上级 BLOCK 的当前解释上下文完全相同。

——对于每一个 TOKENCENCE 结构元素依次调用内容处理器。

——与处理第一个 TOKENSEQUENCE 相关的解释上下文就是在第一步中所建立的解释上下文；

——处理每一个下属 TOKENSEQUENCE 的解释上下文就是内容处理器在处理先前的 TOKENSEQUENCE 的内容完成后返回的解释上下文。

在处理直接下属 TOKENSEQUENCE 的末尾：

——由内容处理器返回的虚拟机最后状态的上下文栈与解释上下文送给内容处理器的上下文栈完全一致；

——由内容处理器返回的虚拟机最后状态的操作数栈将组成一个指向 FontObject 的字典引用(DictionaryReference)。

——定义一个对应于 FontObject 词典的 FontObject。

解释最直接上级 BLOCK 的当前上下文，或者作为处理一个 CONSTRUCTED FONTSPECIFICATION 结果的当前页面图像将不作任何改变。

7.4.2 字形索引映射资源描述 Glyph Index Map Resource Specification

GLYPH INDEX MAP RESOURCE SPECIFICATION 的值是一个复合结构元素，它有以下一些直接下属组成：

——一个 MAP SIZE 结构元素；

——零个或一个 GLYPH IDENTIFIER LIST 结构元素。

MAP SIZE 的值的类型为 INTEGER。GLYPH IDENTIFIER LIST 表示一列字形标识符的值。这一序列的长度为 MAP SIZE 结构元素的值。

GLYPH IDENTIFIER LIST 是复合结构元素，其直接子结构包括：

——零个或多个 GLYPH IDENTIFIER 结构元素。

GLYPH IDENTIFIER 结构元素的值的类型是字形标识符，或是 NULL。

对 GLYPH INDEX MAP RESOURCE SPECIFICATION 的处理包括建立一个字形索引映射向量，定义见第 16 章，对每个介于 0 至 MAPSIZE 减 1 之间的下标值 n，相应的字形索引映射向量中下标为 n 的元素依赖于 GLYPH IDENTIFIER LIST，如下：

——若没有给定 GLYPH IDENTIFIER LIST，则字形索引映射向量的所有元素的值均为默认值“.notdef”。

——若第 n+1 个 GLYPH IDENTIFIER 的值为 NULL，或者 n+1 大于 GLYPH IDENTIFIER LIST 的长度，则字形索引映射向量中下标为 n 的元素的值为“.notdef”。

——若第 n+1 个 GLYPH IDENTIFIER 的值为类型 Name，则字形索引映射向量中下标为 n 的元素的值是类型为 Name 的对象，对象的值与 GLYPH IDENTIFIER 的值相同。

——若第 n+1 个 GLYPH IDENTIFIER 的值是一个结构名，即它是 ISO/IEC 9541.1:1991 之 6.2 所定义的 ISO 10036 字形名，则字形索引映射向量的下标为 n 的元素的值为一类型为 Name 的对象，其值是“afii:nnnn”，其中 nnnn 是非零数字开头的字符串的十六进制表示形式（“0”…“9”/“a”…“f”）。这个字符串代表了所标识字形的“ISO 10036 glyph local name”的值。

——若第 n+1 个 GLYPH IDENTIFIER 的值代表一个非上面所述的结构化名，则下标为 n 的字形索引映射向量的元素的值是类型为 Identifier 的对象，其值是(String Convert To Identifier)的结果，此处 string 为公用标识符的规范串形式，语义等同于 ISO 9070 中定义的结构化名的值。

注：字形索引映射向量的元素是内容数据对象。内容数据对象的类型(用斜体印刷的类型名来表示，如 Name)逻辑上区别于结构元素值的类型(用普通文本印刷的类型名来表示，如 Name)。

7.4.3 字型索引映射资源描述 Font Index Map Resource Specification

FONT INDEX MAP RESOURCE SPECIFICATION 的值是复合结构元素，其直接下属包括：

——一个 MAP SIZE 结构元素；

——零个或一个 INDEX LIST 结构元素。

MAP SIZE 结构元素的值具有类型 Integer。

INDEX LIST 为复合结构元素，其直接下属包括：

——零个或多个 INDEX VALUE 结构元素。

INDEX VALUE 结构元素的个数为 MAP SIZE 结构元素的值。每个 INDEX VALUE 结构元素的值具有 Integer 类型。

对 FONT INDEX MAP RESOURCE SPECIFICATION 的处理包括建立一个如第 16 章中定义的字体索引映射向量，其元素的值与 INDEX LIST 结构元素中相应下属元素的值相同。若没有给定 INDEX LIST，则所产生的字型索引映射向量的所有元素全为零。

7.4.4 彩色空间资源描述 Color Space Resource Specification

COLOR SPACE RESOURCE SPECIFICATION 的值为复合结构元素，其直接下属包括：

——一个 COLOR SPACE NAME 结构元素；

——零个或一个 PRIMARY SET IDENTIFIER 结构元素；

——一个 COLOR SPACE SPECIFICATION 结构元素。

COLOR SPACE NAME 的值为公用对象标识符类型。公用对象标识符所定义的对象名以及这些值的含义在第 15 章中给出。

PRIMARY SET IDENTIFIER 的值是复合结构元素，它具有如下直接下属：

——一个或多个 PRIMARY COLOR IDENTIFIER 结构元素。

PRIMARY COLOR IDENTIFIER 结构类型为引用名(Reference Name)。本标准没有定义 PRIMARYCOLOR IDENTIFIER 公用对象标识符值。

COLOR SPACE SPECIFICATION 值的类型为 TOKENSEQUENCE，定义见第 9 章。

对彩色空间 RESOURCE SPECIFICATION 的处理包括：

——为处理 TOKENSEQUENCE 的内容建立一个解释上下文。

——若最直接的上级 RESOURCE DEFINITION 结构元素处在最高结构层，解释上下文包括：

——一个资源的空集；

——如 9.3.2 中定义的缺省上下文栈；

——一个只包含类型为 Identifier 元素的操作数栈，该元素的值决定于 COLOR SPACE NAME 的值，如下：

——若 COLOR SPACE NAME 的值是本标准给出定义的一种，则操作数栈中的元素为与该值相对应的对象名；

——若 COLOR SPACE NAME 的值不是由本标准所定义的一种，则操作数栈中的元素为一标识符，它是(value Convert To Identifier)的结果。此处 Value 为八位字节串，它表示在结构交换格式中 COLOR SPACE NAME 的值。

——在第 12 章中所定义的状态变量的初值。

——若最直接上级 RESOURCE DEFINITION 结构元素是 DOCOMENT 的下属，则解释上下文包括：

——最直接上级 BLOCK 的当前解释上下文中的资源集；

——最直接上级 BLOCK 的当前解释上下文的上下文栈；

——一个操作数栈，其中只包括一个类型为 identifier 的元素，元素的值决定于 COLOR SPACE NAME 的值，如下：

——若 COLOR SPACE NAME 的值是本标准给出定义的一种，则操作数栈中的元素为与该值相对应的对象名；

——若 COLOR SPACE NAME 的值不是由本标准所定义的一种，则操作数栈中的元素为一标识符，它是(Value Convert To Identifier)的结果。这里 Value 是八位字节串，它表示在所选用的结构交换格式中 COLOR SPACE NAME 的值。

——最直接上级 BLOCK 的当前解释上下文中状态变量的值。

——调用内容处理器，使用第一步中所建立的解释上下文，去处理 TOKENSEQUENCE 的内容。

在解释 TOKENSEQUENCE 内容的末尾，虚拟机的上下文栈与内容处理器解释上下文的上下文栈完全相同，并且由内容处理器返回的虚拟机最后状态的操作数栈包含一个表示彩色空间对象的向量，该向量的内容依赖于所用的彩色空间。

——确认前一步返回的向量所表示的彩色空间对象的类型与 COLOR SPACE NAME 值所标识的彩色空间对象的类型是否相同。

——定义一个由前一步返回的向量所表示的彩色空间对象。

7.4.5 网纹资源描述 Pattern Resource Specification

PATTERN RESOURCE SPECIFICATION 的值是复合结构元素，它有如下直接下属：

—— 一个或多个 TOKENSEQUENCE 结构元素。

TOKENSEQUENCE 的定义见第 9 章。

对 PATTERN RESOURCE SPECIFICATION 的处理包括：

——为处理下属元素 TOKENSEQUENCE 建立一个解释上下文。

——若最直接的上级 RESOURCE DEFINITION 结构元素处在最高结构层，则解释上下文包括：

——一个可用资源的空集；

——如 9.3.2 所定义的缺省上下文栈；

——如第 12 章中所定义的状态变量的初始值；

——一个空的操作数栈。

——若最直接的上级 RESOURCE DEFINITION 结构元素是 DOCUMENT 的下属，解释上下文即是最直接的上级 BLOCK 的当前解释上下文。

——对每个 TOKENSEQUENCE 结构元素依次调用内容处理器。

——处理第一个 TOKENSEQUENCE 的解释上下文就是在第一节中所建立的解释上下文；

——以下各个 TOKENSEQUENCE 的解释上下文就是处理器在处理完前一个 TOKENSEQUENCE 后其结果返回的解释上下文。

在处理完所有的直接下属 TOKENSEQUENCE 之末尾，满足：

——由内容处理器返回的虚拟机最终状态的上下文栈与传送给内容处理器的解释上下文的上下文栈相同；

——虚拟机的最终状态的操作数栈包含一个引用网纹对象词典的 Dictionary Reference。

——定义一个由词典表示的网纹对象，而该词典是由前一步所返回的 Dictionary Reference 所引用，如第 20 章所述。

7.4.6 图案资源描述 Form Resource Specification

FORM RESOURCE SPECIFICATION 的值是复合结构元素。其直接下属为：

——一个或多个 TOKENSEQUENCE 结构元素。

TOKENSEQUENCE 结构元素的定义见第 9 章。

对 FORM RESOURCE SPECIFICATION 的处理包括：

——为处理下属 TOKENSEQUENCE 建立一个解释上下文。

——若最直接上级 RESOURCE DEFINITION 结构元素处于最高结构层，则解释上下文包括：

——一个可用资源的空集；

——如 9.3.2 中所定义的缺省上下文栈；

——如第 12 章定义的状态变量的初值；

——一个操作数栈的空集。

——若最直接上级 RESOURCE DEFINITION 结构元素是一个 DOCUMENT 的下属，解释上下文即是最直接上级 BLOCK 的当前解释上下文。

——对每个 TOKENSEQUENCE 结构元素依次调用内容处理器。

——处理第一个 TOKENSEQUENCE 的上下文就是第一步中建立的解释上下文；

——处理以下各 TOKENSEQUENCE 的解释上下文为其前一个 TOKENSEQUENCE 处理完毕后返回的解释上下文。

处理完所有的直接下属 TOKENSEQUENCE 的末尾，则满足：

——由内容处理器返回的虚拟机的最终状态的上下文栈与传递给内容处理器的解释上下文的上下文栈相同；

——内容处理器返回给虚拟机的最终状态的操作数栈包含一个引用 FormObject Dictionary

Reference 的 DictionaryReference。

——定义一个 Form object，它由前一步返回的 DictionaryReference 所引用的词典所表示，定义见第 23 章。

8 文件生成指令

在表示 SPDL 文件的过程中文件生成指令(DPI)完成两个功能。第一个功能就是提供了对那些超出文件内容记法范围的文件表示过程的控制。例如单面和双面(两面)打印模式的选择、媒体的指定和选择。第二个功能提供了对文件表示过程的控制，而这种表示过程随着表示实例的不同而变化。例如，要打印的份数，还有媒体的指定和选择。这两个功能并不是互斥的。

本章描述了 SPDL 中文件生成指令的使用，定义了可用于 SPDL 表示过程的文件生成指令集的语义，在 8.1 中定义了文件生成指令的模型，8.2 中给出了一些文件生成指令描述的细节和可用于每个页面的文件生成指令描述的方法。在 8.3～8.6 中给出了本标准中定义的那些文本生成指令的语义和语法说明。

8.1 文件生成指令模型

每条文件生成指令由一个名字和一个值组成，这个名字是公用对象标识符值，值的类型依赖于名字所标识的具体文件生成指令。

每条文件生成指令作用于一个参数，该参数影响 SPDL 表示过程，文件生成指令的名字指出了该指令所作用的参数，文件生成指令的值指出了作用于这个参数的途径。

8.1.1 文件生成指令的源定义

为了控制那些超出了内容记法范围的文件表示，本标准提供了在文件结构中指定文件生成指令的方法，作为文件一部分的文件生成指令通过结构元素 DPI DECLARATION 的方式来指定。如果一个 BLOCK 的 PROLOGUE 中的 DPI DECLARATION 指定了一条文件生成指令，则称这条指令“出现在 BLOCK 中”。

为了控制那些随着表示实例的不同而变化的文件表示，部分地而不必去改变每一个表示实例的 SPDL 文件，用户必须在给出文件本身的同时提供适用于文件表示的特定实例的文件生成指令，这种文件生成指令称为 Supplementary DPI(辅助 DPI)。为表示过程提供有效的辅助 DPI 的方法不在本标准讨论之列，辅助 DPI 的适用范围是整个 DOCUMENT。

在一个 SPDL 文件的任何一个特定的表示实例的过程中，上述这些源定义的一种或两种可能只指定了一个文件生成指令的空集。不能接受辅助 DPI 的 SPDL 表示过程对于文件的所有表示实例都只有一个辅助 DPI 的空集。

8.1.2 可扩充性

除了本标准中定义的那些文件生成指令以外，SPDL 语法还允许在 SPDL 文件或辅助 DPI 中包含附加的文件生成指令，这种文件生成指令称为 additional DPI(附加 DPI)，8.6 中说明了在 SPDL 文件中包含的附加 DPI 的语法。

本章说明了那些由本标准中定义的公用对象标识符值来标识的，文件生成指令的语义，附加 DPI 的语义由标识它的公用对象标识符值来指定。

附加 DPI 的执行效果可能是以下两种之一：

——由该附加 DPI 语义所定义的效果；

——这条指令被忽略。

当附加 DPI 的定义方法以及执行附加 DPI 时 SPDL 文件表示所产生的影响超出了本标准的范围时，就尤其需要包含这样的附加 DPI。

8.1.3 文件生成指令的可协调性

既然文件生成指令可以在一个 SPDL 文件的任意 BLOCK 中出现，那么上级 BLOCK 中的文件生成

指令就有可能与下属 BLOCK 中的文件生成指令发生抵触，在这种情况下结构处理器必须协调(reconcile)发生冲突的文件生成指令。

文件生成指令的作用域就是定义它的 DPI DECLARATION 的作用域，如果有两个或者更多的文件生成指令，其中一个出现在其他指令的作用域中，那么说他们具有覆盖作用域。协调两个有覆盖作用生成指令，并且作为每条文件生成指令语义的一部分作了定义。经过协调具有覆盖作用域的文件生成指令再加上指定的默认值后，就产生了一个文件的文件生成指令集，称之为 Document DPI(文件 DPI)。

辅助 DPI 的作用域是整个 DOCUMENT，因此对于具有非空辅助 DPI 集的任何一个表示实例来说，在辅助 DPI 和文件 DPI 之间总有覆盖作用域，在这种情况下，结构处理器必须协调辅助 DPI 和文件 DPI，结构处理器协调辅助 DPI 和文件 DPI 的方法依赖于其中的文件生成指令，并且作为每条文件生成指令语义的一部分来定义。

8.1.4 文件生成指令的分类

文件生成指令分为五类：

——预处理指令，它作用于描绘文件页面图像之前的文件生成处理过程；

——媒体和颜料说明(Medium and Colorant Declarations)：它指定了可以用于文件表示的媒体或颜料；

——表示指令。它作用于描绘文件页面图像的处理过程，并决定它们在媒体上的位置。

——结束指令。它作用于描绘文件页面图像之后的文件生成处理过程；

——管理指令。它们作用于文件生成处理的管理过程。

应用文件生成指令有一个隐含的序列：

——预处理指令作用于在表示媒体上描述页面图像前的文件表示的处理过程。预处理指令聚集的结果构成一个新的 SPDL 文件，这个新文件使用原先的表示指令、结束指令和管理指令来表示。

——表示指令作用于预处理指令的结果。它们可以独立地在媒体上描绘文件页面图像。表示指令可能会用到媒体和颜料说明。

——结束指令作用于媒体上描绘文件页面图像的处理结果。结束指令的使用顺序是有意义的。

——管理指令作用于文件生成过程的所有部分，并且不依赖于使用预处理指令、表示指令、结束指令的顺序。

8.1.5 回撤策略 Fallback Strategies

本标准中说明的有些文件生成指令提供的某些操作，并不是所有的表示过程都需要提供这些操作，因此在每个文件生成指令的语义中除了要说明在具有这种能力的系统中所要做的操作以外，还要求说明在那些没有这种能力的系统中所作出的回撤行为。

在一些普通的系统中，如软拷贝设备，某些文件生成指令，例如结束和拷贝指令，一般是不适用的。如果特定系统中文件生成指令不适用于文件表示，回撤策略则依赖于具体系统。

8.1.6 文件生成指令结构小结

下面归纳了用于文件生成指令说明的结构元素：

DPI DECLARATION
  DPI NAME
  DPI VALUE
   PROPERTY LIST
    PROPERTY
     PROPERTY NAME
     PROPERTY VALUE
PROPERTY LIST
  PROPERTY

PROPETY NAME

PROPETY VALUE

8.2 文件生成指令说明

8.2.1 概述

定义每个文件生成指令的各条说明：

——文件生成指令名；

——文件生成指令值的类型；

——该文件生成指令的值对影响文件表示的一个或多个参数所产生的效果；

——受影响的参数值对 SPDL 文件的表示所产生的效果；

——受影响的参数的默认值；

——对于不能达到指定效果的表示过程所作出的回撤行为；

——该文件生成指令在一个文件的不同 BLOCK 中多次出现所产生的相互作用；

——该文件生成指令在文件 DPI 和辅助 DPI 中出现所产生的相互作用。

8.2.2 DPI 说明 DPI DECLARATION

一个 DPI DECLARATION 说明了一条文件生成指令，它由文件生成指令名和一个值组成，值的类型决定于名，DPI 的语法不仅支持由本标准定义的文件生成指令，也支持专用文本生成指令（附加 DPI）。

DPI DECLARATION 是一个应该包含下列直接下属的复合结构元素：

——一个 DPI NAME 结构元素；

——一个 DPI VALUE 结构元素。

这些结构元素应当按指定次序出现。

DPI NAME 结构元素的值的类型应当是公用对象标识符。DPI VALUE 结构元素值的类型取决于由 DPI NAME 的值所标识的文件生成指令。

除非另外说明，任何一个 BLOCK 中，不能有一个以上的直接从属于同一 PROLOGUE 的 DPI DECLATATION 具有相同的 DPI NAME 结构元素的值。

注

1 DPI NAME 的值可以是本标准中定义的任何一个文件生成指令名，也可以是任何其他的公用对象标识符值。

2 指定了含义的 DPI VALUE 的值集可能并不包含所属类型的所有可能值；这样的话只要 DPI VALUE 的值具有适当的类型，它就可以不在指定的值集中。

8.2.3 属性表 PROPERTY LIST

有一些文件生成指令的值包括属性表，它们由 PROPERTY LIST 和 PROPERTY 结构来代表。

PROPERTY LIST 是一个应该包含下面直接下属的复合结构元素：

——零个或多个 PROPERTY 结构元素；

每个直接下属 PROPERTY 结构元素标识了一个属性并为它指定了一个值，这些属性称之为“由 PROPERTY LIST 结构元素所指定”。

PROPERTY 是一个应当包含以下直接下属的复合结构元素：

——一个 PROPERTY NAME 结构元素；

——一个 PROPERTY VALUE 结构元素。

这些结构元素应当按指定次序出现。

PROPERTY NAME 结构元素值的类型是 ReferenceName，PROPERTY VALUE 结构元素值的类型取决于由 PROPERTY NAME 的值所标识的属性。

PROPERTY NAME 结构元素的值以“name of property（属性的名字）”或“property name（属性名）”的形式引用；PROPERTY VALUE 结构元素的值以“value of property（属性的值）”或“property value（属

性值)"的形式引用。

本标准定义的类型为 PROPERTY LIST 的文件生成指令有一个相联系的 PROPERTY 结构元素集合,这个集合由属性名定义和属性值定义组成,一般说来类型为 PROPERTY LIST 的文件生成指令能够定义附加 DPI 的属性,附加 DPI 的属性是与本标准的定义不同的特殊属性。附加 DPI 属性产生的效果可能是下面两种之一:

——附加 DPI 被忽略。

除非另外说明,在任何一个 PROPERTY LIST 中不允许有一个以上的 PROPERTY 具有相同的 PROPERTY NAME 结构元素值。

8.2.4 名字和公用对象标识符值

所有文件生成指令的名字和很多文件生成指令的值的类型都是公用对象标识符,另外,很多文件生成指令的值是直接下属为 PROPERTY 的复合结构元素,这些属性用公用变量标识符值来命名,这些属性的值也可以是这一类型。

对于任何文件生成指令名或属性名,象下面一种形式的语句:

"X 的名字是 object-name"

意味着将用对应于对象名 object name 的公用对象标识符的值来命名这个文件生成指令或属性。

如果文件生成指令的值或属性值的类型是公用对象标识符,那么下面形式的语句

"公用对象标识符所定义的对象名以及这些值的含义是……"

意味着本标准定义了对应于每个指定对象名的公用对象标识符值,并且这些值的含义由语句的余下部分赋给,本标准中定义的公用对象标识符值在标准附录 B 中说明。

8.2.5 Run Vectors 运行向量

有些文件生成指令在文件显现时对每个页面都发生影响,例如页面选择和媒体选择,这种文件生成指令的值由一连串的值组成,每个值作用于一个页面,而这些页面是由指令所作用的 BLOCK 生成的。

这些值的序列由结构元素 RUN VECTOR 来表示,RUN VECTOR 本身可以产生一个无限长的序列,当用 RUN VECTOR 来表示文件生成指令一个值的序列时,序列中的值由 RUN VECTOR 决定。

RUN VECTOR 是一个应当包含下列直接下属的复合结构元素:

——零个或多个 RUN 结构元素。

RUN 是一个应该包含下列直接下属的复合结构元素:

——一个 RUN LENGTH 结构元素;

——一个 RUN VALUE 结构元素。

这些结构元素应当按指定次序出现。

RUN LENGTH 值的类型是 Integer,RUN VALUE 值的类型是 Name 或 Integer。一个 RUN VECTOR 结构元素,如果它是特定文件生成指令的值,那么从属于它的 RUN VALUE 结构元素的值可能进一步受到限制。

由 RUN VECTOR 表示的值的序列可以通过下列方法生成:

1) 某个 RUN VECTOR 的值由 RUN 的序列 $run_1,\cdots,run_k$ 组成;设第 i 个 RUN run_i 的下属值为 n_i 和 v_i。

2) 每个 n_i 是一个整数,它指出了值 v_i 的出现次数,遍历 run vector 生成 n_1 个 v_1,随后是 n_2 个 v_2,直到 run vector 的所有元素都被耗尽。

3) 第 2 步中描述的值的序列按照要求进行复制以产生任意长的值序列。

4) 从步骤 3 描述的序列中析取值以便为 BLOCK 中的每个页面提供一个值。

这样遍历一次 RUN VECTOR 可能会产生比要求的数量多或少的值,但是所定义的值的序列是很明确的。

8.3 预处理指令

在本标准中没有定义预处理指令。

8.4 媒体和颜料说明

媒体和颜料文件生成指令说明了可能用于SPDL文件表示的物理资源，每个媒体文件生成指令指定了一种媒体。每个颜料文件生成指令指定了一个已命名颜料的集合。

媒体和颜料说明只指出了那些潜在地可能在SPDL文件表示期间使用的物理资源，真正使用这些物理资源必须由8.5节中所定义的表示指令来指定。

8.4.1 媒体文件生成指令

8.4.1.1 媒体 DPI DECLARATION

媒体文件生成指令的名字应该是DPI/Medium。媒体文件生成指令的值应该是一个包含下列直接下属的复合结构元素：

——一个MEDIUM IDENTIFIER结构元素；

——一个MEDIUM DESCRIPTION结构元素。

MEDIUM IDENTIFIER的值的类型是Name，MEDIUM DESCRIPTION是一个PRORERTY LIST(属性表)。

MEDIUM DESCRIPTION指定了一个标识媒体的属性集，各单个属性称为媒体属性，8.4.1.1.1节中说明了使用媒体属性来表征一个物理媒体的途径。

执行媒体文件生成指令的效果就是在参数medium-list中增加或替换一个元素，这个参数说明了文件表示中可能用到的媒体的集合，medium-list参数是一个有零个或多个元素的表，每个元素将一个特定物理媒体与一个名字联系起来，medium-list参数的默认值是一张空表，表示对于所有的页面使用依赖于系统的默认设备。

媒体文件生成指令在任何PAGE或PAGESET或辅助DPI中都是有意义的，任何出现在其他地方的媒体文件生成指令都被忽略，在DPI DECLARATION的作用域中媒体文件生成指令的执行结果就是在medium-list中增加一项，它将一个物理媒体与一个名字联系起来，这个名字就是MEDIUM IDENTIFIER的值。

如果两条具有相同的MEDIUM IDENTIFIER下属值的媒体文件生成指令在两个BLOCK中出现，那么在DPI DECLATATION的作用域中起主导作用的是下属块中出现的媒体文件生成指令；如果一条辅助媒体DPI与一条或多条文件媒体DPI具有相同的MEDIUM IDENTIFIER下属值，那么以辅助DPI为主导。

8.4.1.1.1 媒体说明

在8.4.1.2中定义了本标准说明的媒体属性的集合。在定义每一个媒体的属性时，各条中指出了：

——媒体属性的一般语义；

——媒体属性名；

——媒体属性值的类型；

——用来表示属性含义的值的集合；

——每个指定值的含义。

对于有些媒体属性，用来表示属性含义的值的集合并不包括所属类型的所有可能的值，MEDIUM DESCRIPTION可以将媒体指定为一个含义没有在8.4.1.2中定义的值，只要这个值具有恰当的类型就可以了，这种媒体属性的作用效果依赖于具体的系统。

MEDIUM DESCRIPTION可以将值指定为8.4.1.2中定义的任何一个媒体属性，另外，MEDIUM DESCRIPTION可以包括一个或多个PROPERTY结构元素；其中的PROPERTY NAME值是一个不同于本标准中定义的公用对象标识符值的引用名(Reference Name)，这个PROPERTY的作用效果依赖于具体的系统。

根据MEDIUM DESCRIPTION中指定的一组媒体属性选择一个媒体，并将它赋给MEDIUM I-

DENTIFER。系统选择媒体的方法依赖于具体实现，实际上系统可能为所有的 MEDIUM DESCRIPTION 选择一个缺省媒体，本节的其余部分说明了一个能够在可用媒体表中进行选择的系统是如何将 MEDIUM SPECIFICATION 指定为一个特定媒体的。

媒体属性 Medium Name 是标志媒体的主要方法，Medium Name 可以标志单个媒体或一类介质，如果 MEDIUM NAME 标识了单个媒体，那么媒体属性 Medium Name 从总体上指定了这个媒体；MEDIUM DESCRIPTION 中的任何附加属性只用于媒体替换，如果一个媒体代替了指定的媒体，系统应当发出结构警告。

如果 Medium Name 标识了一类媒体，那么 MEDIUM DESCRIPTION 中的附加属性就用来在属于该类的可用媒体中进行选择，不管 MEDIUM DESCRIPTION 指定的附加属性是什么，选中的介质都应该是 Medium Name 所标志的一类媒体中的一个。

如果 MEDIUM DESCRIPTION 中不包括任何媒体属性 Medium Name，那么 MEDIUM DESCRIPTION 中定义了的属性以及没有定义的属性的默认值一起决定了中选的媒体，将这样的 MEDIUM DESCRIPTION 和某个媒体联系起来的具体方法依赖于系统实现。

注：实际上一个系统可能将所有的 MEDIUM DESCRIPTION 指定为一个缺省媒体或者一个依赖于系统当前状态的媒体（例如，在特定输入支架上的任何媒体）。

8.4.1.2 媒体属性

下面的各条中说明本标准中定义的媒体属性，并解释了指定对应媒体属性的 PROPERTY 结构元素的语法。

8.4.1.2.1 Medium Name（媒体名）

媒体属性 Medium Name 指定了单个媒体或者一类媒体的标识。

媒体属性 Medium Name 的名字是 DPI/Medium/Name，媒体属性 Medium Name 的值的类型是 ReferenceName（引用名）。

定义为公用对象标识符值的对象名以及这些值的含义应该：

对象名	含义
DPI/Medium/Name/defult	依赖于系统的缺省媒体

媒体属性 Medium Name 没有缺省值。

8.4.1.2.2 Medium Size（媒体尺寸）

媒体属性 Medium Size 的名应为 DPI/Medium/Size。媒体属性 Medium Size 的值应该是包含下列直接下属的一个复合结构元素：

——一个 MEDIUM X SIZE 结构元素；

——一个 MEDIUM Y SIZE 结构元素。

这些下属应当以指定的次序出现。

MEDIUM X SIZE 和 MEDIUM Y SIZE 的值的类型应该是 Integer（整型），如果一个媒体属性 Medium Size，它的 MEDIUM X SIZE 的值是 x，它的 MEDIUM Y SIZE 的值是 y，那么就意味着介质的尺寸是 x mm × y mm，并且按下列方式将引用坐标系与具体的媒体联系起来：

——长为 x mm 的边对应于引用坐标系的正 x 轴；

——长为 y mm 的边对应于引用坐标系的正 y 轴。

媒体属性 Medium Size 的默认值依赖于具体系统。

注：一般地说，默认值为[216,279]（8.5in×11in）。或者[210,297]（ISO A4）。

8.4.1.2.3 Medium Color（媒体色彩）

媒体属性 Medium Color 指出了媒体的色彩。

媒体属性 Medium Color 的名应为 DPI/MEDIUM/COLOR，媒体属性 Medium Color 的值的类型是 Reference Name。定义为公用对象标识符值的对象名以及这些值的含义应为：

对 象 名	含义
DPI/Medium/Color/white	白色
DPI/Medium/Color/pink	粉红
DPI/Medium/Color/yellow	黄色
DPI/Medium/Color/bnft	浅黄
DPI/Medium/Color/goldenrod	菊黄
DPI/Medium/Color/blue	蓝色
DPI/Medium/Color/green	绿色
DPI/Medium/Color/nocolor	透明设备 untinted

媒体属性 Medium Color 的默认值应为 DPI/Medium/Color/white

8.4.1.2.4 Medium Weight(媒体重量)

媒体属性 Medium Weight 指出了媒体的重量。

媒体属性 Medium Weight 的名字应为 DPI/Medium/Weight,媒体属性 Medium Weight 的值的类型应为整型。每平方米的克数(g/m^2),舍入到最近的常规值,作为属性值来表示媒体的重量。

Medium Weight 的默认值依赖于具体系统。

注:一般地说,这个默认值是 $75g/m^2$,等价于所谓的"substance(纸张重量单位)20"or "20 lb"的纸张。

8.4.1.2.5 Medium Opacity(媒体的不透明性)

媒体属性 Medium Opacity 指出了媒体的不透明性。

媒体属性 Medium Opacity 的名字应为 DPI/Medium/Opacity。媒体属性 Medium Opacity 的值的类型是 Integer(整型)。定义媒体属性 Medium Opacity 含义的值以及其含义应为:

值	含义
0	透明的;即媒体允许光线透过
1	不透明的;即媒体不允许光线透过

媒体属性 Medium Opacity 的默认值是 1。

8.4.1.2.6 Medium Pre-Finish Type(媒体预处理类型)

媒体属性 Medium Pre-Finish Type 指出了媒体的附加属性,例如制表符截断粘性回撤(tab cuts adhesive backing),连续格式纸(continous form paper)或者特定的媒体类型,例如信封。

媒体属性 Medium Pre-Finish Type 的名字应为 DPI/Medium/Prefinish。媒体属性 Medium Per-Finish Type 的值的类型是 Reference Name。定义为公用对象标识符值的对象名以及这些值的含义应为:

对象名	含义
DPI/Medium/Prefinish/Plain	普通纸
DPI/Medium/Prefinish/PreCutTab	有预截除制表符的纸
DPI/Medium/Prefinish/Continuous	连续格式纸(例如计算机打印纸)
DPI/Medium/Prefinish/oidhes	粘性标签
DPI/Medium/Prefinish/EnevelopPlain	没有窗口的信封
DPI/Medium/Prefinish/envelopWindow	带有一个或多个透明窗的信封

媒体属性 Medium Prefinish Type 的默认值应为 DPI/Medium/Prefinish/Plain。

在 MEDIUM DESCRIPTION 中媒体属性 Medium Pre-Finish Type 可以出现多次,如果媒体属性 Medium Pre-Finish Type 多次出现,那么在选择媒体时对这些媒体属性总体上应完全相等地进行解释。虽然可以对媒体属性 Medium Pre-Finish Type 任意组合,但有些组合是没有什么意义的,根据这种属性、选择的结果依赖于具体系统。

8.4.1.2.7 Inline Hole Count(行孔数)

媒体属性 Inline Hole Count 指出了预先打好的孔的数目，对于用圆环串起来(ring binding)的纸张来说这些孔排成一行。

媒体属性 Inline Hole Count 的名字应为 DPI/Medium/HoleCount，媒体属性 Inline Hole Count 的值的类型是 Integer。

媒体属性 Inline Hole Count 的默认值应为 0。

8.4.1.2.8 Ordered Count And Position(次序和位置)

媒体属性 Ordered Count And Position 指出了媒体是具有 m 个媒体序列中的第 n 个，该序列中的媒体都具有媒体属性 Medium Pre-Finish 所指定的类型。

媒体属性 Ordered Count and Position 的名字应为 DPI/Medium/OrderedCount，媒体属性 Ordered Count and Position 的值应该是一个包含下列直接下属的复合结构元素：

——一个 SEQUENCE NUMBER 结构元素；

——一个 SEQUENCE LENGTH 结构元素。

这些下属应当按指定顺序出现。

SEQUENCE NUMBER 和 SEQUENCE LENGTH 的值的类型都是 Integer。如果媒体属性 Ordered Count and Position 的 SEQUENCE NUMBER 值为 n，SEQUENCE LENGTH 的值为 m，那么意味着介质是具有媒体属性 Medium Pre-Finish 所指定类型的 m 个媒体序列中的第 n 个媒体。

媒体属性 Ordered Count and Position 没有默认值。

8.4.1.2.9 Finish Edge(结束边)

媒体属性 Finish Edge 指定了媒体具有良好的边特性，比如说制表符扩展特性。

媒体属性 Finish Edge 的名字应为 DPI/Medium/FinishEdge。媒体属性 Finish Edge 的值的类型是 Integer(整型)，定义媒体属性含义的值以及这些值的名义应该为：

值	边
0	与 SPDL 引用坐标系(RCS)的 x 轴重合的那条媒体边(底边)
1	平行于 RCS 的 y 轴，并从 y 轴移动了 x-size 这么长距离的媒体边(右边)
2	平行于 RCS 的 x 轴，并从 y 轴移动了 y-size 这么长距离的媒体边(顶边)
3	与 RCS 的 y 轴重合的那条媒体边(左边)

媒体属性 Finish Edge 的默认值为 1(右边)。

8.4.1.2.10 Medium Labels(媒体标签)

媒体属性 Medium Labels 指出了媒体预留标签的布局格式，即通常所指的标签位置(label Stock)。只有当媒体属性 Medium Pre-Finish Type 的值为 DPI/Medium/PreFinish/adhelsiveLables 时媒体属性 Medium Labels 的值才有意义。

媒体属性 Medium Labels 的名字应为 DPI/Medium/Labels，媒体属性 Medium Labels 的值是一个包含下列直接下属的复合结构元素：

——一个 PER COLUMN 结构元素；

——一个 PER ROW 结构元素。

这些下属应当按指定次序出现。

PER COLUMN 和 PER ROW 的值类型都是 Integer，如果一个媒体属性 Medium Labels 的 PER COLUMN 值为 n，PER ROW 值为 m，那么每列有 n 个预留的标签位置，每行有 m 个预留的标签位置。

媒体属性 Medium Labels 的默认值是 PER COLUMN＝0，PER ROW＝0，即没有特别的标签列或行。

8.4.1.2.11 Medium Message(媒体消息)

媒体属性 Medium Message 指出了一个可打印字符串，它描述了该媒体建议的回撤动作以及其他

信息，这些信息传送给文件打印之前的操作符和紧挨着的操作符。

媒体属性 Medium Message 的名字应当是 DPI/Medium/Message。媒体属性 Medium Message 的值的类型就是 6.5 中定义的 Printable String(可打印字符串)。

媒体属性 Medium Message 没有默认值。

8.4.2 颜料文件生成指令

颜料文件生成指令的名字应为 DPI/Colorants。颜料文件生成指令的值应当是一个包含下列直接下属的复合结构元素：

——一个 COLORANT SET IDENTIFIER 结构元素；

——一个 COLORANT SET 复合结构元素。

COLORANT SET IDENTIFIER 的值是一个名字，COLORANT SET IDENTIFIER 标识了由 COLORANT SET 值说明的颜料集合，元素 COLORTANT SET 应当是一个包含下列直接下属的复合结构元素：

——零个或多个 COLORANT IDENTIFIER 元素。

每个 COLORANT IDENTIFIER 值的类型是 Reference Name。

每个 COLORANT IDENTIFIER 元素标识着一套在 SPDL 文件表示期间使用的物理颜料，用来描绘文件的物理颜料与彩色空间资源中定义(Color Space Resource Specifications)的基色不同，它们之间甚至也没有直接的关系，将彩色说明映射到设备输出的算法是依赖于具体设备的。

注：如果有了可用的合适的物理颜料，那么根据整个 SPDL 彩色空间模型提供的机制进行 SPDL 文件的描绘是十分容易的。

颜料文件生成指令的执行效果就是在参数 Colorants-list 中增加或置换一个元素，这个参数指定了一个可用于文件表示的颜料值，参数 colorants-list 是一张有零个或多个元素的表，每个元素将一个名字与一个特定的颜料集联系起来。参数 colorants-list 的默认值是一张空表，从而使得对于所有的页面使用由系统决定的颜料。

在任何 PAGE 或 PAGE SET 或辅助 DPI 中的颜料文件生成指令都是有意义的，出现在其他地方的任何颜料文件生成指令都被忽略，在 DPI DECLARTION 的作用域中一条颜料文件生成指令的执行结果就是在 Colorants-list 中增加一个元素，它将一个颜料集与 COLORANT SET IDENTIFIER 的名字值联系起来。

如果具有相同的 COLORANT SET IDENTIFIER 值的颜料文件生成指令出现在两个 BLOCK 中，那么在这条 DPI DECLARTION 的作用域中下属 BLOCK 中的颜料文件生成指令占主导地位，如果一条辅助颜料文件生成指令与一条或多条颜料文件生成指令有着相同下属 COLORANT SET IDENTIFIER 的值，那么以辅助 DPI 为主导。

参数 Colorant-list 的初始化值应当是一张空表，在 SPDL 文件处理期间，如果由颜料文本生成指令指定的颜料集在已经存在的 Colorants-list 名字/集合对的表中是唯一的，那么就往 COLORANTS-LIST 中增加一个新的名字/集合对；如果颜料文件生成指令所指定的颜料集在已经存在的 colorants-list 名字/集合对的表中是重复的，那么就将已经存在于 colorants-list 值中的名字/集合对置换为新的名字/集合对。要知道新的颜料定义是唯一还是重复的，应当用当前颜料文件生成指令中的 COLORANTS SET IDENTIFIER 与 colorants-list 值中的每个名字/集合对的 COLORANTS SET IDENTIFIER 进行比较，以此来判别名字是否等价。如果这个依赖于系统实现的判别过程的结果表明这是两个完全一致的值，包括(但不仅限于)标点符号和大小写，那么说这是等价的 COLORANTS SET IDENTIFIER 的值。参数 colorants-list 的默认值应该是一张空表，即对所有的页面使用依赖于系统的默认的颜料。

可以有多条颜料文件生成指令直接从属于一个 DOCUMENT 的同一个 PROLOGUE，或者在辅助 DPI 中说明。

颜料文件生成指令只有在文件或辅助 DPI 中才有意义。出现在任何其他地方的指令都被忽略，如

果一条颜料文件生成指令以文件 DPI 和辅助 DPI 的形式出现，那么将用 8.4.2 描述的方法把这些说明归并到参数 Colorants-list 的值中去。

8.5 显现指令

8.5.1 拷贝数 Copies

拷贝文件生成指令的名字是 DPI/Copies。拷贝文件生成指令的值的类型是 Integer。拷贝文件生成指令的执行结果就是把参数 number-of-copies 的值置为拷贝文件生成指令的值，这个参数指出待打印文件的拷贝份数。

对用户来说，文件的拷贝份数的定义方式与输出位置 DPI(Output Position DPI)、叠页 DPI(Stacking DPI)是一致的，如果 number-of-copies=0，那么一份都不打印，并且显现过程将忽略 BLOCK 的所有其他下属。

参数 number-of-copies 的默认值是 1。有些系统不能以整理方式(collated form)输出文本的多份拷贝，如果要这样做系统会产生一个结构警告以提示错误。如果参数 abort-policy 的值允许发生结构警告后继续处理过程，那么回撤策略有两种：或者以非整理方式表示页面(第一页的所有拷贝后面跟着第二页的所有拷贝，等等)；或者以 BLOCK 整理方式(开头 n 页的所有拷贝(整理方式)后面跟着随后 n 页的所有拷贝，等等)。选择非整理方式还是 BLOCK 整理方式以及 BLOCK 的大小依赖于系统实现。

拷贝文件生成指令只有在 DOCUMENT 中或辅助 DPI 中才是有意义的，任何出现在 BLOCK 中而不是出现在 DOCUMENT 中的拷贝文件生成指令都被忽略，如果拷贝文件生成指令以辅助 DPI 形式出现，那么辅助 DPI 将取代文件 DPI。

8.5.2 页面选择 page Select

页面选择文件生成指令指出了文件的要显现那些页面。

页面选择文件生成指令的名字应为 DPI/Page Select。页面选择文件生成指令的值的类型是 RUN VECTOR。RUN VECTOR 的值应当是一个值为 0 或 1 的整数。

页面选择文件生成指令的执行结果就是将参数 page-select 的值置为页面选择文件生成指令的值，参数 page-select 的值是一个由非零即 1 的整数组成的序列。如果一个文件中有 n 个页面，那么对于 $1 \leqslant i \leqslant n$ 顺序定序的第 i 个页面。

——如果 page-select(i)=1 则显现之；

——如果 page-select(i)=0 那么显现过程将忽略之。

page-select 的默认值应该是对于所有的 i 值，page-select(i)=1，这样将显现文件的所有页面，页面选择没有回撤行为。

页面选择文件生成指令可以出现在文件的多重嵌套 BLOCK 中，由下面规则决定了适用于所有页面的 page-select 的有效值：

——上级 BLOCK 中指定的 page-select 值仅仅适用于由下属 BLOCK 的 page-selected 值所决定的已表示的页面序列。

这意味着嵌套最深的 page-select 值选择的页面所构成的页面集合，就是可供次高级 BLOCK 中的 page-select 值从中选择的整个页面集合，上级 BLOCK 中指定的 page-select 值完全依赖于下属 BLOCK 中指定的 page-select 值所选择的页面的结果。

如果页面选择文件生成指令以辅助 DPI 的形式出现，那么只有先用文件 DPI 中的 page-select 值选择表示的页面集合后，才对这个集合使用辅助 DPI 指定的 page-select 值进行进一步选择。

8.5.3 媒体选择显现指令 Medium Selection Presentation Instructions

媒体选择显现指令指出了如何将物理媒体与文件要表示的页面联系起来，“媒体选择”这个词在本条中表示一个与物理媒体有关的处理过程，这个媒体早已由一条媒体文件生成指令指定了。

用于文件显现的媒体由名字来标识，媒体的选择依赖于两个参数：

——参数 medium-select，它的值或为 NULL，或为一个名字序列；

——参数 current-medium，它的值或为 NULL，或为一个单个名字。

参数 medium-select 的值由媒体选择文件生成指令设置，参数 current-medium 的值由当前媒体文件生成指令来设置。

如果一个文件有 n 个页面，那么对于 $1 \leqslant i \leqslant n$，根据参数 medium-select 和 current-medium 的值将按如下方法决定用于显现第 i 个页面的媒体名：

——如果参数 medium-select 值为 NULL，那么根据参数 current-medium 的值来选择媒体；

——如果参数 current-medium 为 NULL，将选择由系统决定的缺省媒体；

——如果参数 current-medium 不为 NULL，那么根据参数 medium-list 所建立的对应关系选择与参数 current-medium 的值对应的媒体；

——如果参数 medium-select 不为 NULL，那么根据参数 medium-list 所建立的对应关系选择与 medium-select(i)的值相对应的媒体。

如果参数 medium-select 和 current-medium 都不为 NULL，则根据参数 medium-list 所建立的对应关系选择与参数 current-medium 或 medium-select(i)的名字值相对应的物理媒体，参数 medium-list 由 8.4.1 中定义的媒体文件生成指令来设置，如果 medium-list 中没有一个元素将一个物理媒体与参数 current-medium 或 medium-list(i)的名字值对应起来，那么将出现结构警告并使用由系统决定的媒体。

不管参数 plex(在 8.5.5 中定义)的当前值是什么，选择一个新媒体的操作将在指定介质的正面表示指定的页面，尤其是如果参数 plex 的值是 DPI/plex/duplex 或 DPI/plex/tumble Duplex，并且将要在当前媒体的反面表示一个页面，选择一个由参数 medium-select 的值指定了新的媒体，那么这种选择新媒体的后果将导致该页面在新的具体媒体的正面表示。

注：嵌套最深的媒体指定将取代文件 DPI，辅助 DPI 也会取代文件 DPI。

8.5.3.1 媒体选择 Medium Select

媒体选择文件生成指令对于文件的每个页面，指出了用于显现的媒体。

媒体选择文件生成指令的名字是 DPI/Medium Select，媒体选择文件生成指令的值的类型应是 RUN VECTOR，RUN VECTOR 所代表的值的类型应该是 Name(名字)。

执行媒体选择文件生成指令的效果就是将参数 medium-select 的值置成媒体选择文件生成指令的值，参数 medium-select 的值应为 NULL 或一个名字序列，参数 medium-select 的默认值是 NULL。

既然将一个媒体赋给 Medium Description 的方法是依赖于系统实现的，正如 8.4.1.1 中所定义的，那么不需要考虑回撤。

注：系统实际上可以把每个 Medium Description 以及用 NULL 值来标识的媒体都赋为缺省媒体。

媒体选择文件生成指令只有在辅助 DPI 中才有意义，任何出现在文件 DPI 中的媒体选择文本生成指令都被忽略。

8.5.3.2 当前媒体 current Medium

当前媒体文件生成指令指出了一个用以表示一个 BLOCK 中页面的媒体，这个媒体由一个名字来标识，正如 8.4.1 中所指出的这个名字已经对应于一个媒体描述。当前媒体文件生成指令的名字应为 DPI/Current Medium，当前媒体文件生成指令的值的类型是 Name(名字)。

在 DPI DECLARATION 的作用域中，执行当前媒体文件生成指令的结果就是将参数 current-medium 的值赋成当前媒体文件生成指令的值，参数 current-medium 的值的类型可以是 Name(名字)或 NULL。参数 current-medium 的默认值是 NULL，它表示使用系统定义的媒体。

既然在 8.4.1.1 中定义的将一个媒体与一个媒体说明联系起来的方法是依赖于系统的，那么不需要考虑回撤。

当前媒体文件生成指令在任何 PAGE，PAGE SET 或 DOCUMENT 中都是有意义的。任何出现在 PICTURE 的而不是 PAGE 的 PROLOGUE 中的当前媒体文件生成指令将被忽略。如果在一个文件的多个 BLOCK 中出现了当前媒体文件生成指令，那么在下属 BLOCK 的 DPIDECLARATION 的作用域

中下属 BLOCK 中的当前媒体文件生成指令占主导地位；如果当前媒体文件生成指令出现在辅助 DPI 和文件 DPI 中，则在文件 DPI 的作用域中将由文件 DPI 取代辅助 DPI。

8.5.4 颜料选择指令

颜料选择文件生成指令指出了如何把物理颜料集与将要表示的文件页面联系起来。“颜料选择”这个词在本条中表示一个与物理颜料集有关的处理过程，这个物理颜料集早已由一条颜料文件生成指令指定了。

用于表示文件的颜料由名字来标识，颜料的选择依赖于两个参数：

——参数 colorants-select，它的值或为 NULL，或为一个名字序列；

——参数 current-colorants，它的值或为 NULL，或为一单个名字。

参数 colorants-select 的值由颜料选择文件生成指令来设置，参数 current-colorants 的值由当前颜料文件生成指令来设置。

如果一个文件有 n 个页面，那么 forl≤i≤n，根据参数 colorants-selected 和 current-colorants 按如下方法决定用于表示第 i 个页面的颜料名：

——如果参数 colorants-select 的值为 NULL，那么根据参数 current-colorants 的值来选择颜料；

——如果参数 current-colorants 也为 NULL，应当选择由系统决定的缺省颜料；

——如果参数 current-colorants 不为 NULL，那么根据参数 colorants-list 所建立的对应关系选择与参数 currents-select(i)的值相对应的颜料。

如果参数 colorants-select 和 current-colorants 都不为 NULL，那么就根据参数 current-colorants 所建立的对应关系选择与参数 colorants-select(i)或 current-colorants 的名字值相对应的物理颜料集，参数 colorants-list 由 8.4.2 中定义的颜料文件生成指令设置，如果 colorants-list 中没有一个元素能将一个物理颜料值与参数 current-colorant 或 colorant-list(i)的名字值联系起来，那么将出现结构警告并使用由系统决定的颜料集。

注：嵌套最深的颜料可取代文件 DPI，辅助 DPI 也可取代文件 DPI。

8.5.4.1 颜料选择 colorants select

颜料选择文件生成指令对于每个文件页面指出了用于显现的颜料。

颜料选择文件生成指令的名字应为 DPI/Colorants-Select。颜料选择文件生成指令的值的类型应为 RUN VECTOR，RUN VECTOR 所表示的值的类型应为 Name(名字)。

执行颜料选择文件生成指令的效果就是将参数 colorants-select 的值置成颜料选择文件生成指令的值，参数 colorants-select 的值应为 NULL 或一个名字序列，参数 colorants-select 的默认值是 NULL。

既然将颜料赋给一个 Colorants Description(颜料描述)的方法是依整于系统实现的，就不需要考虑回撤。

注：实际上系统可以把所有的颜料描述都赋值为缺省颜料。

颜料文件生成指令只有在辅助 DPI 中才是有意义的，任何出现在文件 DPI 中的颜料文件生成指令都被忽略。

8.5.4.2 当前颜料 current Colorants

当前颜料文件生成指令指出了用于表示一个 BLOCK 页面的一个颜料集，用来标识这个颜料集的名字已经有一个对应的颜料描述(colorants Description)，正如 8.4.2 中所指出的那样。

当前颜料文件生成指令的名字应为 DPI/Current Colorants，当前颜料文件生成指令的值的类型是 Name(名字)。

执行当前颜料文件生成指令的效果就是在 DPI DECARATION 的作用域中将参数 current-colorants 的值置成当前颜料文件生成指令的值，参数 current-colorants 的值的类型应为 Name(名字)或 NULL，参数 current-colorants 的默认值是 NULL，它表示使用系统确定的颜料。既然将颜料赋给一个颜料说明的方法是依赖于系统的，那么就不需要考虑回撤。

当前颜料文件生成指令在任何 PAGE,PAGESET 或 DOCUMENT 中都是有意义的,任何出现在 PICTURE 的而不是 PAGE 的 PROLOGUE 中的当前颜料文件生成指令将被忽视。如果在一个文件的多个 BLOCK 中出现了当前颜料文件生成指令,那么在下属 BLOCK 的 DPI DECLARATION 的作用域中,下属 BLOCK 中的当前颜料文件生成指令占主导地位。如果当前颜料文件生成指令出现在辅助 DPI 和文件 DPI 中,则在文件 DPI 的作用域中将由文件 DPI 取代辅助 DPI。

8.5.5 单双面 Plex

单双面文件生成指令指出了是在页式媒体(sheet medium)的一面还是两面输出页面图像,并且当两面都使用时,还指出了正面和反面页面图像之间的关系。单双面文件生成指令的名字应为 DPI/plex。单双面文件生成指令的值的类型是公用对象标识符。

执行单双面文件生成指令的结果就是将参数 plex 的值置为单双面文件生成指令的值,参数 plex 从总体上指出了每个实际物理媒体上将要使用的可成像区域的数目,以及可成像区域的引用坐标系与物理媒体之间的关系,定义单双面文件生成指令含义的值以及这些值的含义应为:

对象名	含　义
DPI/plex/simplex	值 DPI/plex/simplex 表示只使用被选页式媒体的正面(前面)
DPI/plex/duplex	值 DPI/plex/duplex 表示将使用被选页式媒体的正面和反面(前面和后面)两面的底边(RCS 的 x 轴)是媒体的同一条边
DPI/plex/tumbleplex	值 DPI/plex/tumbleplex 表示将使用被选页式媒体的正面和反面(前面和后面),并且正面的底边(RCS 的 x 轴)和反面的底边分别对应于媒体相对的两条边

plex 的默认值应为 DPI/plex/simplex,在一个不支持在被选页媒体的两面成像的系统如果想要这么做,那么系统将产生一个结构警告以表示出错,如果参数 abort-policy 的值允许发出结构警告后继续处理过程,那所作出的回撤行为应该是在被选页式媒体的单个可成像表面上表示所有页面,以便随后的脱机还原处理过程能够产生原来要在两个表面上表示的结果,这种成像应该提供必要的图像变换,包括 x 轴、y 轴的位移和旋转。

单双面文件生成指令在文件的任何 PAGESET 中都有意义。下属 PAGESET 中的单双面文件生成指令在下属 PAGESET 的文件生成指令的作用域中将取代上级 PAGESET 的单双面文件生成指令,如果单双面文件生成指令以辅助 DPI 的形式出现,辅助 DPI 将取代文件 DPI。

8.5.6 x-图像移动 x-Image-shift

x-图像移动文件生成指令指出了 Current Transformation 的一个初始化值,它从物理介质的边界开始移动页面图像,并沿着新的图像位置设置 RCS 的 y 轴,它用来位移页面图像以允许使用以后定义的操作。

x-图像移动文件生成指令的名字为 DPI/x ImageShift。x-图像移动文件生成指令的值的类型应为 Number。

执行 x-图像移动文件生成指令的结果就是将参数 x-image-shift 的值置为 x-图像移动文本生成指令的值,参数 x-image-shift 指出了 Current Transformation 的初始值的变化,这种变化由页数和参数 plex 的值决定。如果 page-number 是一个状态变量,它表示当前 PAGE 在文本顺序排序的 PAGE 序列中的顺序号,那么 x-image-shift 值为 x 的效果就是通过下列操作改变 PAGE 的当前解释上下文中 Current Transformation 的初始化值:

{0 x (Negate) plex "DPI/plex/duplex"Equal page-number 2 Remainder o Equal And if Translate CoucatT}

x-image-shift 的默认值应为零(0),它表示不改变 Current Transformation。对于不支持 x-图像移动文件生成指令的系统如果想要这么做,那么系统将发生结构警告以表示出错。如果参数 abort-policy 的值允许发生结构警告后继续进行处理,那么继续处理的结果等于缺省情况,本指令没有回撤行为。

x-图像移动文件生成指令在文件的任何 PAGESET 中都是有意义的，下属 PAGESET 中的 x-图像移动文件生成指令在它的作用域中将取代上级 PAGESET 中的 x-图像移动文件生成指令。

如果 x-图像移动文件生成指令以辅助 DPI 的形式出现，那么辅助 DPI 应取代文件 DPI。

8.5.7 y-图像移动 y-image-shift

y-图像移动文件生成指令指出了 Current Transformation 的一个初始化值，它从物理介质的边界开始移动页面图像，并沿着新的图像位置设置 RCS 的 y 轴，它用来位移页面图像以允许以后定义的操作。

y-图像移动文件生成指令的名字为 DPI/Y Image Shift。y-图像移动文件生成指令的值的类型是 Number。

y-图像移动文件生成指令的执行效果应是将参数 y-image-shift 的值置为 y-图像移动文本生成指令的值，参数 y-image-shift 指出了 CurrentTransformation 的初始化值的变化，这种变化由页数和参数 plex 的值决定。如果 page-number 是一个状态变量，它表示当前 PAGE 在文件顺序排序的 PAGE 序列中的顺序号，那么 y-image-shift 值为 y 的效果就是通过下面操作改变 PAGE 的当前解释上下文中 Current-Transformation 的初始化值：

{y (Negate) plex"DPI/plex/tumble-duplex"Equal Page-number 2 Remainder 0 Equal And If 0 Translate ConcatT}。

y-image-shift 的默认值应为零(0)，它表示不改变 CurrentTransformation。对于不支持 y-图像移动文件生成指令的系统如果强要这样做，那么系统将发生结构警告以表示出错。如果参数 abort-policy 的值允许发生结构警告后继续进行处理，那么继续处理的结果等于默认值。本指令没有回撤行为。

y-图像移动生成指令在文件的任何 PAGE SET 中都是有意义的，下属 PAGE SET 中的 y-图像移动文件生成指令在它的作用域中将取代上级 PAGE SET 中的 y-图像移动文件生成指令。

如果 y-图像移动文件生成指令以辅助 DPI 的形式出现，那么辅助 DPI 应取代文件 DPI。

8.5.8 双面输出 Only On Duplex

双面输出文件生成指令指出一个特定页面，只有在参数 plex 的值为 DPI/poxe/duplex 或 DPI/plex/tumbleDuplex 时才输出。

双面输出文件生成指令的名字应为 DPI/only-on-duplex，双面输出文件生成指令的值的类型应为 Boolean(布尔型)。

执行双面输出文件生成指令的效果就是把直接上级 PAGE 的参数 page-select 的值乘以值：

{plex"DPI/plex/duplex"Equal plex"DPI/plex/tuble-duplex"Equal or (1) (0) ifelse}

对于不能执行双面输出文件生成指令的系统如果强要这么做，那么系统将发生结构警告以表示出错。如果参数 abort-policy 的值允许发出结构警告后继续进行处理，那么继续处理的结果与系统有关，本指令没有回撤行为。

双面输出文件生成指令只有在 PAGE 中才有意义，任何出现在一个不是 PAGE 的 BLOCK 中或者以辅助 DPI 形式出现的双面输出文件生成指令应被忽略。

8.6 结束和发送指令

SPDL 定义了文件结束和输出发送文件生成指令，结束指令描述了与一个内容很广泛的操作集的接口，它可以满足各种要求，从生成简单的记录到装订好小册子。发送指令描述了与一组操作的接口，这些操作能提供诸如输出支架(output bin)选择与叠页位移(stacking offset)等功能。

8.6.1 间隔页面 break page

打印机在每个文件输出之前常常打印一个称为"间隔页面"的特殊页面。间隔页面一般包含一种特殊图案以便能够快速地辨别作业边界，页面还可标志打印作业的接受者，它也可以给出文件名、生成时间、打印时间、接受者名字以及描绘错误信息等等。打印在间隔页面上的信息也可以由外部来源提供，间隔页面的格式和内容是由系统决定的，间隔页面类型文本生成指令提供了对所要的间隔页面类型的控制。

间隔页面类型文件生成指令的名应为 DPI/Break Page Type。间隔页面类型文件生成指令的值的类型是公用对象标识符。

执行间隔页面类型文件生成指令的效果就是将参数 break-page-type 的值置为间隔页面类型文件生成指令的值。参数 break-page-type 指出了将要表示的间隔页面的类型。定义 break-page-type 含义的值以及这些值的含义应为：

对象名	含　义
DPI/BreakPageType/none	不对间隔页面进行表示
DPI/BreakPageType/terse	以包括最少信息的简短格式来表示间隔页面
DPI/BreakPageType/verbose	表示包含信息量最大的间隔页面

break-page-type 的默认值由系统决定。

打印设备可以取消间隔页面类型文件生成指令，间隔页面类型文件生成指令可被软拷贝设备忽略。

间隔页面类型文件生成指令只在 DOCUMENT 中或在辅助 DPI 中有意义，任何出现在一个不是 DOCUMENT 的 BLOCK 中的间隔页面类型文件生成指令都被忽略，如果间隔页面类型文件生成指令以辅助 DPI 的形式出现，辅助 DPI 应取代文件 DPI。

8.6.2 输出位置　output position

输出位置文件生成指令为每份拷贝指出了一个输出位置。输出位置信息以一种依赖于系统的方式映射到打印机的可用设施上，这种设施一般对应于类似输出托盘或支架的机制。

输出位置文件生成指令和叠页文件生成指令(8.6.3)在功能上是垂直的，输出位置文件生成指令选择发送拷贝的位置，叠页文件生成指令指出在某输出位置中如何将多份拷贝排序或隔离。

输出位置文件生成指令的名字应为 DPI/OutputPosition。输出位置文件生成指令的值应为值是整数的 RUN VECTOR 向量。

执行输出位置文件生成指令的效果就是将参数 output-position 的值置为输出位置文件生成指令的值，参数 output-position 是一个整数向量，它指出了用于每份拷贝的输出位置，forl≤i≤n，每 i 份拷贝使用由 output-position(i)决定的输出位置，从 output-position 值到实际输出位置的映射方法是由系统决定的。

output-position 的默认值是对于所有的 i 值 1≤i≤n，有(i)＝0，由值 0 决定的输出位置是由系统决定的。对于不能执行输出位置文件生成指令的系统如果强要这么做，系统应会发出结构警告以表示出错，如果参数 abort-policy 的值允许发出结构警告后继续进行处理，那处理的结果等价于默认值；本指令没有回撤行为。

输出位置文件生成指令只有在一个 SPDL 文件的最外层 BLOCK 中才有意义。如果辅助 DPI 和文件 DPI 都出现，那么辅助 DPI 应占主导地位。

注：系统如果不能提供象输出位置文件生成指令所指出的那么多不同的输出位置，那么它有可能将多个值映射到同一个输出位置，实际有效的输出位置的数目不需要超过 1；即系统可能将所有的输出位置都置为一个输出位置。

8.6.3 叠页处理　Stacking

叠页文件生成指令指出了如何组织打印输出以及是否要在各拷贝之间插入隔离页(separator sheet)。

叠页文件生成指令的名字应为 DPI/Stacking。叠页文件生成指令的值应为一个复合结构元素，它有以下直接下属：

——一个 PROPERTY LIST 结构元素。

PROPERTY LIST 指出了一组属性。这些属性指明了所要求做的叠页操作，每个属性称为叠页属性，由叠页文件生成指令指出的一套叠页属性称为叠页说明(stacking specification)。

执行叠页文件生成指令的效果应为参数 stacking-operations 的值置为叠页文件生成指令的值。参数 stacking-operations 指出了用于组织和发送打印输出的叠页说明。

各个定义叠页属性的条指出了：

——叠页属性的一般语义；

——叠页属性的名字；

——叠页属性值类型；

——指定了属性含义的值的集合；

——每个指定值的含义。

指定了含义的值的集合可能不包含所属类型的所有可能值。叠页说明可以为一个叠页属性指定一个值，只要这个值有恰当的类型，这个值就可以不是由 8.6.3.1 定义含义的那些值之一。这种叠页属性的效果由系统决定。

叠页说明可以为 8.6.3.1 中定义的任何叠页属性指定值。另外，叠页说明可能包含一个或多个 PROPERTY NAME 的值是没有在本标准中定义的公用对象变量值的结构元素 PROPERTY，这种 PROPERTY 的效果由系统决定。

对于不能执行叠页文件生成指令的系统如果强要这么做，则会发出结构警告以表示出错。如果参数 abort-policy 的值允许发出结构警告后继续进行处理，那么这种继续处理的后果由系统决定，本指令没有回撤算法。

叠页文件生成指令仅在最外层 BLOCK 中有意义，任何出现在其他地方的叠页文件生成指令都是无效的。如果一条叠页文件生成指令同时以辅助 DPI 和文件 DPI 的方式出现，那么辅助 DPI 将占主导地位。

8.6.3.1 叠页属性

下面的各条说明了本标准中定义的叠页属性，并指出了结构元素 PROPERTY 的语法，PROPERTY 指定了对应的叠页属性。

8.6.3.1.1 collated(整理)

叠页属性 Collated 能够控制输出是发送各页面的所有拷贝的集合(未整理)还是发送各拷贝的页面的集合(已整理的)。

叠页属性 Collated 的名字应为 DPI/stacking/collated。叠页属性 collated 的类型应为 Boolean。定义了叠页属性 Collated 含义的值以及这些值的含义应为：

值	含 义
True	以整理顺序发送输出
False	以非整理顺序发送输出

叠页属性 Collated 的默认值是 True。

8.6.3.1.2 offset(位移)

叠页属性 offset 能够控制在叠置多份拷贝时是否互相之间边边相联。

叠页属性 offset 的名字应为 DPI/stacking/offset。叠页属性 offset 的值的类型是 Boolean。定义了叠页属性 offset 的含义的值及这些值的含义应为：

值	含 义
True	每份拷贝按由系统决定的方式与他的邻接拷贝之间有一段可分辨的位移
False	没有间隔地叠置起来作为输出

叠页属性 offset 的默认值应为 False。

8.6.3.1.3 Slip Sheet(薄衬纸)

叠页属性 Slip sheet 能够控制是否在输出的各份拷贝之间插入一张薄衬纸。

叠页属性 SlipSheet 的名字应为 DPI/stacking/slipsheet，叠页属性 slipsheet 值的类型应为 Boolean。

定义了叠页属 slipsheet 的含义的值以及这些值的含义如下：

值	含 义
True	发送拷贝时有衬纸，具体的用来插入薄衬纸的媒体由系统决定
False	发送拷贝时没有薄衬纸

叠页属性 Slipsheet 的默认值应为 FALSE。

8.6.4 结束指令

所有可用的复杂结束操作构成一个内容广泛的、可扩充的集合，结束文件生成指令指出了与这个集合的一个接口，可用的结束操作可以包括纸页插入，模具切割，胶合和缝合装订。

8.6.4.1 结束操作抽象模型的描述

本条描述了结束操作的抽象模型以及用来说明结束文件生成指令的术语，为了能够清楚地理解后面的结束文件生成指令说明以及各条的内容，本条详细地描述了抽象模型，图 8-1 和图 8-2 有助于形象化地表达本条中提供的描述。

在本条创立的抽象模型中，术语“结束”一般指应用于文件的最后生成操作，这个操作发生在页式媒体成像之后，文件发货之前。一组一般由传统的印刷装订厂家提供的处理过程使用这个结束操作，每个处理过程称为结束处理(Finishing Process)。本标准在结束文件生成指令中显式地提供了一个名为 Finishing Operation 的属性来表示一个在结束操作期间使用的结束处理。

结束处理是一台机器用于文件的基本操作，例如修剪文件，折叠文件纸张，装订文件等等。从概念上说，不管进行结束处理的机器的能力如何，结束处理总是不时地以一种指定顺序对文件操作。在本标准中，每个结束处理以及所有要求用来指定其用途的参数都被集中到一个基本结构中，这个结构称为结束说明(Finishing Specification)。

适用于平行参考边的结束处理

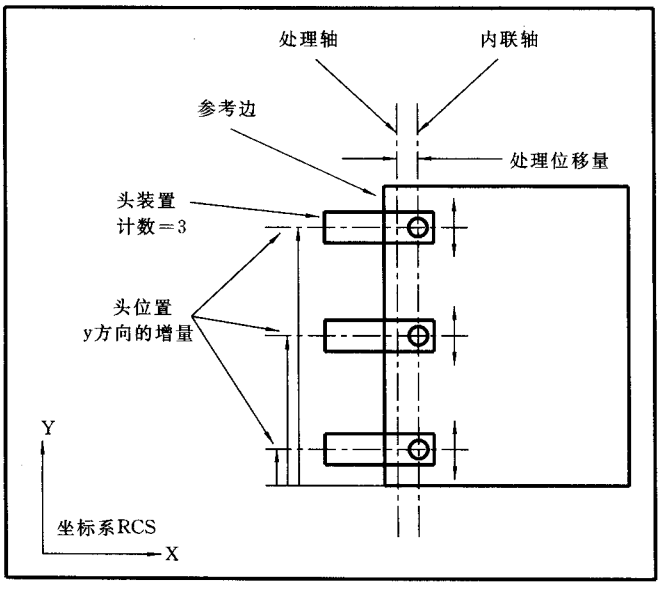


图 8-1

如果要说明多个结束处理，那么可以根据想要完成的应用序列按照结束文件生成指令的值定义一系列结束说明来做到这一点，必须仔细排列应用结束处理的次序，因为先把文件的纸张折迭起来修剪和先修剪然后再折叠所产生的结果是很不一样的，说明多重结束处理时总是根据想要完成的最后效果而说明的。

一个应用于文件的结束处理可能要由多个物理机构来完成，例如要进行边缘装订的结束处理可以

只用一个装订的物理机构，而进行三眼打孔的结束处理可能用到三个物理机构，每个用于结束处理的物理机构为头(Head)。在有些完成结束处理的机器中，头位置是静态的，而在功能更强的机器中头位置是可以移动的，只是受到某个具体机器的物理限制。对于那些功能更强的机器，所有的头一般可以互相之间排成一条线。头定位的物理限制意味着不能任意地设置头的位置。本标准中的结束文件生成指令的说明限制了这种能力，它指出了与大多数系统一致的头的位置。所有头沿一条理想的内联轴定位(inline-axis)，内联轴是一条假想的横切各个头的中心线，这条线贯穿一个给定头的确切位置并不正式指出，因为它与特定的结束处理有关。但是对于某个结束处理，内联轴贯穿每个头的位置在各个头的物理机构上都是一致的。

适用于垂直参考边的结束处理

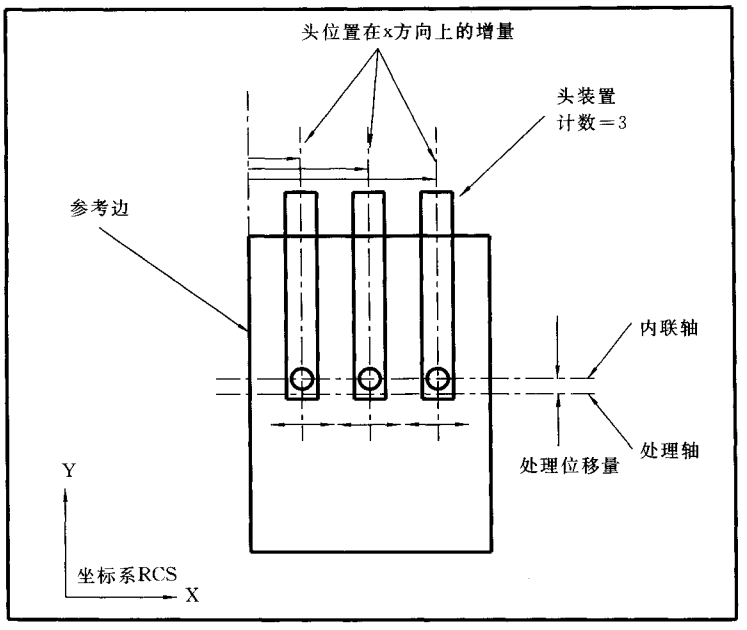


图 8-2

为了对某个结束处理的头进行定位，本标准隐含地以内联轴为参考线。在这一结束操作的抽象模型中，使用内联轴使这种限制变得很明显，当头都以垂直于内联轴的方向定位时就能校准相关联的头。

垂直于内联轴方向的头的移动是受到限制的，而沿着内联轴的线性方向移动没有正式地加以限制。对于头在内联轴线性方向上的移动的唯一限制是机器的物理能力。文件有一条边，结束处理都是相对于它进行的，称之参考边。因为传统的装订系统可以将文件重定位到任意一条边，所以把参考边限制为文件的某个特定边是不够的，参考边可以是文件的四条可识别的边中的任意一条，因而改变对应于一个给定结束处理的参考边隐含着对文件进行重定位，本标准在结束文件生成指令中显式地提供了一个称之为 Reference Edge 的结束属性，以便能够进行这种灵活的定位。

每个结束处理都定义了一条相对于参考边的轴，以这条轴为基准对头进行校正，这条轴可以平行于参考边也可以垂直于参考边，但它不能是任意角度。另外，这条轴常常设置在一个指定位置，这个位置可能相对于参考边有一个位移。这条轴的相对于参考边的角度和位置是指定结束处理的一个隐含特性，这个轴称为处理轴(Process Axis)。根据给定结束处理的定义，处理轴是固定不能改变的。

内联轴和处理轴有着特殊的对应关系，内联轴和处理轴常常是互相平行的，当根据默认方式进行结束处理时，内联轴和处理轴重合，如果头定位在附加的位移距离上，那么这个距离就被指定为内联轴和处理轴之间的距离，本标准在结束文件生成指令的属性中显式地提供了一个名为 Process Offset 的结束属性以便能够进行这种定位。

8.6.4.2 结束文件生成指令的结构

结束文件生成指令的名应为 DPI/Finishing，结束文件生成指令的值应该是一个包含下列直接下属的复合结构元素：

——零个或多个 PROPERTY LIST 结构元素。

每个结构元素 PROPERTY LIST 指出了一个属性的集合，该集合表示一个可以对文件进行操作的结束处理。各单个属性称为结束属性，结构元素 PROPERTY LIST 指出的结束属性的集合称为结束说明(Finish Specification)。

结束文件生成指令的执行效果就是将参数 finishing-operations 的值置为结束文件生成指令的值。参数 finishing-operations 指出了将在文件结束操作期间使用的结束说明的集合。在参数 finishing-operations 的值中可能有多个结束说明，在有多个结束说明的情况下，应该按照参数 finishing-operations 的值中所排列的次序依次处理各个结束说明，这种处理的结果就是按照指定顺序对文件依次进行各结束说明所定义的结束处理。

对于每个结束属性，在定义它的条中指出了：

——结束属性的一般语义；

——结束属性的名字；

——结束属性值的类型；

——指出了属性含义的值的集合；

——每个指定值的含义。

具有指定含义的所有值的集合可能并不包含所属类型的所有可能值。结束说明可以为一个结束属性指定一个值，只要这个值有恰当的类型，它就可以不是已定义了含义的那些值之一。这种结束属性的效果是由系统定义的。

结束文件生成指令的默认值就是不进行任何结束处理，对于不能执行结束文件生成指令或者不支持某些结束说明中的某些结束属性的系统如果强要这么做，系统应能发出结束警告以表示出错，如果参数 abort-policy 的值允许发出结构警告后继续进行处理，那么这种继续处理的结果由系统决定。

结束文件生成指令只有在 SPDL 文件的最外层 BLOCK 中才有意义，如果结束文件生成指令以文件 DPI 和辅助 DPI 的形式出现，那么辅助 DPI 应当为主导地位。

8.6.4.3 参考尺寸

结束属性 Reference Size 为相应结束说明中所有的媒体计算并指出了以毫米为单位的额定(媒体)尺寸。

结束属性 Reference Size 的名字应为 DPI/Finishing/ReferenceSize。结束属性 Reference Size 的值应为一个含有下列直接下属的复合结构元素：

——一个 MEDIUM X SIZE 结构元素；

——一个 MEDIUM Y SIZE 结构元素。

这些下属应以指定顺序出现。

MEDIUM X SIZE 和 MEDIUM Y SIZE 的值的类型应为 Number。如果属性 Reference Size 的 MEDIUM X SIZE 的值是 x，MEDIUM Y SIZE 的值是 y，那么意味着媒体尺寸为 x mm×y mm，并且引用坐标按下面的途径与实际媒体相联系：

——一条长为 x mm 的边对应于引用坐标系的正 x 轴；

——一条长为 y mm 的边对应于引用坐标系的正 y 轴。

Reference Size 的默认值是由页面选择文件生成指令选择的第一个页面的媒体尺寸。

8.6.4.4 参考边

结束属性 Reference Edge 指定了媒体的一条边，对应的结束说明中的一个结束处理就相对于它进行。

结束属性 Reference Edge 的名字应为 DPI/Finishing/Reference Edge，结束属性 Reference Edge 值的类型是 Integer。定义了结束属性 Reference Edge 的含义的值以及这些值的含义应为：

值	含　义
0	与 RCS 的 X 轴一致的边(底边)
1	平行于 RCS 的 Y 轴并离它最远的那条边(右边)
2	平行于 RCS 的 X 轴并离它最远的那条边(上边)
3	与 RCS 的 Y 轴一致的边(左边)

Reference Edge 的默认值为 3(左边)。

8.6.4.5 结束位置

结束属性 Finishing Locations 指出了一组位置，一个或多个头在相应位置上对页式介质进行结束处理。

结束属性 Finishing Locations 的名字应为 DPI/Finishing/Locations。结束属性 Finishing Locations 的值是一个包含下列直接下属的复合结构元素：

——一个 PROPERTY LIST 结构元素。

每个结构元素 PROPERTY 是一个指出了结束属性 Finishing Locations 的一个分量的结束属性，在本标准的 8.6.4.2.1 到 8.6.4.2.3 中指出了结束属性 Finishing Locations 中应当有的那些结束属性

8.6.4.5.1 头数

结束属性 Head Count 指出了相应结束说明中的结束处理将要用到的头的数目。

结束属性 Head Count 的名字应为 DPI/Finishing/Loctions/HeadCount。结束属性 HeadCount 值的类型应为 Integer。如果结束属性 Head Count 的值是 0，或 Head Count 的值未指定，那么不进行任何结束处理。

结束属性 Head Count 的默认值应为 0。

8.6.4.5.2 处理偏移

结束属性 Process Offset 以毫米为单位指出了从内联轴到处理轴的偏移量，结束属性 ProcessOffset 的变动会使内联轴上所有的头统一地进行重定位。

结束属性 Process Offset 的名字应为 DPI/Finishing/Locations/ProcessOffset。结束属性 Process Offset 值的类型应为 Number。如果结束属性 Process Offset 的值为 0，或者未指定，那么进行相应的结束说明中的结束处理时内联轴与处理轴是重合的。

结束属性 Process Offset 的默认值为 0。

8.6.4.5.3 头位置

结束属性 Head Locations 指出了多个可用头所处的位置。头的中心处于内联轴上，并且所有的头按照结束属性 Head Locations 所指出的位置沿着内联轴线性定位。

结束属性 Head Locations 的名字应为 DPI/Finishing/Locations。结束属性 Head Locations 的值应为一个包含下列直接下属的复合结构元素：

——零个或多个 NUMBER 结构元素。

结构属性 Head Locations 的 PROPERTY LIST 值中的结构元素构成了一个位置的有序集，每个结构元素应该是一个数值(Number)，并沿内联轴方向以递增的 X 或 Y 指出位移量，将引用坐标系原点与处理轴联系起来。

如果结束属性 Reference Edge 的值是 0 或 2，并且相应结束说明中结束处理的处理轴平行于参考边，那么结束属性 Head Locations 的 PROPERTY LIST 值中的结构元素 Number，应当是按 X 升序的距引用坐标系原点的位移量，如果处理轴垂直于参考边，那么这些值应当是按 Y 升序的距引用坐标系原点的位移量。

如果结束属性 Reference Edge 的值是 1 或 3，并且相应结束说明中结束处理的处理轴平行于参考

边，那么结束属性 Head Locations 的 PROPERTY LIST 值中的结构元素 Number 应当是按 Y 升序的距引用坐标系原点的位移量；如果处理轴垂直于参考边，那么这些值应当是按 X 升序的距引用坐标系原点的位移量。

结束属性 Head Locations 的 PROPERTY LIST 值中的结构元素数目应当与结束属性 HeadCount 的值所指出的位置数目一样。如果结束属性 Head Locations 的 PROPERTY LIST 值中的结构元素数目比结束属性 Head Count 的值所指出的位置数目多，那么多余的结构元素应被忽略。

结束属性 Head Locations 的默认值依赖于具体的结束处理。

8.6.4.6 插入

结束属性 Inserts 指出了一个由一个或多个插入纸页构成的集合。

所有的插入纸页必须有一个标准方向，上边界、底边界、左边界、右边界以及上表面和下表面。对于每个要插入的纸页，标记设备先标记完在此之前的物理纸页，然后引入预先打印好的插入纸页，这样该纸页便插入到整理好的已打印纸页的集合中(有些插入机构可能等到作业的所有页面都标记完之后才插入纸页)。

纸页插入处理使用了一种插入机构，它一般能够从几个输入支架的一个或多个中插入纸页。插入纸页不用媒体指令来识别，也不通过专为标记机械本身所设计的输入支架来输入给结束处理。

结束属性 Inserts 的名字应为 DPI/Finishing/Inserts，结束属性 Inserts 的值应为一个包含下列直接下属的结构元素：

——零个或多个 PROPERTY LIST 结构元素。

每个结构元素 PROPERTY LIST 指出了一个结束属性的集合，它定义了一个将由插入结束处理插入到文件中去的纸页。插入操作在结束属性 Finishing Operation 的值中以 DPI/Finishing/Inserting 的形式指定，正如 8.6.4.11 中所定义的那样，在本标准的 8.6.4.5.1 到 8.6.4.5.6 指定了结束属性 Inserts 的值中应当有的结束属性。

8.6.4.6.1 插入名

结束属性 Insert Name 指出了要插入的所用纸页的名字，这个名字唯一地对应一个纸页，该页面应该是预先打印好了的。

结束属性 Insert Name 的名字应为 DPI/Finishing/Inserts/InsertName，结束属性 InsertName 值的类型应为 ReferenceName。

InsertName 没有默认值。

8.6.4.6.2 插入尺寸

结束属性 Insert Size 以毫米为单位指出了插入纸页的尺寸。

结束属性 Insert Size 的名字应为 DPI/Finishing/Inserts/InsertSize，结束属性 InsertSize 的值应当是一个包含下列直接下属的复合结构元素：

——一个 INSERT X SIZE 结构元素；

——一个 INSERT Y SIZE 结构元素。

这些下属应当以指定顺序出现。

INSERT X SIZE 和 INSERT Y SIZE 的值的类型应为 Number，属性 InsertSize 中 INSERT X SIZE 的值为 x，INSERT Y SIZE 值为 y，那么插入纸页的尺寸应为 x mm×y mm，并且引用坐标系按这种途径映射到具体的一张纸页上：

——长为 x mm 的边对应于引用坐标系的正 x 轴；

——长为 y mm 的边对应于引用坐标系的正 y 轴。

结束属性 Insert Size 的默认值应为在相应结束说明中的结束属性 ReferenceSize 的值。

8.6.4.6.3 插入边

结束属性 Insert Edge 指出了插入纸页的一条边，它应当与结束属性 ReferenceEdge 的值一致。

结束属性 Insert Edge 的名字应为 DPI/Finishing/Inserts/InsertEdge，结束属性 Insert Edge 值的类型应为 Integer。定义了结束属性 Insert Edge 的含义的值以及值的含义应为：

值	含　义
0	与 RCS 的 X 轴一致的边(底边)
1	与 RCS 的 Y 轴平行并离得最远的边(右边)
2	与 RCS 的 X 轴平行并离得最远的边(上边)
3	与 RCS 的 Y 轴一致的边(左边)

Insert Edge 的默认值为 1(左边)。

8.6.4.6.4　插入面

结构属性 Insert Top Surface 指出了插入纸页的哪一面与文件中其他纸页的正常阅读顺序的方向相一致。

结构属性 Insert Top Surface 的名字应为 DPI/Finishing/Inserts/InsertTopSurface，结构属性 Insert Top Surface 值的类型是 Integer。定义了结束属性 Insert Top Surface 含义的值以及值的含义应为：

值	含　义
0	插入纸页的上表面与文件中其他纸页的正常阅读的方向一致
1	插入纸页的下表面与文件中其他纸页的正常阅读顺序的方向一致

结束属性 Insert Top Surface 的默认值为 0(上表面)。

8.6.4.6.5　插入支架号

结束属性 Insert Bin 指出了进纸机构的一个支架号，从这个支架上输入纸页。支架从 0 开始编号，最大的支架号依赖具体系统。

结束属性 Insert Bin 的名字为 DPI/Finishing/Inserts/InsertBin，结束属性 InsertBin 的值的类型应为 Integer。

InsertBin 没有默认值。

8.6.4.6.6　插入页号

结束属性 Insert After 指出了 SPDL 文件的一个 PAGE 号，插入纸页将插到该页之后。

结束属性 Insert After 的名字应为 SPI/Finishing/Inserts/InsertsAfter，结束属性 Insert After 的值的类型为 Integer。如果参数 plex 的值是 DPI/plex/Duplex 或 DPI/Plex/tumbleDuplex，插入纸页便插入到打印 PAGE 图像的那个媒体的后面。

Insert After 没有默认值。

8.6.4.7　实际尺寸

结束属性 TrimmedSize 为一组纸页中的所有纸面指出了对它们进行修剪结束处理后得到的最终区域。正如 8.6.4.11 中所定义的那样，在结束属性 Finishing Operation 的值中用 DPI/Finishing/Trimming 来定义修剪操作，本节中的图 8-3 有助于形象化地说明结束属性 Trimmed Size。

结束属性 Trimmed Size 的名字应为 DPI/Finishing/TrimmedSize，结束属性 TrimmedSize 的值应为一个包含下列直接下属的复合结构元素：

——一个 TRIM X SIZE 结构元素；

——一个 TRIM Y SIZE 结构元素；

——一个 REFERENCE EDGE OFFSET 结构元素。

这些下属应该是以指定顺序出现。

修剪尺寸文本生成指令

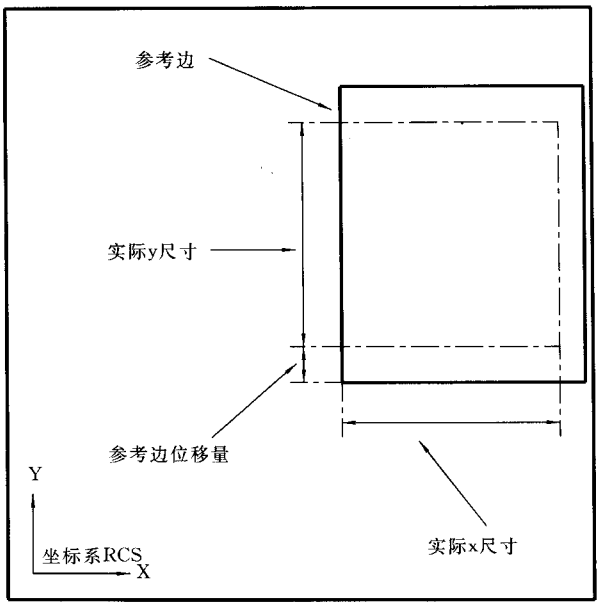


图 8-3

TRIM X SIZE、TRIM Y SIZE 和 REFERENCE EDGE OFFSET 的值的类型是 Number。如果属性 Trimmed Size 的 TRIM X SIZE 的值是 x，TRIM Y SIZE 的值是 y，REFERENCE EDGE OFFSET 的值为 z，那么意味着修剪后的纸张尺寸应为 x mm×y mm，并且进行的修剪操作距离相应结束说明中的参考边有一个 z mm 的位移量，引用坐标系按下面的方法与一个具体媒体联系起来：

——长为 x mm 的边对应于引用坐标系中的正 X 轴；

——长为 y mm 的边对应于引用坐标系中的正 Y 轴。

结束属性 Trimmed Size 的 TRIM X SIZE 和 TRIM Y SIZE 的默认值应为相应结束说明中的结束属性 Reference Size 的值，结束属性 Trimmed Size 的 REFERENCE EDGE OFFSET 的默认值应为 0。

8.6.4.8 切割名

结束属性 Die Cut Name 指出了将由模具切割结束处理进行的一个模具切割名、正如 8.6.4.11 所描述的那样，在结束属性 Finishing Operation 的值中用 DPI/Finishing/DieCutting 来指出模具切割操作。

结束属性 Die Cut Name 的名字应为 DPI/Finishing/DieCutName，结束属性 Die Cut Name 的值的类型应为 Reference Name。

Die Cut Name 没有默认值。

8.6.4.9 切割位置

结束属性 Die Cut Position 以毫米为单位指出了一个将由模具切割结束处理进行的模具切割的尺寸和位置。正如 8.6.4.11 中指出的那样，在结束属性 Finishing Operation 的值中用 DPI/Finishing. DieCutting 来指出模具切割操作，本节中图 8-4 有助于形象化地说明结束属性 Die Cut Position。

模具切割位置文本生成指令

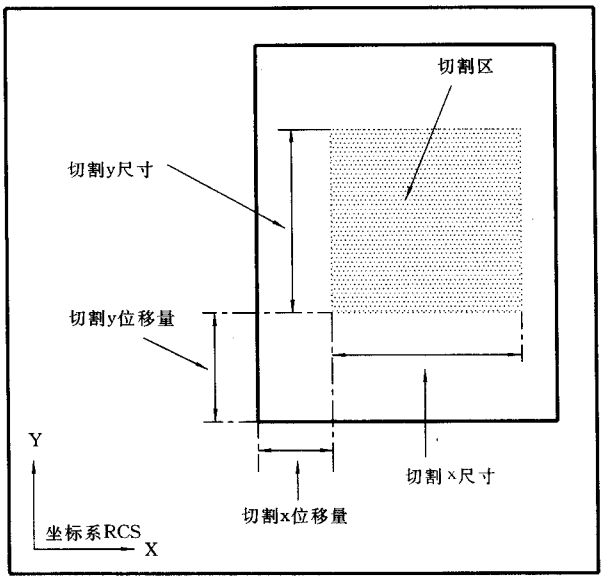


图 8-4

结束属性 Die Cut Position 的名字应为 DPI/Finishing/DieCutPosition，结束属性 DieCutPosition 的值应该是一个包含下列直接下属的复合结构元素：

——一个 DIE CUT X SIZE 结构元素；

——一个 DIE CUT Y SIZE 结构元素；

——一个 DIE CUT X OFFSET 结构元素；

——一个 DIE CUT Y OFFSET 结构元素。

这些下属应当以指定顺序出现。

DIE CUT X SIZE. DIE CUT Y SIZE，DIE CUT X OFFSET 和 DIE CUT Y OFFSET 的值的类型是 Number，如果属性 Die Cut Position 的 DIE CUT X SIZE 的值为 x，DIE CUT Y SIZE 是 y，DIE CUT X OFFSET 是 x′，DIE CUT Y OFFSET 是 y′，那么意味着模具切割的尺寸就包含在一个 x mm×y mm 的区域中，并且这块区域的位置离引用坐标系原点的位移为 x′mm×y′mm。

结束属性 Die Cut Position 的 DIE CUT X SIZE，DIE CUT Y SIZE，DIE CUT X OFFSET 和 DIE CUT Y OFFSET 的默认值应为 0。

8.6.4.10 装订类型

结束属性 Binding Type 指出了将由装订结束处理进行的装订的类型。正如 8.6.4.11 中所指出的那样，在结束属性 Finishing Operations 的值中用 DPI/Finishing/Binding 来指出装订操作，装订的类型隐含着使用由处理过程决定的相应的材料。

结束属性 Binding Type 的名字应为 DPI/Finishimg/BindingType，结束属性 BindingType 的值的类型应为 ReferenceName。定义为公用对象标识符值的对象名以及这些值的含义应为：

对象名	含　义
DPI/Finishing/BindingType/tape	使用带子包扎
DPI/Finishing/BindingType/plastic	使用塑料包扎
DPI/Finishing/BindingType/velour	使用丝绒面革包装
DPI/Finishing/BindingType/perfect	使用一种通常称之为“perfect”的装订方式

DPI/Finishing/BindingType/spiral	使用螺旋式装订
DPI/Finishing/BindingType/default	使用由系统决定的装订类型

binding Type 的默认值应为 DPI/Finishing/BindingType/default。

8.6.4.11 装订颜色

结束属性 Binding Color 指出了装订结束处理时所用的颜色，正如 8.6.4.11 中所定义的那样，在结束属性 Finishing Operation 的值中用 DPI/Finishing/binding 来指出装订操作。对于某些装订材料也许结束属性 Binding Color 值并不适用。

结束属性 Binding Color 的名字应为 DPI/Finishing/Binding Color。结束属性 Binding Color 的值的类型应为 Reference Name。定义为公用对象标识符值的对象名以及这些值的含义应为：

对象名	含　义
DPI/Finishing/BindingColor/black	黑色装订材料
DPI/Finishing/BindingColor/blue	蓝色装订材料
DPI/Finishing/BindingColor/gray	灰色装订材料
DPI/Finishing/BindingColor/brown	棕色装订材料
DPI/Finishing/BindingColor/defaul	系统决定的颜色的装订材料

Binding Color 的默认值是 DPI/Finishing/BindingColor/default。

8.6.4.12 结束操作

结束属性 Finishing Operation 指出了一个适用于文件的结束处理，结束属性 Finishing Operation 的值指定了结束处理，影响这个处理过程的值就是那些由相应结束说明中的有关结束属性所指出的值。与结束属性 Finishing Operation 值中定义的任何结束处理有关的结束属性和值都在本标准的 8.6.4.2 到 8.6.4.10 中指出了。

结束属性 Finishing Operation 的名字应为 DPI/Finishing/FinishingOperation。结束属性 Finishing Operation 的值的类型应为 ReferenceName。定义为公用对象标识符值的对象名以及这些值的含义应为

对象名　DPI/Finishing/convenienceStaple

含义：　简便订书操作。指出了一个以系统决定的方式装订一个订书针的结束处理。默认行为就是在一个角落上装订，方向与当地习惯一致。

简便订书操作不要求附加参数，并在没有提供其他复杂结束能力的命令的环境中使用。

简便订书操作不能与其他复杂结束指令一起使用。

对象名　DPI/Finishing/edgeStitching

含义：　边缝合操作。指出了进行一条以上缝合的结束处理。默认行为是把缝合线定位在平行于参考边并靠近它的位置，进行缝合结束处理的头通过 Head Locations 的值定位。如果进行边缝合操作的头只能在 Process Offset 的一个有限范围内定位，那么系统应使用一个固定的、合理的默认位移，并发出一个结构警告。

对象名　DPI/Finishing/binding

含义：　装订操作。指出了一个使用适当材料进行装订的结束处理，这个装订材料由结束属性 Binding Type 和 Binding Color 共同指定。默认行为就是将装订线定位在平行于参考边并靠近它的位置，进行装订的头由 Head Locations 的值来定位。如果进行装订操作的头只能在 Process Offset 的有限范围中定位，在那个系统应使用一个固定的、合理的缺省位移，并发出一个结构警告。

对象名　DPI/Finishing/saddleStitching

含义：　鞍形缝合操作。指出了进行一圈一个或多个针脚缝合的结束处理，圈的长度等于参

考边的长度。默认行为就是将缝合线定位在平行于参考边，并位于参考边及其他边的中点的位置，进行鞍形缝合的头由 Head Locations 的值定位，圈上的针脚的数目由 Head Count 的值来决定。如果进行鞍形缝合的头只能在 Process Offset 的有限范围中定位，那么系统应该使用一个固定的、合理的缺省位移，并发出一个结构警告。

对象名　DPI/Finishing/punching

含义：　打孔操作。指定了一个可打一个或多个孔的结束处理。默认行为是将所打的孔定位在平行于参考边并靠近它的位置，进行打孔的头通过 Head Locations 的值来定位。

如果进行打孔操作的头只能在 Process Offset 的有限范围内定位，系统应使用一个固定的、合理的缺省位移，并发出一个结构警告。

对象名　DPI/Finishing/Perforating

含义：　穿孔操作。指出了进行一圈或多圈穿孔的结束处理，一圈上穿孔的数目由系统决定，圈的长度等于垂直于参考边的长度。默认行为就是把穿孔位置定在垂直于参考边的方向，进行穿孔的头通过 Head Locations 的值沿着参考边定位。如果进行穿孔操作的头只能在 Process Offset 的有限范围内定位，系统应使用一个固定的、合理的缺省位移，并发出一个结构警告。

对象名　DPI/Finishing/Slitting

含义：　切纸操作。指出了一个切割一叠经过一个或多个结束操作的纸页的结束处理，切纸线的长度等于垂直于参考边的边的长度。默认行为就是将切纸线定位为垂直于参考边，进行切纸的头由 Head Locations 值沿参考边定位。切纸操作改变了参考尺寸，随后的依赖于参考尺寸的操作都会受到影响，任何随后的结束说明都应使用基于切纸操作结果的参考尺寸的校正值。如果进行切纸操作的头只能在 Process Offset 的有限范围中定位，那么系统将使用固定的，合理的默认值并发出一个结构警告。

对象名　DPI/Finishing/trimming

含义：　修剪操作。指出了一个结束处理，它切割一叠被处理纸张的除了参考边所指出的边之外的所有边。结束属性 Trimmed Size 指出了进行修剪结束处理的机构的位置，不管物理处理是否由一个有三个刀片的修剪器完成，修剪操作的效果总是沿三条边修剪，修剪操作改变了参考尺寸，随后的依赖于参考尺寸的操作都可能受到影响，任何随后的结束说明都应使用基于修剪操作结果的参考尺寸的校正值。

对象名　DPI/Finishing/folding

含义：　折叠操作。指出了一个进行折叠过程的结束处理。折叠线的长度等于垂直于参考边的边的长度。

默认行为就是将折叠线定在垂直于参考边的方向，进行折叠操作的头用 Head Locations 的值沿参考边定位。

折叠操作改变了参考尺寸，随后的依赖于参考尺寸的操作可能受到影响，任何随后的结束说明都应使用基于折叠操作结果的参考尺寸的校正值，如果进行折叠操作的头只能在 Process Range 的范围中定位，那么系统应使用一个固定的合理的缺省位移并发出一个结构警告。

对象名　DP/Finishing/inserting

含义：　插入操作指出了一个将纸面插入到 PAGE 的已整理的图像集合中的结束处理。结束属性 Inserts 指出了插入操作所需要的所有信息，以便识别出在什么位置插入什么纸页。

对象名　DPI/Finishing/dieCutting

含义：　模具切割操作指出了一个进行模具切割的结束处理，这种切割受到预定义的形状的限制。

结束属性 Die Cut Name 和 Cut Position 指出了模具切割操作所要求的所有信息，以便识别正确的模具切割位置。

结束属性 FinishingOperation 的默认值应为 NULL，这个默认值显式说明了不进行任何结束操作。

8.6.4.13　结束消息

结束属性 Finishing Message 指定了一个可打印串，它将在打印文件前通过操作员接口（如果有的话）进行显示，而且与文件打印的时间尽可能接近。

结束属性 Finishing Message 的名字应为 DPI/Finishing/Message，结束属性 Finishing Message 的值的类型应为 6.5 中定义的 PrintableString 类型。

结束属性 Finishing Message 没有默认值，对于不能显示可打印串的系统如果硬要这么做，那么系统将给出结构警告以表示出错。

8.7　管理指令

管理指令实现对文件生成过程的管理。

8.7.1　文件注释

文件注释文件生成指令指出了一个打印在间隔页面上（如果有的话）的可打印串。文件注释文件生成指令的名字应为 DPI/DocumentComment，文件注释文件生成指令的值的类型应为 6.5 中定义的 PrintedString。

文件注释文件生成指令没有默认值，不打印间隔页面的系统将忽略文件注释文件生成指令，不能打印全部可打印串的系统应在间隔页面上打印尽可能多的可打印串。文件注释文件生成指令只有在一个 SPDL 文件的最外层 BLOCK 中才有意义，如果一条文件注释文件生成指令以辅助 DPI 形式出现，那么辅助 DPI 应取代文件 DPI。

8.7.2　文件开始消息

文件开始消息文件生成指令指出了一个可打印串，它将在打印文件前输出到操作员接口（如果有的话），并且被确认接收，这个时间必须与文件打印的时间尽可能接近。文件开始消息文件生成指令的名字应为 DPI/DocumentStartMessage，文件开始消息文件生成指令的值的类型应为 6.5 节中定义的 PrintableString。

文件开始消息文件生成指令没有默认值，对于不能显示可打印串的系统如果强要这么做，那么系统会发出结构警告，表示出错。

文件开始消息文件生成指令只有在一个 SPDL 文件最外层的 BLOCK 中才有意义，如果它以辅助 DPI 形式出现，那么辅助 DPI 应取代文件 DPI。

8.7.3　文件结束消息

如果文件结束消息文件生成指令指出了一个可打印串，它将在打印文件后输出到操作员接口（有的话），并且被确认接收，这个时间应当与文件打印时间尽可能接近。

文件结束消息文件生成指令的名字应为 DPI/DocumentEndMessage，文件结束消息文件生成指令的值的类型应为 6.5 节中定义的 Printable String。

文件结束消息文件生成指令没有默认值。对于不能显示可打印串的系统如果强要这么做，那么系统将产生一个结构警告以表示出错。

文件结束消息文件生成指令仅仅在一个 SPDL 文件的最外层 BLOCK 中才有意义，如果它以辅助 DPI 形式出现，那么辅助 DPI 应取代文件 DPI。

8.7.4　超时

超时文件生成指令指出了允许文件解释和表示过程所用的最大时间，从开始解释表示的文件起计

时，如果表示处理现在正在执行其他任务，可以扩展这个时间。如果超过了这个时间，文件打印将失败。

超时文件生成指令的名字应为 DPI/TimeOut，超时文件生成指令的值的类型为 Integer。这个值以秒为单位。

超时文件生成指令的默认值由系统决定，但不能少于 360。

超时文件生成指令仅仅在一个 SPDL 文件的最外层 BLOCK 或辅助 DPI 中才有意义，如果它以辅助 DPI 形式出现，辅助 DPI 应取代文件 DPI。

8.7.5 文件失败策略

文件失败策略文件生成指令指出了在文件表示期间出现错误和异常所带来的影响。这种错误和异常可分为“结构警告”和“结构错误”两类。

文件失败策略文件生成指令的名字应为 DPI/AbortPolicy。文件失败策略文件生成指令的值的类型应为公用对象标识符。这个值指出了文件表示期间出现错误和异常所产生的影响。

执行文件失败策略文件生成指令的效果就是将参数 abort-policy 的值置为文件失败策略文件生成指令的值，定义了 about-policy 的含义的值以及值的含义应为：

值	含 义
DPI/AbortPolicy/OnWarning	任何结构警告或结构错误发生后文件表示应该失败
DPI/AbortPolicy/OnError	出现任何结构错误都使文件表示失败，但不是任何结构警告都会使文件表示失败
DPI/AbortPolicy/StuggleOn	即使出现了结构错误或结构警告，表示处理也将尽最大努力继续表示文件

文件失败策略文件生成指令的默认值是由系统决定的。所有的系统都应实现本标准定义的值以及相应的策略。如果 DPI DECLARATION 指定了一个系统不支持的值，那么就使用默认值。

文件失败策略文件生成指令仅仅在一个 SPDL 文件的最外层 BLOCK 或一个辅助 DPI 中才是有意义的，如果它以辅助 DPI 的形式出现。那么辅助 DPI 应取代文件 DPI。

9 文件结构和内容处理

一个 SPDL 文件的表示输出是通过表示过程解释 SPDL 文件内容得到的，这一过程由内容处理器(Content Processor)完成。

TOKENSEQUENCE 结构元素的值是一个八位字节串，它包含了 SPDL 内容标记(token)序列。这个八位字节串称为 TOKENSEQUENCE 的内容(content)，TOKENSEQUENCE 结构元素的定义见 9.1。

除了待处理的内容本身以外，内容处理器还需要一个处理内容元素的解释上下文(Context of Interpretation)。每次进行内容处理时，结构处理器总是把一个解释上下文同内容元素一起传递给内容处理器。解释上下文的定义见 9.2。

每个 BLOCK 有一个相关的当前解释上下文，它用于其下属内容元素的解释。BLOCK 的当前解释上下文在 BLOCK 处理的开始时建立，在 BLOCK 处理结束时撤消。BLOCK 处理与 BLOCK 的当前解释上下文之间的关系见 9.3。

内容处理器的接口是一个抽象的过程调用。过程调用的参数以及内容处理器的返回值详见 9.4。内容处理对当前页面图像的作用见 9.5。结构边界(structure boundaries)的作用在 9.6 中讨论。

9.1 TOKENSEQUENCE(标记序列)

TOKENSEQUENCE 结构元素是一个基本结构元素，其值的类型为八位字节串。TOKENSEQUENCE 的值，内容处理器把它看作是一个 SPDL 内容标记法中的标记序列而加以解释。

9.2 解释上下文 Context of Interpretation

解释上下文包括：

——一个可用资源集；

——虚拟机的状态。

可用资源集中的资源可以在处理某一特定的内容元素时使用。

虚拟机状态依次包括：

——一个上下文栈(Context Stack)；

——一个操作数栈(Operand Stack)；

——一个状态变量的集合。

上下文栈中包含一个有序词典的集合，以及被这些词典中的对象引用(ObjectReference)所直接或间接引用的所有数据结构的内容。

操作数栈包含一个有序对象的集合，以及由操作数栈中的对象引用所直接或间接引用的所有数据结构的内容。

状态变量的定义见第 12 章。

9.3 BLOCK 的当前解释上下文

BLOCK 的当前解释上下文在 BLOCK 处理开始时建立，在 BLOCK 处理完毕时撤消。

BLOCK 要么是一个 DOCUMENT，要么是某上级 BLOCK 的直接下属。若 BLOCK 是 DOCUMENT，则当前解释上下文的初始值为：

——一个可用资源的空集；

——如 9.3.2 中所定义的默认上下文栈；

——如第 12 章中所定义的状态变量的初值；

——一个空的操作数栈。

若 BLOCK 不是 DOCUMENT，则其当前解释上下文的初值为：

——一个可用资源的集合，其中的资源由 BLOCK 上下文中的 RESOURCE DECLARATION 所标识；

——上下文栈，它是在处理最直接上级 BLOCK 的 PROLOGUE(序言部分)结束时给出的值；

——状态变量的值，它是最直接上级 BLOCK 的当前解释上下文的状态变量的值；

——一个空的操作数栈。

BLOCK 的当前解释上下文可能随处理其下属 BLOCK 的结果而改变。一个 BLOCK 的当前解释上下文与其最直接上级的当前解释上下文是有区别的。对一个 BLOCK 的当前解释上下文的改变不影响其最直接上级 BLOCK 的当前解释上下文。

9.3.1 可用资源集

可用资源集中的资源是在 BLOCK 的上下文中由 RESOURCE DECLARATION 结构元素所标识的资源，或是先于当前 TOKENSEQUENCE 的 BLOCK 的 PROLOGUE 中由 RESOURCE DECLARATION 结构元素所标识的资源。

9.3.2 虚拟机的状态

9.3.2.1 上下文栈

BLOCK 的初始当前解释上下文中，上下文栈为：

〈User Dict:Dictionary〉

〈System Dict:Dictionary〉

System Dict 和 User Dict 的定义见 10.1.4。一个 BLOCK 的当前解释上下文的上下文栈可以在处理 CONTEXT DECALRATION 时进行修改。

9.3.2.2 操作数栈

操作数栈包含一个有序对象的集合，以及由这些对象直接或间接引用的所有数据结构的内容。

一个 BLOCK 的初始当前解释上下文的操作数栈不包含任何对象。对 TOKENSEQUENCE 结构的

处理过程可能改变 BLOCK 的当前解释上下文的操作数栈。DICTIONARY GENERATOR 或 SETUP PROCEDURE 下属的解释上下文的初始虚拟机状态中的操作数栈也可以在处理一个或多个下属 TOKENSEQUENCE 的结构的过程中进行修改。

9.3.2.3 状态变量

状态变量定义见第 12 章。一个 BLOCK 的当前解释上下文的状态变量的初值决定于 BLOCK 是 DOCUMENT 还是其上级 BLOCK 的下属。

——若 BLOCK 是 DOCUMENT，状态变量的初值定义见第 12 章；

——若 BLOCK 是另一 BLOCK 的下属，状态变量的初值与其最直接上级 BLOCK 的初始当前解释上下文中的状态变量的初值相同。

一个 BLOCK 的当前解释上下文的虚拟机状态的状态变量可以在处理 SETUP PROCEDURE 或 TOKENSEQUENCE 结构元素时被修改。

9.4 内容处理器的接口

调用内容处理器处理 TOKENSEQUENCE 结构元素的内容时，结构处理器将下列内容传送给内容处理器：

——一个八位字节串；

——一个解释上下文。

八位字节串是 TOKENSEQUENCE 结构元素的值，解释上下文在结构处理时与 TOKENSEQUENCE 相联系。

若 TOKENSEQUENCE 是一个 PICTURE 的直接下属，结构处理器也把最直接上级 PAGE 的当前页面图像传送给内容处理器。

内容处理器把八位字节串解释为内容标记法中的一个标记序列。内容处理可以影响虚拟机的状态，它是结构处理器传送来的解释上下文的一部分。若 TOKENSEQUENCE 是 PICTURE 的直接下属，内容处理过程也可能影响结构处理器传送来的当前页面图像。

内容处理完毕，内容处理器将返回：

——解释上下文；

——当前页面图像(若结构处理器传送来的话)；

——状态。

解释上下文是内容处理器作用于由结构处理器传送来的解释上下文所产生的结果。当前页面图像是内容处理器作用于由结构处理器传送来的当前页面图像所产生的结果。状态是以下内容之一：

——无错误或警告；

——内容警告；

——内容出错。

异常处理详见第 24 章。

9.5 当前页面图像 Current Page Image

当前页面图像受对 PICTURE 的直接下属 TOKENSEQUENCE 结构元素的处理过程的影响。对其他非 PICTURE 直接下属 TOKENSEQUENCE 结构元素的处理只影响虚拟机的状态，而不影响当前页面图像。

SPDL 文件的每个 PAGE 与一个单独的表示媒体实例相关。相应地，当前页面图像在处理各 PAGE 之初初始化为空页面。媒体指默认媒体或由文件生成指令给出的媒体。表示一个空页图像，其结果是媒体的自然色彩。

解释 PAGE 下属 TOKENSEQUENCE 结构元素的结果是去修改页面图像，正如第 7 章中所述。单双面上页面图像的实际表示可能是随着 TOKENSEQUENCE 结构元素的处理而一步步生成的；也可能是对页面图像不断进行修改从而在当前页画图像中逐步积累直到 PAGE 的处理结束，最后将页面图

像传送给媒体。PAGE 处理的最终结果是对页面图像的描绘，它是一系列基本图像操作作用在媒体上的结果。

9.6 结构边界的作用

序言(PROLOGUE)结构元素以及先前的 TOKENSEQUENCE 结构元素累积的作用受两类结构边界的约束，它们是 BLOCK 边界和序言结构元素的作用域(Scope)。

虽然 DICTIONARY GENERATOR 所初始化的上下文词典将影响同一 BLOCK 中的所有后继 TOKENSEQUENCE 结构元素，但是上下文词典的内容不能对处在 DICTIONARY GENERATOR 作用域之外的 TOKENSEQUENCE 结构元素起作用。另外，由 DICTIONARY GENERATOR 产生的上下文词典，除非已明确地在下属 BLOCK 的 COTEXT DECLARATION 中被引用，否则不作为下属 BLOCK 中 TOKENSEQUENCE 的解释上下文的一部分。

SETUP PROSEDURE 结构元素只对其作用域内的上下文栈的状态起作用。

BLOCK 边界对解释上下文中的所有成分都起作用。随着 TOKENSEQUENCE 结构元素的处理，解释上下文中各成分的状态在不断积累。一个 BLOCK 的所有内容在处理完毕后，解释上下文的状态恢复到刚开始处理此 BLOCK 时的状态。这一状态复位对解释上下文的所有成分都起作用。

解释上下文中的可用资源集，以及在 CONTEXT DECLARATION 中说明的上下文词典集，其作用域局限于 BLOCK 的边界。序言结构元素的作用域永不可超出说明它的 BLOCK。

由于作用域和状态复位的性质与 BLOCK 边界有关，所以一个 BLOCK 中不可能存在任何信息(结构或内容)可以对同级或上级结构元素起作用。然而，一个 BLOCK 的信息可以对其下属的解释上下文起作用。

注：BLOCK 边界性质的一个结果是，SPDL 文件的每个 PAGE 与其他 PAGE 是完全独立无关的。

9.7 字型资源和字型对象

与 ISO/IEC 9541-1 一致的字型资源可以用 ASN.1 数据结构或 SGML 数据结构来表示，定义见 ISO/IEC 9541-2 和 ISO/IEC 9541-3。每个字型资源中的字形(glyphs)由结构化名来标识，它是字母名(GNAME)特性的值。字型资源中的字形若与 ISO/IEC 9541-1 不一致，则可以由基数或简单名来标识。

在 SPDL 中字型资源是用字型词典来表示的，其中字形由标识符值来标识。这些标识值与字形在字型资源中标识方式的关系如下：

——若字形是由一个代表 ISO/IEC 9541-1:1991 中 6.2 所定义的 ISO 10036 字形名的结构化名所标识，这一标识符为名字“afii:nnnn”，其中 nnnn 是一字符串，它是开头不为零的一串十六进制数码(“0”..“9”,“a”..“f”)，串的内容表示了该标识字形的“ISO 10036 glyph local _name”的值。

——若字形不是由上述的结构化名所标识，标识符是执行(string ConvertToIdentifier)的结果，此处 string 是公用标识符的规范串格式，语义上等同于 ISO 9070 定义的结构化名的值。

——若字形是用简单名定义的，标识符值的值是该简单名的名字(Name)。

字型对象用字型词典的方式进行定义，见第 16 章。

10 文件内容处理子模型

SPDL 文件的内容由文件中 TOKENSEQUENCES 的内容组成。与自然定义的结构表示法不同，这些 TOKENSEQUENCES 可用为文件内容处理器解释的一种过程式语言来表达。文件内容及相应的解释上下文在文件结构处理过程中采用第 9 章中所述的方法进行标识，然后把这些数据提供给文件内容处理器。

文件内容的处理包括对 SPDL 文件所描绘的一组页面图像的成像处理。精确地定义这一复杂行为需要一个子模型。

本章定义了虚拟机，它是整个内容解释过程的模型，也定义了成像模型和坐标系统。第 15 章给出了彩色模型，第 16、17 和 18 三章定义了字符文本和字型、光栅图形和几何图形的子模型。第 19 章定义了

裁剪区域的特性，第 20 章定义了用于图案的逻辑油墨的子模型，第 22，23 和 24 三章定义了用于光栅图形数据的压缩和传输、图案，以及出错处理等子模型。

10.1 虚拟机 Virtual Machine

文件内容的处理采用一种虚拟机模型。虚拟机顺序地解释一个个标记，以状态方式积累信息，并完成成像操作，其结果积累在页面图像上。以下一些条给出虚拟机的定义。

注：虚拟机模型仅用于定义文件内容的语义，它不去支配文件内容处理器的具体实现，也不限制文件内容语义的具体实现方法。

文件内容处理器的虚拟机模型包括以下一些成分：

——解释器；

——标记序列；

——操作数栈；

——词典集；

——上下文栈；

——资源集；

——状态变量集；

——隐含的图形状态栈；

——页面图像。

图 10-1 给出了虚拟机模型各成分的示意以及定义它们的章条。

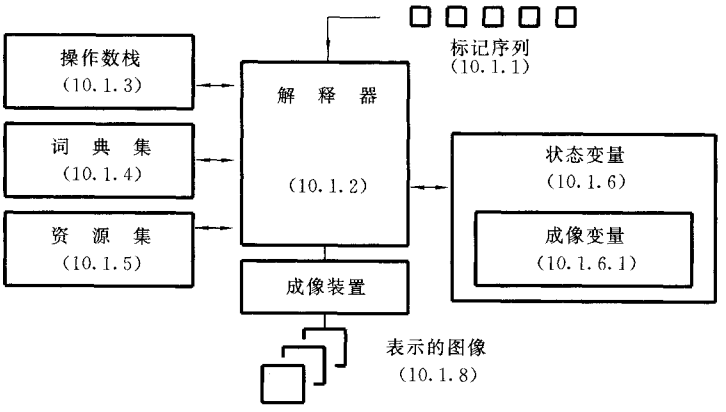


图 10-1 虚拟机

对虚拟机各部分的定义依赖于 4 个概念：对象、状态、操作和序列(seqaence)。

对象是离散的数据单位，它可以由虚拟机进行操作。每个对象有一个类型和一个值，本标准对对象类型的定义见第 11 章，例如：

——Integer(整型)

——Vecter(向量型)

——Boolean(布尔型)

虚拟机状态是以下几种对象的组合：所有词典的集合，上下文栈，资源的集合，隐含的图形状态栈，操作数栈，以及状态变量的集合。这些状态可以通过以下方法来改变：

——文件结构处理器；

——虚拟机所执行的操作。

操作指虚拟机执行的动作，包括：

——对象处理；

——虚拟机状态处理；

——成像动作。

操作受以下一些因素的影响：

——用作操作数的对象；

——操作之前的虚拟机状态。

操作不受操作前页面图像的影响。

序列是虚拟机所必须的，序列中所执行的操作对虚拟机状态和页面图像产生一个累积作用。与文件结构有关的累积作用表现如下：

——在单个 TOKENSEQUENCE 中，其标记按出现次序进行解释；

——在一个高层结构内（即 DOCUMENT，PAGESET，PAGE 或 PICTURE），其中文件内容以出现次序处理。

序列文件内容处理与文件结构的相互影响，以及处理结果对虚拟机状态产生的影响已在第 9 章中给出。以下几条描述虚拟机的各组成部分。

10.1.1 标记序列

每个 TOKENSEQUENCE 包含的内容是以内容交换格式给出的数据，见第 25 章。这些数据由一序列标记（token）组成，每个标记或者是字面值（literal）或者是可执行的（executable）。

可执行是与对象联编的名字以内容交换格式给出的一种表示方法。

字面值是对象以内容交换格式给出的表示方法，并不是所有的对象类型都有字面表示形式。

TOKENSEQUENCE 处理的虚拟机模型是：对每个 TOKENSEQUENCE，文件内容处理器从结构处理器那里接受一执行上下文，内容处理器解释此 TOKENSEQUENCE，它可能会修改执行上下文，最后返回给结构处理器一个新的解释上下文。

10.1.2 解释器

解释器顺序解释标记（token）序列中的每一个标记。

对一个字面标记的解释就把该字面标记所表示的对象压入操作数栈。

对一个可执行标记的解释就是对与该标记所表示的名字联编的对象进行解释。

对一个操作符类型对象的解释就使虚拟机去完成这一特定操作符所指定的动作。本标准所定义的操作符集详见第 13 至 24 章。

注：操作符集综述见附录 A。

解释一个操作符引起的动作可能包括：

——从操作数栈中取出对象（作为操作数）；

——对这些对象进行计算或其他处理；

——产生新的对象；

——将新对象（作为结果）存入操作数栈；

——查询或改变词典中的值；

——查询或改变状态变量的值；

——引用解释上下文中的资源；

——执行成像动作。

每个操作符所执行的动作随操作符一起定义在第 13 至 24 章中。

10.1.3 操作数栈 Operand stack

操作数栈是一个可以存放任意数目对象的栈，它是一个异构型栈，其中可存放任意类型组合的对象。

注：本标准中，栈、栈顶、压入、弹出等术语的用法与一般计算机科学中的用法一致。

操作符需要显式操作数对象，它可从操作数栈中弹出而获得。返回显式结果对象，且被压入操作数栈。

操作数栈中的对象是构成虚拟机状态的一部分。

10.1.4 词典集 Set of Dictionaries

词典构成了虚拟机状态的一部分，词典集指适用于特定 TOKENSEQUENCE 的所有词典的集合（其中的一部分可能在上下文栈中）。

词典集和上下文栈是用词典(Dictionaries)这一术语定义的。词典提供了一个存放值的机制，其中的值不象操作数栈那样顺序访问。

词典是对象有序对的集合。每个有序对包括一个标识符类型的对象和另一个任何类型的对象。每一个有序对中的第一个元素称为名字元素，第二个元素称为值元素。在一个词典中任何两个有序对不会具有相同名字的元素。若存在一个名字元素为 name 的有序对在某一词典中，则说对象 name 在该词典中，或该词典中包含 name。

若 name 在词典中，则对此词典而言 name 的值即是名字元素为 name 的有序对中的值元素。若从上下文明显可知词典包含 name，则上述说法可简称为 name 的值。词典不是对象类型，不能被压入操作数栈（然而，词典引用(DictionaryReference)是一个对象类型，见第 11 章）。字典可由文件内容中的操作符来建立，上下文词典是由文件结构中的 DICTIONARY GENERATORS 建立的。

10.1.4.1 词典-显式引用 Dictionary-Explicit Referencing

词典集是虚拟机的一个组成部分，它包括由 DICTIONARY GENERATORS 建立的词典和处理文件内容时操作符 MakeDictionary 所建立的词典。

非上下文词典可以通过 DictionaryReference 被显式引用，构成默认上下文栈的词典也可以通过 DictionaryReference 被显式引用。

DictionaryReference 允许对象与一个标识符联系起来，并用 put 操作符把对象存放到被引用的词典中，然后可以使用该标识符和对该词典的引用通过 Get 操作符在引用词典中检索。上下文词典中的对象可以用 Get 操作符去检索，但是上下文词典中的内容不可用 Put 操作符去修改。DictionaryReference 也可以通过使用 PushContextStack 操作符来增加一个词典到上下文栈中。

词典集中的词典以及每个词典的内容构成了虚拟机状态的一部分，词典自产生以后的任何时刻都可以被引用，直到它从状态中被除去。

10.1.4.2 词典-隐式引用 Dictionaries-Implicit Referencing

上下文栈是虚拟机的一部分，上下文栈中的词典是词典集的一个子集，并构成了虚拟机状态的一部分，上下文栈中的词典构成解释上下文的一部分。

这些词典可以用上下文栈引用操作符 GetValue 来隐式地引用，这一操作符允许在上下文栈的词典中检索与标识符相对应的对象。

上下文栈是上下文词典的一个有序表。当引用上下文栈去检索一个标识符时，它是以相反的次序依次检索的，最末一个词典最先检查。若找到，则返回其值；否则检索下一个词典，依此类推。

伪类型 Name 是所有其内容可直接编码的标识符的子集（见第 11 章）。内容中遇到的可执行名使文件内容处理器对来自上下文栈的一个隐式标识符进行检索，并对检索到的对象隐式解释。

不可能隐式地引用一个不在上下文栈中的词典。

10.1.4.3 默认上下文栈 Default Context Stack

任何给定的 TOKENSEQUENCE 的解释上下文包括一个上下文栈，它包含一个默认上下文栈中的词典表，后跟任何由 CONTEXT DECLARATION，DICTIONARY GENERATORS 以及先前的 TOKENSEQUENCE 等附加在其后的词典。默认的上下文栈包括以下一些词典，按其在上下文栈中概念上的次序给出：

——系统词典(SystemDict)——这一词典包含每一个 SPDL 操作符的名字-值对。对每个名字-值对，名字是在本标准附录 A 中定义的操作符名，值是一个操作符类型的对象，它实现操作符的功能。词典中还包含一个名字-值对，其名字为 SystemDict，值是一个引用 SystemDict 的词典引用(Dictio-

naryReference)类型对象。系统词典中最后一个必须的名字-值对是名字为 UserDict，值为词典引用类型的对象，它引用缺省上下文栈的用户词典(UserDict)(见下面所述)。系统词典(SystemDict)总是上下文栈中的第一个词典，并且不可以为 SPDL 文件所修改。

——用户词典(User Dict)——其容量至少可以存放 100 个新的名字-值对，UserDict 总是默认上下文栈的最末一个词典，并且可以被 SPDL 文件所修改。

默认上下文栈中的词典不可以从状态中除去，它总是作为每一个解释上下文的上下文栈的一部分，可修改的默认上下文栈词典的内容受状态复位正常语义的控制。

10.1.5 资源集 Set of Resources

资源集是解释上下文的一部分，它在文件结构处理过程中被指定。资源集是适用于一特定 TOKENSEQUENCE 内容的所有资源的集合。

资源可以通过查找操作符在内容中引用，无论是使用通用的 FindResources 操作符还是给定资源类型的特定操作符均可。这类操作符把资源名作为操作数，这一名字是在 RESOURCE REFERENCE 中与资源联编的 INTERNAL NAME。这类操作符返回一个相应的资源，把它作为一个对象压入操作数栈。在结构层定义资源的方法见第 8 章。各类资源的值的语义见第 15 章、16 章、20 章、22 章和 23 章。操作符 FindResource 的定义在第 13 章中给出。

10.1.6 状态变量 StateVariables

虚拟机的成分之一是状态变量的集合。状态变量的值是对象，这些对象充当许多操作符的隐含操作数。状态变量是全局的，总可以在内容中被引用。状态变量的值构成了虚拟机状态的一部分。状态变量及其初值的定义见第 12 章。

10.1.6.1 成像变量 ImagingVarables

用于表示成像装置状态的一组值是状态变量的一个子集，称为成像变量。它们充当大多数成像操作符的隐含参数。例如 CurrentTransformation 和 CurrentColour。

10.1.7 隐含的图形状态栈 Implicit Graphics State Stack

虚拟机的成分之一是图形状态栈。称这个栈为隐含的是因为和操作数栈不同，它不存在任何操作符可以显式地操作图形状态栈的内容。图形状态栈中的对象是因实现而异的数据对象，它们表示对象在建立入栈时成像变量集的状态。与此栈相关的操作符的语义见第 13 章。

10.1.8 页面图像 PageImage

描绘文本、几何图形或光栅图形的内容时解释器将执行成像操作。这些操作使页面图像积累起来。

在处理 PAGE 内容的开始，页面图像是空的，即页面上没有图像元素(见 10.2.4)。文件内容处理器的责任是用它的解释器解释执行成像操作的操作符从而建立起页面图像，文件结构处理器的责任是把完整的页面图像与表示媒体的实例相联系，将空页面送到表示媒体实例上，媒体不起任何作用。

10.2 成像模型

成像动作将按照成像模型去执行。

这个成像模型是一个二维图形模型，与设备无关，以满足高质量印刷和出版的需要。页面图像是通过顺序地在开始为空的页面上不断添加图像元素而逐步建立起来的。图像元素放置到页面上是通过掩模和裁剪区域将油墨涂到页面上来实现的，如图 10-2 所示。

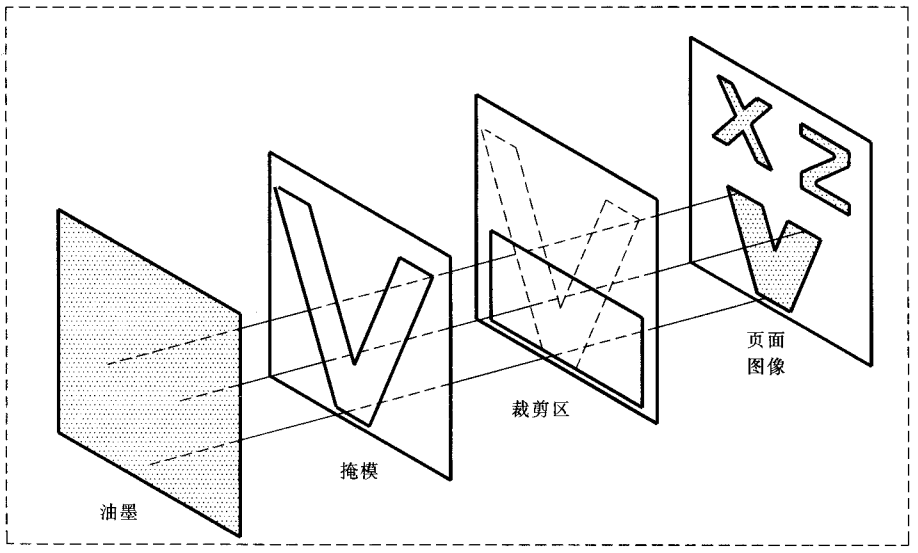


图 10-2 成像模型

成像模型包括 4 个组成部分：

——色彩

——掩模

——裁剪区域

——页面图像

每个成像操作符都包括这 4 个部分的具体应用：掩模(mask)作为成像操作符的一个操作数，色彩和裁剪区域是从成像变量中得到的两个当前值，页面图像保存在虚拟机一个单独的部分中。下面详细讨论各部分功能。

10.2.1 色彩 Colour

成像操作使表示设备通过逻辑油墨在当前所选彩色空间中用当前所选彩色进行描绘。术语油墨指由表示设备显示的颜色和可见纹理。油墨可以是任何颜色，包括黑、白和任意灰度。所有油墨逻辑上都是不透明的，它将遮盖先前画在页面上同一区域内的图像元素。

除了 ImageRasterElement 操作符这一例外，每个图像操作所描绘的颜色以及引用的彩色空间，分别取自 CurrentColour 和 CurrentColourSpace 两个图像状态变量。

10.2.2 掩模 Mask

掩模给出了图像元素的形状。掩模可以用几何轮廓线的内部来表示，如笔划的几何轮廓线边界，二维布尔值数组(位图 bitmap)，或者直接引用字型的字形(glyphs)。几何轮廓线可以描绘简单的凸形、凹形或者具有自交或分离边界的区域。所有这些种类的掩模都可以用成像操作符来描述。

掩模描述是作为操作数提供给成像操作符的。在几何图形成像操作符和 MarkBitMap 中，操作数是显式的；在 ImageRasterElement 和字符文本成像操作符中，操作数是隐式的。

10.2.3 裁剪区域 ClippingRegion

裁剪区域是页面图像上可以涂油墨的区域。落在裁剪区域内的油墨可以影响页面图像，落在裁剪区域外的油墨不起作用。于是图像元素的形状可以被裁剪区域所裁剪。

裁剪区域的边界是用与描述掩模形状相同的几何方法来描述的。初始裁剪区域是一个包含页面上整个可成像区域的矩形。然而它可以通过与另一个几何区域求交的办法来临时受限于某一区域。

每个成像操作符所使用的裁剪区域均取自 CurrentClipRegion 成像变量。

10.2.4 页面图像 PageImage

页面图像是通过顺序地在开始为空的页面上添加图像元素逐步建立起来的。一个图像元素是单个

成像动作的结果：如一个填充的几何区域，一个描绘(stoke)的几何边界，一个成像的光栅图形或一个简单的字形。

当前页面图像作为虚拟机的一个单独部分来保存。所有成像操作均与它们执行时页面图像的状态无关。

注：页面图像的复制区域(assured reproduction area)是指图像媒体实例中可以作标记的区域。关于引用坐标系RCS中这一区域的位置和大小因媒体的不同和SPDL实现的不同而改变；然而，SPDL提供了允许在内容解释过程中确定这一位置和大小的状态变量(见12.19)。复制区域对任何给定的媒体总是取其默认的CurrentClipRegion，并且总是在媒体上。

10.3 坐标系统

10.3.1 引用坐标系 Reference Coordinate System

引用坐标系(RCS)提供了对媒体上位置的明确描述。

引用坐标系是迪卡尔坐标系，两坐标轴的单位都是毫米(millineters)。RCS在媒体上的位置和取向是媒体特性而不是描绘到媒体上的图像特性。一可视区域的矩形媒体，常常保持媒体视见区的取向如下：

——原点在媒体的左下角；

——X轴沿底边以向右为正方向；

——Y轴沿左边以向上为正方向。

对于象纸张一类没有可自然辨别的视见区的媒体，引用视见区常常将长边作为垂直方向(通常称为肖像画取向——portrait orientation)。媒体上文件所完成的页面图像常常需用另一种方向去看(例如将长边作为水平方向，通常称之为风景画取向——landscape orientation)；注意点是媒体本身没有自然视见区。

10.3.2 用户坐标系 User Coordinate Systems

用户坐标系(UCS)提供了一种描述页面图像位置的灵活方法。

成像操作所使用的位置均是以UCS加上用户坐标系到引用坐标系的映射来给出的。映射完成两个功能：

——它把RCS中的物理度量单位与UCS中设有尺寸的单位相联系；

——它提供用户方便地使用用户坐标系来描述位置和形状。这样的UCS在向页面添加图像元素时不会去限制结果形状的大小和取向，因为映射是可以任意改变的。

这一映射由CurrentTransformation成像变量来给出，这个变量是虚拟机的一个状态变量。CurrentTransformation允许用户坐标系到引用坐标系的任意映射，格式如下：

$$x' = a \times x + c \times y + e$$

$$y' = b \times x + d \times y + f$$

式中a、b、c、d、e和f是以毫米为单位给出的实数，x、y是UCS中的坐标，x′y′是RCS中的坐标，这一变换表示UCS空间中描述的图像元素和加到页面(在RCS空间)中去的图像元素之间，可以经过任何变比、平移和旋转三种变换的任意组合变换。

注：任何映射，若

$$a \times d - b \times c = 0$$

则将是退化的和不可逆转的。

建立变换对象的操作可作为CurrentTransformation成像变量，这将在第14章中定义。

11 数据类型

11.1 对象类型

SPDL虚拟机所处理的对象是由相关的对象类型和值构成。这一章定义本标准中所使用的对象类型。

以内容交换格式(Content Interchange Format)进行文字描述的对象类型的子集在第 25 章中给出。

注：见第 25 章开头的对象类型列表，他们没有以内容交换格式给出文字描述。

11.1.1 布尔型 Boolean

布尔型对象具有两个值中的一个，这两个值为真(true)和假(false)。值 true 和 false 与布尔操作相关。

11.1.2 词典引用 Dictionary Reference

词典引用类型的对象是对词典的引用，词典的定义见 10.1.4。

11.1.3 标识符

标识符类型对象的值由一系列八进制表示的 GB 1988 字符代码构成。所有标识符的子集，其内容可以直接编码(见 11.3.5 中的伪类型 Name)，值与名字(Name)伪类型的规定不一致的标识符可以类似八位字节串进行编码，并可通过执行 ConvertToIdentifier 操作符转换成标识符。

11.1.4 整数

整数类型的对象即为一个整数。

11.1.5 标记 Mark

标记类型的对象具有唯一的值，此值可通过 Mark 操作符放入操作数栈。标记类型的对象可用来分隔操作数栈中的一系列变量。

11.1.6 空 Null

空类型对象具有唯一的值，它可通过 Null 操作符放入操作数栈。在用 MakeVecter 操作符建立向量(Vector)时，该对象可用来初始化向量元素。

11.1.7 八位字节串引用 OctetStringReference

八位字节串引用类型的对象是对一个八位字节串数据对象的引用。

11.1.8 操作符

在执行操作符类型的对象时，致使虚拟机去执行一特定动作或一系列动作。当遇到此类对象的名字时，或当 Execute 操作符的操作数是操作符类型或操作符类型对象词典中的标识符时，虚拟机就执行这类对象。SPDL 文件不能建立操作符类型的对象，但可以将标识符定义成此类对象。

11.1.9 路径引用 PathReference

路径引用类型的对象是对路径(Path)数据对象的引用。

11.1.10 实数

实数类型对象是其值在由 IEEE 854 定义的浮点数集合中的实数。

11.1.11 保存状态引用 SavedStateReference

保存状态引用类型的对象是对保存状态数据对象的引用。

11.1.12 流对象 StreamObject

流对象类型的对象提供了对数据源(DataSource)资源中的数据和从过滤器(filter-见第 22 章)输出数据的内容级(content-level)的访问。

11.1.13 向量引用 VectorReference

向量引用类型的对象是对向量数据对象的引用。

11.2 属性 Attributes

某些 SPDL 对象具有与它们相关的一种或多种属性。两种可能的属性是：可执行属性，它决定了该对象是被当成数据压入操作栈还是被当成可执行对象；存取属性，它决定了该对象的值是否可以被 SPDL 操作符显式地读或写。

11.2.1 可执行属性 Executable Attribute

可执行属性具有布尔值 true 和 false，它们仅适用于标识符、操作符或向量引用等类型的对象。操作

符类型对象的可执行属性总是 true，标识符和向量引用类型的对象，它的可执行属性可通过执行 ConvertToExecutable 操作符而显式地置成 true。

11.2.1.1 标识符

通过执行操作符 ConvertToIdentifier 所建立的标识符，在其刚建立时它的可执行属性为 false，在程序中遇到可执行名字时可执行属性为 true。程序中的字面名(literal name)的可执行属性为 false。

11.2.1.2 向量引用

由 MakeVector 操作符所建立的向量引用对象的可执行属性是 false，执行操作符 ConvertToExecutable，它们的可执行属性可改变成 true。可执行属性为 true 的向量引用对象也可被伪类型过程(Pseudo-type Procedure)所认识。

属性作用于向量引用，而不是被引用的向量。也就是说，两个向量引用，可能有不同的可执行属性，但引用相同向量，因而它们在程序中可能被区别对待。

11.2.2 存取属性

存取属性只作用于词典引用、八位字节串引用、向量引用等类型的对象，下面是三种可能的存取属性：

——可读写(ReadWrite)

具有可读写存取属性值的引用对象，在程序中可显式地被操作符读出或写入。

——只读(ReadOnly)

具有只读存取属性值的引用对象，在程序中可以被显式地读取，但不能显式地写入。

——只可执行(ExecuteOnly)

具有只可执行属性值的引用对象，在程序中不能被显式地读出或写入，但可以被执行。

在程序中建立的类型为词典引用、八位字节串引用或向量引用等新的对象，除非另外指定，否则都具有可读写的存取属性。

对于八位字节串引用和向量引用类型的对象，属性是作用于引用而不是作用于被引用的对象(见 11.2.1.2 中对可执行属性的讨论)。对于词典引用类型的对象，属性则作用于被引用的词典的值。存取属性可通过操作符 MakeReadOnly 和 MakeExecuteOnly 降级。

11.3 伪类型 Pseudo-types

本条定义了一些伪类型，它们或者表示对象类型的严格子集或者对象类型的组合，本标准中使用伪类型是为了在定义操作符的语义时方便。

11.3.1 任意类型 Any

伪类型 Any 表示在 11.1 中定义的所有对象类型的组合。

11.3.2 基数 Cardinal

伪类型 Cardinal 表示整数对象类型的子集，且具有非负值，即它的值可以为零或正整数。

11.3.3 词典关键字 Dictionary Key

伪类型 DictionaryKey 表示对象类型标识符和伪类型基数的组合。

注：在 10.1.4 中论述了词典包含名/值的对，伪类型 DictionaryKey 定义了名—值对中名字对象的集合。

11.3.4 字型对象 Font Object

伪类型 FontObject 表示词典引用对象类型的子集，它所引用的词典包含了用特定字型进行字符文本描述所需的信息。

注：在文件内容中，FontObject 可使用 DefineFont，PutWMode，ScaleFont 和 TransformFont 等操作符来构造，或使用 Find Font 和 Find Resource 操作符从文件结构可用资源集中选择。FontObject 可以通过赋值成像变量 CurrentFont 来产生字符文本。

11.3.5 字形串 Glyph String

伪类型 GlyphString 是 OctetStringReference 对象类型的子集，它可作为 ShowString 操作符的操

作数。

注：GlyphString 的主要用途是有效地表示字符文本。GlyphString 的元素可解释为一列字形索引，通过引用与 Currentfont 相关的字形索引映象来表示字形。

11.3.6 名字 Name

伪类型 Name 表示其内容可直接编码的 Identifier 对象类型的子集。Name 的第一个八进制值必须是拉丁字符集中的 GB 1988 编码字符，如：a,b,c,…,z;A,B,C,…,Z;或句号。其后一系列的八进值可以是拉丁字形、十进制数字 0,1,2,…,9、下划线_、冒号:或句号...。根据 SPDL 文件格式，冒号字符只有当名字被用作字形标识(GlyID)时才能出现在名字中(见第 16 章)。

在程序中，Name 类型的对象可以是可执行的标记或字面标记(literal token)，它们的处理见 10.1.2。

11.3.7 数字 Number

伪类型 Number 表示对象类型 Real 和 Integer 的组合。

11.3.8 对象引用 Object Reference

伪类型 ObjectReference 是表示对象类型 DictionaryReference、PathReference、OctetStringReference 和 VectorReference 的组合。

11.3.9 过程 Procedure

伪类型 Procedure 是对象类型 VectorReference 的子集，它们的可执行属性是 true。

注：在程序中，过程可以被编码或通过对 VectorReference 使用操作符 ConvertToExecutable 来建立。在程序中当遇到一个可执行的过程时，过程就被解释；或遇到操作符 Execute 时，过程也将被解释。

11.3.10 变换 Transformation

伪类型 Transformation 是 VectorReference 的子集，被引用的向量以下面的格式给出：

[a b c d e f]

表示从一个迪卡尔坐标系到另一个坐标系的线性映射变换，具体见下面的论述。

所有的映射变换均可表示成下面的形式：

$x' = a \times x + c \times y + e$

$y' = b \times x + d \times y + f$

其中 a,b,c,d,e 和 f 均是数字值，x 和 y 是第一个迪卡尔坐标系中的坐标，x'y'是第二个迪卡尔坐标系中的坐标。

注：当 $a \times d - b \times c = 0$，任何映射变换将退化，且没有逆变换。

一组特定的系数 a,b,c,d,e,f 可以说是与一个特定的变换相对应，也可以说定义了一个特定的变换。系数是没有尺寸的，除非相关的变换是当前变换 CurrentTransformation，这时系数的尺寸是毫米。

上述等式也可以表示成矩阵的乘法。

$$[x'y'1] = [x \quad y \quad 1] \times \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix}$$

因此，若矩阵 M 为：

$$M = \begin{bmatrix} ab0 \\ cd0 \\ ef1 \end{bmatrix}$$

则可以说相对应的或所定义的变换$[T_M]$由上式给出。

11.4 数据对象 Data Objects

本条定义了可由虚拟机隐式处理的几种数据对象，但它们不是对象类型，不能被压入操作数栈。

注：——这里定义的数据对象只是虚拟机模型的一部分。与虚拟机模型一样，在本标准中，它们只是用来定义文件

内容语义的一种手法。它既不用来控制文件内容处理机的具体实现，也不用来限制在特定实现过程中文件内容语义的实现方法。

——因为数据对象不是正式的对象类型，所以它们在内容交换格式中没有字面表示。但是根据它们的类型，可以在文件中被引用或处理。

11.4.1 词典 Dictionary

词典数据对象在 10.1.4 中已作了论述。

注：与词典引用不同，词典不是对象的一种类型，也不能压入操作数栈。词典可在文件内容中由操作符建立，或在文件结构中由 DICTIONARY GENERTORS 建立。

11.4.2 八位字节串 OctetString

OctetString 数据对象的值由一系列的元素组成，每一元素均为基数(Cardinal)类型，其值在 0 到 255 之间。

11.4.3 路径 Path

路径数据对象的值由 0 个或多个路径段构成，每一路径段包括一个起点和 0 个或多个路径元素，路径元素是下面三者之一：

——线段

——圆或椭圆的一段弧

——Bezier 曲线

每一非空路径、路径段和路径元素都有一个起点和一个终点，这里的点是在引用坐标系 RCS 中的位置，并可由两个实数来分别表示 x，y 的坐标。空路径的定义在本节的稍后给出。

每一路径段中第一个路径元素的起点就是该路径段的起点，其余路径元素的起点都是前一路径元素的终点。

每一路径段均有一个终点，它也是路径段中最后一个路径元素的终点。如果路径段只包含一个起点，则它的终点将与起点一致。如果路径段有一个或多个路径元素，那么该路径段就称为有向的(directed)，其方向就定义成从其起点到终点。

每一路径均有一个起点，它也是第一个路径段的起点。每一路径有一终点，它也是最后一个路径段的终点。只有一点例外，即一个没有任何路径段的路径被称为是空路径，它没有起点和终点。

注：路径主要用来构造几何图形成像元素和裁剪区域。

11.4.4 保存状态 SavedState

SavedState 数据对象的值由所有必要的信息构成，这些信息可以通过 RestoreState 操作符将虚拟机恢复到由 SavedState 数据对象建立时的状态。SavedState 数据对象只能由 SaveState 操作符建立。

11.4.5 向量 Vector

向量数据对象的值由一系列称为元素的对象构成，序列中的每一个对象与一个索引相对应，每一个元素的索引是一整数，其值从 0 到 n－1，其中 n 是序列中元素的个数。索引表示了元素在序列中的位置，向量中的元素并不要求都是同一对象类型。

注：一旦向量建立后，它将不能扩展。

12 状态变量

虚拟机的一个组成部分是一组状态变量。状态变量的值构成了虚拟机状态的一部分，状态变量是全局变量，可在程序的任何地方引用。成像变量集是状态变量的子集，其值在成像操作时使用。下面定义了每个状态变量、它们的类型以及初始值。

注：——许多操作符将状态变量的值作为显式操作数。这种用法在每一特定操作符的定义中给出。注意在下面的每一条中都引用了这些定义操作符及其语义的条文。

——某些状态变量能显式地设置，某些能显式地被查询，还有一些能由程序中的操作符来设置和查询。执行这些功能的操作符参考相应的条文。

12.1 当前黑色生成 CurrentBlackGeneration

CurreatBlackGeneration 成像变量有一个类型为 Proceder 的值，初始值依赖于具体实现。

注：CurrentBlackGeneration 指出了从 RGB 设备到 CMYK 设备转换过程中产生的 CMYK 设备黑色成分函数。处理 CurrentBlackGeneration 的操作符在第 18 章中定义。

12.2 当前裁剪区域 CurrentClipRegion

CurrentClipRegion 成像变量指出页面成像的区域，初始值是与介质上整个可成像区域相对应的区域。

注：CurrentClipRegion 是成像模型的一部分，它决定了介质上受成像操作影响的区域范围。它已在第 10 章中定义，在第 16 至 18 章中使用。处理 CurrentClipRegion 的操作符在 112 中定义。

12.3 当前色 CurrentColour

CurrentColour 成像变量的值指出当前彩色空间中的当前色/网纹，它在默认彩色空间(DeviceGrey)中的初始值是 0(100%黑色)。

注：在第 15～18 章，CurrentColour 指出了(在当前彩色空间)所有几何图形和文字成像操作中使用的彩色模型。每个彩色空间的 CurrentColour 初始值由 SetColourSpace 操作符来选择，处理 CurrentColour 的操作符在第 15 和 20 章中定义。

12.4 当前画图色 CurrentColourRendering

CurrentColourRendering 成像变量具有词典类型的值，它的初始值依赖于具体实现。

注：CurrentColourRendering 指出了 CIE 相关色到设备彩色转换函数的参数(类似一个画图色词典)，处理 CurrentColourRendering 的操作符在第 21 章中定义。

12.5 当前彩色空间 CurrentColourSpace

CurrentColourSpace 成像变量具有向量类型的值，它的初始值为单元素向量，该元素的值为〈ColourSpace/DeviceGrey：Identifier〉。

注：CurrentColourSpace 指出彩色映像的范围，以及执行此映像的方法。处理 CurrentColourSpace 的操作符在第 15 章中定义。

12.6 当前线型 CurrentDashPattern

CurrentDashPattern 成像变量的值由一个数值向量和一个数值型的偏移量构成。初始值是一个空向量，偏移量为 0。

注：CurrentDashPattern 指出了由 StrokePath 操作符沿路径成像的线的特点。处理此状态变量的操作符在第 18 章中定义。

12.7 当前字型 CurrentFont

CurrentFont 成像变量具有 FontObject 类型的值，它的初始值为空。如果在 CurrentFont 被赋值前进行存取 CurrentFont 的操作，将会引起 InvalidFont 异常。

注：CurrentFont 指出所有字符文本成像操作中所使用的 FontObject；见第 16 章，处理 CurrentFont 的操作符也在第 16 章中定义。

12.8 当前半色调 CurrentHalftone

CurrentHalftone 成像变量具有词典类型的值，它的初始值依赖于具体实现。

注：CurrentHalftone 指出半色调函数的参数(类似半色调词典)，处理 CurrentHalftone 的操作符在第 21 章中定义。

12.9 当前线接限止值 CurrentMitreLimit

CurrentMitreLimit 成像变量具有数值型的值，初始值是 10。

注：CurrentMitreLimit 为 StrokePath 操作符所使用，它决定了在何种情况下用平交线代替尖交线(见第 18 章)，处理此状态变量的操作符也在第 18 章中定义。

12.10 当前路径 CurrentPath

CurrentPath 成像变量是一路径，初始值是空路径，所有 CurrentPath 中的点是指在引用坐标系中的点。

注：CurrentPath 用于成像路径，由建立路径的操作符来处理。见第 18 章。

12.11 当前位置 CurrentPosition

CurrentPosition 成像变量的值是引用坐标系中的一个点，初始值是坐标(0,0)的点。

注：CurrentPosition 指出了在 RCS 坐标中的位置，它是字符文本序列成像的起始位置，或构造几何图形序列的下一个路径段的起点位置，处理该状态的操作符在第 16 和 18 章中定义。

12.12 当前画线调整 CurrentStrokeAdjust

CurrentStrokeAdjust 成像变量具有布尔类型的值，它的初始值依赖于具体实现。

注：当通过执行 StrokePath 操作符沿路径画线时，由于光栅效应，扫描转换算法会产生粗细不一的线条。粗细不一的问题在低分辨率设备(如软拷贝显示器)上特别明显。如果 CurrentStrokeAdjust 的值为 true，则线宽和画线的坐标会自动作必要的调整，以产生粗细一致的线条，其线宽与要求的宽度之差不超过半个像素宽度(线宽由 CurrentLineWidth 成像状态变量决定。处理此状态变量的操作符在第 18 章中定义。

12.13 当前线端 CurrentStrokeEnd

CurrentStrokeEnd 成像变量有类型 Cardinal 的值，初始值为 0。

注：CurrentStrokeEnd 指出由 StrokePath 操作符成像的路径端点的性质，处理此状态变量的操作符在第 18 章中定义。

12.14 当前线交 CurrentLineJoin

CurrentLineJoin 成像变量有类型为 Curdinal 的值，其初始值为 0。

注：CurrentStrokeJoin 指出由 StrokePath 操作符成像的不相切的两个路径元素相交的性质。处理此状态变量的操作符在第 18 章中定义。

12.15 当前线宽 CurrentStrokeWidth

CurrentStrokeWidth 成像变量有 Real 类型的值，其初始值为 1.0。

注：CurrentStrokeWidth 指出由 StrokePath 操作符成像的线的宽度(UCS 单位)，处理此状态变量的操作符在第 18 章中定义。

12.16 当前转换 CurrentTransfer

CurrentTransfer 成像变量的值包括四个过程对象，初始值依赖于具体实现。

注：CurrentTransfer 指出了进行 gamma 修正的函数(见 21.4)，处理 CurrentTransfer 的操作符在第 21 章中定义。

12.17 当前变换 CurrentTransformation

CurrentTransformation 成像变量有 Transformation 类型的值，其初始值为 UCS 坐标轴到 RCS 相应坐标轴的映像变换，它将 UCS 的 1 个单位映射到 RCS 的 1 个单位(1mm)。

注：CurrentTransformation 指出从 UCS 坐标到 RCS 坐标变换所应用的映射，此映射在 10.3.1 中定义，在第 16 至 18 章中使用。处理 CurrentTransformation 的操作符在第 14 章中定义。

12.18 当前底色去除 CurrentUnderColourRemoval

CurrentUnderColourRemoval 成像变量具有 Procedure 类型的值，初始值依赖于具体实现。

注：CurrentUnderColourRemoval 指出从 RGB 设备到 CMYK 设备转换时，用来调节 CMY 的值，以补偿黑色生成函数的作用，处理 CurrentUnderColourRemoval 的操作符在第 21 章中定义。

12.19 设备描述词典 DeviceDescriptionDictionary

DeviceDescription 状态变量具有 Dictionary 类型的值，本只读词典的内容至少包括下述一些项目，这些必需项目的初始值以及任何附加项目的名和值均依赖于具体实现。

注：DeviceDescription 的内容中涉及表示设备的某些当前状态。DeviceDescription 字典的当前值可通过执行第 13 章中定义的操作符 GetDeviceDescription 来获得。

12.19.1 颜色类 ColorClass

ColorClass 词典项目有 Name 类型的值，ColorClass 的合法值如下：

——单色(Monnochrome)——表示设备只有一种颜色；

——三级(Trilevel)——表示设备有两种颜色；

——全色(Fullcolor)——表示设备有三种以上的颜色。

12.19.2 当前分辨率 CurrentResolution

CurrentResolutionX 和 CurrentResolutionY 词典项都具有 Number 类型的值，表示供文件内容使用设备的当前分辨率，单位是 RCS 中每毫米的像素数。

12.19.3 当前介质大小 CurrentMediumSize

CurrentMediumSizeX 和 CurrentMediumSizeY 词典项具有数值类型的值，供文件内容使用的介质真正的大小，单位为 RCS 中沿 X 和 Y 轴的毫米数。

12.19.4 当前可成像区域 CurrentImageableRegion

CurrentImageMinX，CurrentImageMaxX，CurrentImageMinY 和 CurrentImageMaxX 均具有数值类型的值，可提供给文件内容使用的介质可允许的 RCS 中 X、Y 坐标的最大范围。

13 控制和计算操作符

本章指出了未直接包含在页面元素管理或成像中的操作符和相关的对象，这些包括：

——算术和逻辑操作符；

——处理操作数栈中值的操作符；

——处理词典、八位字节串和向量的操作符；

——流程控制和过程管理的操作符；

——保存和恢复虚拟机状态的操作符；

——对象类型转换和资源管理的操作符。

这些操作符和它们的语义的描述包括了对某些条件的描述，而这些条件在操作符的解释过程中可能引起内容异常的条件。内容异常和异常处理将在第 24 章中定义。除了这些特定操作符异常以外，还有一些几乎所有操作符的解释过程都可能发生的普通异常，这些普通异常和它们的语义在 24.6.2 中论述。

注：值得强调的是 SPDL 语法仅指出了得到的结果，而没有指出为了得到这种结果而必须采用的方法。为了叙述方便，本章中的定义将以特定的次序去要求特定的操作数栈管理和对象管理，而实际的实现可能会以不同方式进行，以达到给定操作符的预期效果。

13.1 算术和逻辑操作符

下述的算术运算符所执行的操作与 IEEE854 算术集 4 中的规定一致。

13.1.1 AbsoluteValue 绝对值

AbsoluteValue 取一个操作数；

〈X：Number〉

返回一个值：

〈Y：Number〉

其中 Y 是 X 的绝对值。如果操作数 X 是整数类型，则结果 Y 也是整数类型，否则 Y 是实数类型。

13.1.2 Add 加

Add 操作符取两个操作数：

〈Y：Number〉

〈X：Number〉

并返回一个结果：

〈Z：Number〉

其中 Z=X+Y。如果两个操作数为整数类型，则结果 Z 也是整数类型，否则 Z 是实数类型。

13.1.3 ArcTangent 反正切

ArcTangent 操作符取两个操作数：

〈X：Number〉

〈Y：Number〉

返回一个结果：

〈a：Real〉

其中 a 是 0～360 之间的角度值，其正切(tangent)值是 y/x，操作数的符号将决定结果的象限：

X	Y	结　果
+	+	正 X，正 Y 象限
+	−	正 X，负 Y 象限
−	+	负 X，正 Y 象限
−	−	负 X，负 Y 象限
任意	0	0，0
0	+	90，0
0	−	270，0
0	0	引起无定义结果(UndefinedResult)异常

13.1.4　And 与

And 操作符取两个操作数：

〈Y：Boolean 或 Integer〉

〈X：Boolean 或 Integer〉

返回一个结果：

〈Z：Boolean 或 Integer〉

两个操作数必须是同一类型，如果两者均是 Boolean 类型，则 Z 也是 Boolean 类型，并且 Z 的值是 X 和 Y 的布尔乘积；如果两个操作数均是 Integer，则 Z 也是 Integer 类型，且 Z 的值是 X 和 Y 的二进表示值的按位“与”。

13.1.5　Ceiling 取上限

Ceiling 操作符取一个操作数：

〈X：Number〉

返回一个结果：

〈Y：Number〉

其中 Y 是大于或等于 X 的最小整数。

13.1.6　Cosine　余弦

Cosine 操作符取一个操作数；

〈a：Number〉

返回一个结果：

〈c：Number〉

其中 a 是角度，c 是其余弦值。

13.1.7　Divide 除

Divide 操作符取两个操作数：

〈Y：Number〉

〈X：Number〉

返回一个结果：

〈Z：Number〉

其中 Z 是 X 被 Y 除的结果，如果 Y 是 0，则引起无定义结果(UndefinedResult)异常。

13.1.8　Equal 等于

Equal 操作符取两个操作数：

〈Y：Any〉

〈X:Any〉

返回一个结果：

〈Z:Any〉

其中 X 和 Y 是同一类型且其值相等，则 Z 是 true，否则 Z 为 false。但下面列举的情况例外：

如果 X 和 Y 的类型都是 Number，则它们一个为 Integer，另一个为 Real 时，也可以进行比较。

如果 X 和 Y 的类型均为 ObjectReference，则它们引用同一对象时，就认为是相等的。

类型为 Identifier 和 OctetStringReference 的对象可以进行比较，在这种情况下，它们都被看成是一串八进字符。

13.1.9 Exponentiate 指数

Exponentiate 取两个操作数：

〈Y:Number〉

〈X:Number〉

返回一个结果：

〈Z:Number〉

其中 Z 是 X 的 Y 次方幂。如果两个操作数都是整型，则结果 Z 也是整型，否则 Z 为实型。如果 X 为负数，且 Y 是分数，则将引起无定义结果(UndefinedResult)异常。

13.1.10 False 逻辑假

False 操作符无操作数，返回一个结果：

〈false:Boolean〉

它是一个值为 false 的布尔对象。

13.1.11 Floor 取下限

Floor 操作符取一个操作数：

〈X:Number〉

返回一个结果：

〈Y:Number〉

其中 Y 是小于或等于 X 的最大整数。

13.1.12 GreaterOrEqual 大于或等于

GreaterOrEqual 操作符取两个操作数：

〈Y:Number〉

〈X:Number〉

返回一个结果：

〈Z:Boolean〉

其中 Z 的结果为：

{X Y LessThan Not}

13.1.13 GreaterThan 大于

GreaterThan 操作符取两个操作数：

〈Y:Number〉

〈X:Number〉

返回一个结果：

〈Z:Number〉

其中当 X>Y 时，Z 的值为 true，否则 Z 的值为 false。

13.1.14 IntegerDivide 整数除法

IntegerDivide 操作符取两个操作数：

〈Y:Integer〉

〈X:Integer〉

返回一个结果：

〈Z:Integer〉

其中结果 Z 为：

{X Y Divide Truncate}

如果 Y 为零，将引起 UndefinedResult 异常。

13.1.15 LessOrEqual 小于或等于

LessOrEqual 操作符取两个操作数：

〈Y:Number〉

〈X:Number〉

返回一个结果：

〈Z:Boolean〉

其中结果 Z 为：

{X Y GreaterThan Not}

13.1.16 LessThan 小于

LessThan 操作符取两个操作数：

〈Y:Number〉

〈X:Number〉

返回一个结果：

〈Z:Boolean〉

其中，如果 X＜Y，则 Z 的值为 True，否则 Z 为 false。

13.1.17 Logarithm 对数

Logarithm 操作符取一个操作数：

〈n:Number〉

返回一个结果：

〈log:Real〉

其中 log 是 n 的常用对数(以 10 为底)

13.1.18 Logicalshift 逻辑移位

LogicalShift 操作符取两个操作数：

〈S:Integer〉

〈m:Integer〉

返回一个结果：

〈n:Integer〉

其中 m 是二进制表示且用来移位的值，S 指出移位的位数和移位的方向。n 是移位的结果。如果 S 是正的，则 m 左移 S 位；如果 S 是负的，则 m 右移。移入位的值为零。

13.1.19 Multiply 乘

Multiply 操作符取两个操作数：

〈Y:Number〉

〈X:Number〉

返回一个结果：

〈Z:Boolean〉

其中 Z=X×Y。如果两个操作数均为 Integer 类型，则 Z 是 Integer 类型，否则 Z 为 Real 类型。

13.1.20 NaturalLogarithm 自然对数

NaturalLogarithm 操作符取一个操作数：

〈n：Number〉

返回一个结果：

〈In：Boolean〉

其中 In 是 n 的自然对数(以 e 为底)。

13.1.21 Negate 取负

Nagate 操作符取一个操作数：

〈X：Number〉

返回一个结果：

〈Y：Boolean〉

其中 Y=－X。如果操作数 X 是 Integer 类型，则 Y 也是 Integer 类型，否则 Z 为 Real。

13.1.22 Not 取反

Not 操作符取一个操作数：

〈X：Boolean 或 Integer〉

返回一个结果：

〈Z：Boolean 或 Integer〉

如果操作数是 Boolean 类型，则 Z 也是 Boolean 类型，且当 X 为 false 时，Z 的值为 true，否则 Z 为 false。如果操作数是 Integer 类型，则 Z 也是 Integer 类型，且 Z 的值是 X 的二进制表示的补码。

13.1.23 NotEqual 不等于

NotEqual 操作符取两个操作数：

〈Y：Any〉

〈X：Any〉

返回一个结果：

〈Z：Boolean〉

其中结果 Z 为：

{X Y Equal Not}

13.1.24 Null 空

Null 操作符无操作数，返回一个结果：

〈null：Null〉

它是 Null 类型的对象。

13.1.25 Or 或

OR 操作符取两个操作数：

〈Y：Boolean 或 Omteger〉

〈X：Boolean 或 Integer〉

返回一个结果：

〈Z：Boolean 或 Integer〉

两个操作数必须是同一类型。如果两个操作数都是 Boolean 类型，则 Z 也是 Boolean 类型，且 Z 的值是 X 和 Y 值的布尔联合(union)。如果两个操作数都是 Integer 类型，则 Z 也是 Integer 类型，且 Z 的值是二进制表示的 X 和 Y 值的按位“或”。

13.1.26 Rand 随机数

Rand 操作符无操作数，返回一个结果：

〈rand：Cardinal〉

其中返回值是一个 0 到 231-1 之间的伪随机数。

13.1.27 RandSetState 置随机数状态

RandSetState 操作符取一个操作数：

〈seed:Integer〉

无结果返回，它的作用是初始化随机数发生器的状态。一个给定的种子值在特定的实现中将使 Rand 操作符以同样的顺序返回同一组数值。在不同的实现中，一给定的种子值可以产生不同数值。种子值在高层结构(DOCUMENT,PAGESET,PAGE 或 PICTURE)开始时被设置成一个随机值。

13.1.28 Remainder 余数

Remainder 操作符取两个操作数：

〈Y:Integer〉

〈X:Integer〉

返回一个结果：

〈Z:Integer〉

其中 Z 是 X 被 Y 除所得的余数。Z 的符号与 X 的符号相同。如果 Y 是零，将引起 UndefinedResult 异常。

{X Y Remainder}和{X Y X Y IntegerDivide Muliply Subtract}

是等价的。

13.1.29 Round 取整

Round 操作符取一个操作数：

〈X:Number〉

返回一个结果：

〈Y:Integer〉

其中 Y 是最接近 X 的整数。如果 X 与两个整数同样接近，则 Y 取两者中较大的数。

13.1.30 Sine 正弦

Sine 操作符取一个操作数：

〈a:Number〉

返回一个结果：

〈s:Real〉

其中 a 是角度值，s 是此角度的正弦值。

13.1.31 SquareRoot 平方根

SquareRoot 操作符取一个操作数：

〈X:Number〉

返回一个结果：

〈Y:Real〉

其中 X 必须为非负数，且 Y 为 X 的非负平方根。

13.1.32 Subtract 减

Subtract 操作符取两个操作数：

〈Y:Number〉

〈X:Number〉

返回一个结果：

〈Z:Number〉

其中 Z=X—Y。如果两个操作数均为 Integer 类型，则 Z 也是 Integer 类型，否则 Z 为 Real 类型。

13.1.33 true 逻辑真

true 操作符无操作数，返回一个结果：

〈true:Boolean〉

它是值为 true 的 Boolean 类型对象。

13.1.34 Truncate 截尾

Truncate 操作符取一个操作数:

〈X:Number〉

返回一个结果:

〈Y:Number〉

其中 X 大于或等于零,则 Y 是小于或等于 X 的最大整数。如果 X 小于零,则 Y 是大于或等于 X 的最小整数。

13.1.35 Xor 异或

Xor 操作符取两个操作数:

〈Y:Boolean 或 Integer〉

〈X:Boolean 或 Integer〉

返回一个结果:

〈Z:Boolean 或 Integer〉

两个操作数必须是同一类型。如果两个操作数都是 Boolean 类型,则 Z 也是 Boolean 类型,且当 X 和 Y 不同值时 Z 的值为真(true),否则为假(false)。如果两个操作数都是 Integer 类型,则 Z 也是 Integer 类型,且 Z 的值是二进制表示的 X 和 Y 值按位"异或"的结果。

13.2 管理操作数栈的操作符

13.2.1 ClearStack 清除栈

ClearStack 操作符无操作数,也不返回结果。ClearStack 操作符的作用是清除栈中的所有对象。

13.2.2 ClearToMark 清除栈到标记

ClearToMark 操作符无操作数,也不返回结果。ClearToMark 操作符的作用是清除栈中最高标记(Mark)以上的全部对象,也清除该标记。如果栈中没有 Mark 对象,将引起 UnmatchedMark 异常。

13.2.3 ConvertToExecutable 变为可执行

ConvertToExecutable 操作符取一个操作数:

〈X:VectorReference 或 Identifier〉

返回一个结果:

〈X:Procedure 或 Identifier〉

其中结果 X 是将操作数对象 X 的可执行属性置成 true(见 11.2.1 关于可执行属性的论述)。可执行属性为 true 的 VectorReference 通常被认为是伪类型过程。

13.2.4 ConvertToIdentifier 变为标识符

ConvertToIdentifier 操作符取一个操作数:

〈X:Cardinal 或 Name 或 OctetStringReference〉

返回一个结果:

〈Id:Identifier〉

如果操作数是 Cardinal 类型,则认为它是一个 ISO/IEC 9541 注册的标识符。ConvertToIdentifier 操作符返回形式为 afii:××××的标识符,其中××××是一串 1 至 8 个 GB 1988 编码字符,范围为 0~9,A~F;以非零数字开头,将操作数的值表示成一个无符号的十六进制数。

如果操作数是伪类型 Name,则它就简单地返回。

如果操作数是 OctetStringReference 类型,则被引用八位字节串的八位字节可被解释成一串 GB 1988编码的字符,并返回一个表示该序列的唯一标识符。

13.2.5 ConvertToInteger 变为整数

ConvertToInteger 操作符取一个操作数：

〈X:Number 或 OctetStringReference〉

返回一个结果：

〈int:Integer〉

如果操作数是 Integer 类型，则它简单地返回。如果操作数是 Real 类型，则它将由 Truncate 操作符截尾，并返回结果。

如果操作数是 OctetStringReference 类型，则被引用的八位字节串的每八位字节可被解释成一个 GB 1988 编码字符，这些字符可以解释成就象在纯文本内容中遇到的一个数，然后，得到的 Real 或 Integer 操作数就用上述方法进行解释。如果八位字节串的值不能正确地解释成在内容交换格式中所指定的 Real 或 Integer，则将引起 SyntaxError 异常。

13.2.6 ConvertToReal 变为实数

ConvertToReal 操作符取一个操作数：

〈X:Number 或 OctetStringReference〉

返回一个结果：

〈real:Real〉

如果操作数是 Integer 类型，则它被转换成实型并返回。

如果操作数是 Real 类型，则它简单返回。

如果操作数的类型是 OctetStringReference，则被引用的八位字节串中的每八位字节可被解释成一个 GB 1988 编码字符，这些字符就象在纯文本内容中遇到数字一样地进行解释；得到的 Real 或 Integer 再用上述方法进行解释。如果八位字节串的值不能正确地解释成在内容交换格式中所指定的 Real 或 Integer，则将引起 SyntexError 异常。

13.2.7 ConvertToString 变为字符串

ConvertToString 操作符取两个操作数：

〈S:OctetStringReference〉

〈X:Any〉

返回一个结果：

〈newS:OctetStringReference〉

ConvertToString 操作符用 X 的八位字节串表示来重写操作数 S；返回值 newS 是引用操作数 S 中重写的一个新的 OctetStringReference。

如果操作数 X 的类型是 Number，则 News 的内容将会是这样一些字符，若在文件内容中遇到，则表示 X 的数将被压入操作数栈。

如果操作数是 Identifier 类型，则 newS 的内容是一个八位字节串，它包含与 Identifier 的值相同的一串八位字节。

如果操作数是 Operator 类型，则在系统词典 SystemDict 中与该操作符相对应的标识符将按上述方法转换成八位字节串。

如果操作数是 OctetStringReference，则被引用的八位字节串将拷贝给 S。

如果操作数是 Boolean 类型，则 newS 的内容为八位字节串 true 或 false，根据 X 取适当的值。

如果操作数是其他类型，则 newS 的内容是八位字节串“—nost ringval—”。

13.2.8 Copy 拷贝

13.2.8.1 拷贝 n 个操作数栈的元素

Copy 操作符取 n+1 个操作数，其中 n 是这一系列操作数中最后一个值：

〈n:Cardinal〉

〈X_{n-1}:Any〉

…

〈X_1:Any〉

〈X_0:Any〉

返回 2n 个结果：

〈X_{n-1}:Any〉

…

〈X_1:Any〉

〈X_0:Any〉

〈X_{n-1}:Any〉

…

〈X_1:Any〉

〈X_0:Any〉

栈顶的操作数 n 被弹出，用以决定要拷贝对象的数目，然后剩下的 n 个操作数被弹出，并重新压回栈中，最后，同样的 n 个元素以同样的顺序再次压入栈中。

13.2.8.2 向量、词典、八位字节串的拷贝

Copy 操作符取两个操作数：

〈X_1:VectorReference,DictionaryReference 或 OctetStringReference〉

〈X_0:VectorReference,DictionaryReference 或 OctetStringReference〉

返回一个结果：

〈X_2:VectorReference,DictionaryReference 或 OctetStringReference〉

两个操作数必须为同一类型，把 X_0 引用对象中的所有元素拷贝到 X_1 引用对象中。X_1 引用的对象必须足够大，以容纳 X_0 引用对象的所有元素。

如果操作数是 DictionaryReference，则 X_1 引用的词典必须为空，并且返回值为 DictionaryReference 操作数 X_1。

如果操作数是 OctetStringReference 或 VectorReference，则 X_1 中没有被 X_0 中元素覆盖的元素依然保留，并且返回值是对新建立的八位字节串或向量的 OctetStringReference 或 VecterReference，它的长度与被 X_0 引用对象的长度一样，其内容与 X_0 引用对象一致。

13.2.9 Count 计数

Count 操作符无操作数，返回一个结果：

〈n:Cardinal〉

其中 n 是栈中对象的个数。

13.2.10 CountToMark 计数到标记

CountToMark 操作符无操作数，且返回一个结果：

〈n:Cardinal〉

其中 n 是栈中最高标记以上的对象的个数，如果栈中没有 Mark 对象，则将引起 UnmatchedMark 异常。

13.2.11 Dup 复制

Dup 操作符取一个操作数：

〈X:Any〉

返回 2 个结果：

〈X:Any〉

〈X:Any〉

除了栈顶元素被复制以外没有别的作用。

13.2.12 Exchange 互换

Exchange 操作符取两个操作数：

〈Y：Any〉

〈X：Any〉

返回 2 个结果：

〈X：Any〉

〈Y：Any〉

除了栈顶的两个元素交换以外没有其他作用。

13.2.13 Index 索引

Index 操作符取 n+1 个操作数，其中 n 是栈中最后一个操作数的值：

〈n：Cardinal〉

〈X_{n-1}：Any〉

〈X_{n-2}：Any〉

…

〈X_1：Any〉

〈X_0：Any〉

返回 n+1 个结果：

〈X_0：Any〉

〈X_{n-1}：Any〉

〈X_{n-2}：Any〉

…

〈X_1：Any〉

〈X_0：Any〉

结果是从栈中弹出 n，然后从栈顶向下数，找到第 n 个元素（把栈顶元素作为第 0 个元素），并将此元素拷贝到栈顶。

注：（〈X：Any〉〈0：Cardinal〉Index）与（X：Any＞Dup）是等价的。

13.2.14 mark 标记

Mark 操作符无操作数，且返回一个结果：

〈mark：Mark〉

Mark 操作符的唯一作用是将 Mark 对象压入栈。

13.2.15 Pop 弹出

Pop 操作符取一个操作数。

〈X：Any〉

且不返回结果，它唯一的作用是消除栈顶元素。

13.2.16 Roll 滚动

Roll 操作符取 n+2 个操作数，其中 n 是操作数表中倒数第二个操作数的值：

〈m：Inteter〉

〈n：Cardinal〉

〈X_{n-1}：Any〉

〈X_{n-2}：Any〉

…

〈X_1：Any〉

〈X_0：Any〉

返回结果为 n 个元素。首先弹出顶上的两个操作数 m 和 n，用以决定滚动对象的个数（n）、滚动的方向

(m 的符号)、以及滚动的位置个数(m)。然后弹出剩下的 n 个操作数,并对这 n 个对象执行环形滚动,结果压回栈中。

如果 m 的值为正,则滚动一个位置相当于移出对象序列的顶部元素,将其插入这一对象序列的底部,并将栈中相关的对象升高一个位置:

〈X_{n-2}:Any〉

…

〈X_1:Any〉

〈X_0:Any〉

〈X_{n-1}:Any〉

如果 m 的值为负,则滚动一个位置相当于移出对象序列底部的元素,将栈中元素降低一个位置,并将移出的元素压入栈顶:

〈X_0:Any〉

〈X_{n-1}:Any〉

〈X_{n-2}:Any〉

…

〈X_1:Any〉

13.2.17 Type 类型

Type 操作符取一个操作数:

〈X:Any〉

返回一个结果:

〈t:Identifier〉

其中 t 标识操作数 X 的类型,为下面若干类型之一:

Boolean	(布尔型)
Dictionary	(词典型)
Identifier	(标识符型)
Integer	(整型)
Mark	(标记型)
Null	(空类型)
OctetString	(八位字节串型)
Operator	(操作符型)
Path	(路径型)
Real	(实型)
SavedState	(保存状态)
Streamobject	(流对象)
Vector	(向量型)

13.3 处理词典、向量和八位字节串的操作符

13.3.1 AnchorSearch 锚定搜索

AnchorSearch 操作符取两个操作数:

〈search:OctetStringreference〉

〈string:OctetStringReference〉

返回两个或三个结果,这依赖于由 string 引用的八位字节串的初始子串是否与 search 引用的八位字节串相匹配。如果 string 引用的八位字节串的初始子串与搜索串相匹配,则 AnchorSearch 返回三个结果:

〈true:Boolean〉

〈match:OctetStringReference〉

〈post:OctetStringReference〉

其中 match 是对 string 引用的八位字节串中匹配的初始子串的引用，post 是对 string 引用的八位字节串中未匹配的剩余部分的引用。如果搜索串与 string 引用的八位字节串的初始子串不匹配，则 AnchorSearch 返回两个结果：

〈false:Boolean〉

〈string:OctetStringReference〉

其中 string 是对原先被搜索的八位字节串的引用。

注：不论在何种情况下，在执行 AnchorSearch 后栈顶元素是一个布尔值。如果搜索串与被搜索串的初始子串相匹配，则此布尔值为 true，否则为 false。

13.3.2 Capacity 容量

Capacity 操作符取单个操作数：

〈X:VectorReference,DictionaryReference 或 OctetStringReference〉

返回一个结果：

〈capacity:Cardinal〉

如果 X 的类型为 VectorReference 或 OctetStringReference，则返回对象是 X 引用对象的元素个数。

如果 X 的类型为 DictionaryReference，则返回对象是能够装入 X 所引用词典的有序对的最大总数。

13.3.3 ContextStack 上下文栈

ContextStack 操作符取单个操作数：

〈V:VectorReference〉

返回一个结果：

〈V:VectorReference〉

ContextStack 操作符枚举上下文栈中的词典，并将其存入由操作数 VectorReference V 引用的向量。假设 n 是上下文栈中词典个数，则返回的 VectorReference V 是原先向量的子向量的引用，它包含 n 个元素，每一个都是 DictionaryReference 类型。返回的 VectorReference V 引用的向量，其索引为 0 的元素，包含了对上下文栈中最底词典的引用；索引为 n−1 的元素，包含对上下文中最顶词典的引用。如果操作数 VectorReference V 引用的向量的长度小于上下文栈的深度，则引起 RangeCheck 异常。

13.3.4 Define 定义

Define 操作符取两个操作数：

〈new:Any〉

〈key:Dictionarykey〉

无返回结果。Define 操作符修改上下文栈中最顶部的词典，在词典中搜索所有名字元素与 key 相等的有序对，如果找到这样的有序对，则与该名字相对应的值将更新为 new。

如果没有找到名字与 key 相等的有序对，则把有序对(Key,new)加入词典。如果词典的容量不足以装入另一个有序对，则将引起 Dictionaryfull 异常。

注：词典中对应的名字在文件内容中可以是可执行的，见 10.2 的论述。

13.3.5 EntriesUsed 已用(词典)项数

EntriesUsed 操作符取一个操作数：

〈X:DictionaryReference〉

返回一个结果：

〈entries:Cardinal〉

这里的返回对象是在 X 引用的词典中当前定义的有序对的数目(相对于词典的最大容量)。

13.3.6 Get 取

〈key:Dictionarykey〉

〈X:VectorReference,DictionaryReference 或 OctetstringReference〉

返回一个结果：

〈Value:Any〉

如果 X 是 VectorReference 或 OctetStringReference，key 必须是 Cardinal 类型，返回对象是 X 引用对象中索引值为 key 的一个元素。如果 key 大于或等于对象的长度，或为负值，则引起 Rangecheck 异常。

如果 X 是一个 DictionaryReference，则返回对象是 X 引用词典中与 key 相对应的值。如果词典中不存在 key，则引起 Undefinedkey 异常。

13.3.7 GetCurrentDictionary 取当前词典

GetCurrentDictionary 操作符无操作数，返回一个结果：

〈DictRef:DictionaryReference〉

返回的词典引用是对当前上下文栈顶词典的引用。

13.3.8 GetInterval 取区间数据

GetInterval 操作符取三个操作数：

〈count:Cardinal〉

〈index:Cardinal〉

〈X:VectorReference 或 OctetStringReference〉

返回一个结果：

〈interval:VectorReference 或 OctecStringReference〉

返回的 interval 与 X 的类型相同，引用对象的长度为 Count，它的元素由 X 引用对象中索引为 index 的元素开始的连续 Count 个元素组成。

如果 index 在原向量或八位字节串中不是一个有效索引，或者 index+count 大于原向量或八位字节串的长度，则引起 RangeCheck 异常。

13.3.9 GetTest 取测试结果

GetTest 操作符取两个操作数：

〈key:DictionaryKey〉

〈DictTest:DictionaryReference〉

返回一个结果：

〈found:Boolean〉

如果 DictTest 引用的词典中存在其名字元素是 key 的有序对，则 GetTest 返回布尔值 true，否则返回 false。

13.3.10 GetValue 取值

GetValue 操作符取一个操作数：

〈key:DictionaryKey〉

返回一个结果：

〈Value:Any〉

返回对象由解释上下文过程中的上下文栈决定。从上下文栈顶的词典开始，顺序检测上下文栈中的词典，包含词典关键字 key 的最顶部的词典将用来决定与该词典相关的 key 的值，此值将返回操作数栈，如果上下文栈中的任何词典均不包含 key，则引起 Undefinedkey 异常。

13.3.11 GetValueTest 取值测试

GetValueTest 操作符取一个操作数：

〈key:DictionaryKey〉

返回对象由解释过程中的上下文栈决定。从上下文栈顶的词典开始，顺序检测上下文栈中的词典，如果发现有一个词典包含词典关键字 key，则 GetValueTest 操作符返回两个结果：

〈true:Boolean〉

〈DictRef:DictionaryReference〉

其中 DictRef 是对包含 key 的词典的引用。如果上下文栈中的词典均不包含 key，则 GetValueTest 返回一个结果：

〈false:Boolean〉

注：不论何种情况，GetValueTest 执行后的栈顶元素总是一个布尔值。如果上下文栈中的任何词典包含 key，则此布尔值为真，否则为假。

13.3.12 MakeDictionary 生成词典

MakeDictionary 操作符取一个操作数：

〈size:cardinal〉

返回一个结果：

〈DictRef:DictionaryReference〉

其中 DictRef 是一个新的空词典，其最大容量是 size 个有序对。

注：词典中的名字在文件内容中是一个可执行名，见 10.1.2 中的叙述。

13.3.13 MakeString 生成串

MakeString 操作符取一个操作数：

〈n:Cardinal〉

返回一个结果：

〈S:OctetStringReference〉

其中 n 是 S 引用的新创建的八位字节串元素的数目，MakeString 操作符用 0 来初始化新八位字节串中的每一个元素。

13.3.14 MakeVector 生成向量

MakeVector 取一个操作数：

〈n:Cardinal〉

返回一个结果

〈V:VectorReference〉

其中 n 是 V 引用的新创建向量中元素的数目，MakeVector 操作符用 Null 类型的对象来初始化新向量的每一个元素。

13.3.15 MakeandStoreDictionary 生成并存储词典

MakeandStoreDictionary 操作符取(n×2)+1 个操作数的列表，其中操作数栈中的第一操作数是 Mark 类型的对象，剩下的 n×2 个操作数表示 n 个名/值对：

〈$valuen_{n-1}$:Any〉

〈$valuen_{n-1}$:Any〉

〈$valuen_{n-2}$:Any〉

〈$name_{n-2}$:Any〉

…

〈$valuen_{0}$:Any〉

〈$name_{0}$:Any〉

〈mark:mark〉

返回一个结果：

〈DictRef:DictionaryReference〉

mark 以上的 n×2 个对象的数值被确定，一个容量为 n 的词典即已创建，然后栈中的 n 个名-值对被清除并存入词典，包括 Mark 在内的所有操作数均将被 MakeandstoreDictionary 操作符清除。如果栈中无 Mark 对象，则引起 Unmatched 异常。

13.3.16 MakeandStoreVector 创建并存储向量[1]

MakeandStoreVector 操作数取 n+1 个操作数的列表，栈中第一个操作数是 Mark 类型的对象：

〈$object_{n-1}$:Any〉

…

〈$object_1$:Any〉

〈$object_0$:Any〉

〈mark:mark〉

返回一个结果：

〈V:VectorReference〉

mark 以上 n 个对象的数值被清除并存入向量；最顶部的对象存入向量的第 n−1 个元素，最底部的对象存入向量的第 0 个元素。栈中包括 mark 在内的所向操作数均被 MakeandStoreVector 操作符清除。如果栈中无 mark 对象，则引起 UnmatchedMark 异常。

13.3.17 PopContextStack 弹出上下文栈

PopContextStack 操作符无操作数，也不返回结果。PopContexStack 操作符解释的结果是清除上下文栈中最顶部的词典。如果上下文栈是初始状态，即只包含默认的上下文栈，则引起 ContextStack-Underflow 异常。

13.3.18 PushContextStack 压入上下文栈

PushContextStack 操作符取一个操作数：

〈DictRef:DictionaryReference〉

不返回结果。PushContexStack 操作符解释的结果是将 DictRef 引用的词典压入上下文栈，也即将它作为当前上下文执行过程中第一个被查找的词典。如果该操作超越了具体实现时上下文栈的最大词典引用的数目，则引起 ContextStackOverflow 异常。

13.3.19 Put 元素替换

Put 操作符取三个操作数：

〈new:Any〉

〈key:Dictionarykey〉

〈X:VectorReference,DictionaryReference 或 OctetStringReference〉

不返回结果，它替换向量、词典或八位字节串的一个元素。

如果 X 是一个向量引用或八字节串引用，则索引 key 必须为 Cardinal 类型，new 是放入 key 索引位置的值，由 x 引用的对象其索引为 key 的元素用 new 来替换。如果 key 大于或等于引用对象的元素个数，则引起 RangeCheck 异常。

如果 X 是一个词典引用，则 Put 操作符按下述步骤修改 X 引用的词典：首先，在词典中查找名字元素与 key 相等的有序对，如果找到此有序对，则将它从词典中清除。其次，把有序对(key,new)加入到词典中。如果无有序对被找到并清除，且词典的容量不足以容纳新的有序对，则引起 DictionaryFull 异常。

注：词典中的名字在文件内容中是一个可执行名，见 10.1.2 的论述。

13.3.20 PutInterval 区间替换

PutInterval 操作符取三个操作数：

采用说明：

1] ISO/IEC DIS 10180 原文的条文编号为“13.3.15”，有误。同下。

〈Y:VectorReference 或 OctetStringReference〉

〈index:Cardinal〉

〈X:VectorReference 或 OctetStringReference〉

不返回结果。X 和 Y 必须为同一类型，效果是用 Y 的内容替换 X 引用对象中从索引为 Index 开始的元素，长度与 Y 对象序列的元素个数相同。

如果 index 不是 X 的有效索引，或 X 不够大，无法从 index 指出的位置容纳所有的 Y，则引起 RangeCheck 异常。

13.3.21 PutValue 存值

PutValue 操作符取两个操作数：

〈new:Any〉

〈key:Dictionarykey〉

不返回结果，Put 操作符的效果由上下文解释过程中的上下文栈决定。从栈顶的词典开始按顺序检测，上下文栈中的词典包含词典关键字 key 的最顶部的词典按如下方法改变，其中 key 对应的值替换成新值 new。如果下文栈中的词典均不包含 key，则 PutValue 操作符显示执行 Define 操作符，在上下文栈顶词典中定义有序对(key,new)。在后一种情况，如果栈顶词典没有足够的容量包含新的有序对，则引起 DictionaryFull 异常。

13.3.22 Search 查找

Search 操作符取两个操作数：

〈search:OctetStringReference〉

〈string:OctetStringReferecnce〉

返回两个或四个结果，这依赖于 String 引用的八位字节串是否有一个子串与 search 匹配。如果 string 引用的八位字节串有一个子串与 search 串相匹配，则 String 返回四个结果：

〈true:Boolean〉

〈pre:OctetStringReference〉

〈match:OctetStringReference〉

〈post:OctetStringReference〉

其中 match 是对 string 引用的八位字节串中第一个匹配子串的引用，pre 是对 string 引用的八位字节串中匹配子串之间的子串的引用，post 是对 String 引用的八位字节串中匹配子串之后未匹配的剩余子串的引用。如果 search 串不与 string 引用的八位字节串中的任何一个子串相匹配，则 Search 返回两个结果：

〈false:Boolean〉

〈string:OctetStringReference〉

其中 string 是对要查找的原串的引用。

注：不论何种情况，Search 执行后的栈顶元素均为布尔值。如果查找串与被查找串中的一个子串相匹配，则此布尔值为 true，否则为 false。

13.3.23 StoreVector 存储向量

StoreVector 操作符取 n+1 个操作数的列表，其中 n 是操作数 VectorReference V 所引用的向量的长度：

〈V:VectorReference〉

〈$object_{n-1}$:Any〉

…

〈$object_1$:Any〉

〈$object_0$:Any〉

返回一个结果：

〈V：VectorReference〉

向量引用 V 被弹出堆栈，得到其大小(n)，然后清除栈中最顶部的 n 个对象，并把它们存储到 V 所引用的向量中；最顶上的对象存储到 V 引用向量的第 n－1 个元素，最底下的对象存储到 V 引用向量的第 0 个元素，最后 V 压回栈中。

13.3.24 VectorLoad 装载向量

VectorLoad 取一个操作数：

〈V：VectorReference〉

返回结果为 n＋1 个元素的列表，其中 n 为向量 V 的长度：

〈V：VectorReference〉

〈$object_{n-1}$：Any〉

…

〈$object_1$：Any〉

〈$object_0$：Any〉

向量引用 V 被弹出堆栈，然后把 V 引用的向量，从索引为 0 的元素开始，按顺序压入栈中。最后 V 被压回栈中。

13.4 控制流程、过程和状态操作符

13.4.1 Execute 执行

Execute 操作符取一个操作数：

〈X：Any〉

可能返回结果，这依赖于它的操作数类型，见下面的论述：

如果 X 是 Operator 类型，则它被执行。

如果 X 是 Procedure 型，其效果就象是在文本中遇到一样去解释过程中的一串标记(tokens)。

如果 X 是 Identifier 类型，并且其可执行属性值为 true，执行效果就象是在文本中遇到一个可执行名一样去解释之。

如果 X 是 Identifier 类型，并且其可执行属性值为 false，或 X 是其他类型，则它被压回操作数栈。

13.4.2 Exit 退出

Exit 操作符无操作数，也不返回结果。

在 Repeat，Loop，For 或 ForAll 等循环操作符过程中 Exit 的解释效果是结束对此循环操作符的解释。在有嵌套循环操作符的情况下，仅结束包含 Exit 操作符的过程的解释。

如果在 Repeat，Loop，For 或 ForAll 等过程之外的文本中遇到 Exit 操作符，则将引起 InvalidExit 异常。

13.4.3 For 循环

For 操作符取四个操作数：

〈proc：Procedure〉

〈limit：number〉

〈increment：Number〉

〈initial：Number〉

不返回结果。过程被重复地解释，与解释 Execute 操作符一样。

在每次解释 proc 之前，作为 proc 操作数的一个值被放入操作数栈；这个值在解释 For 的过程中，从 intial 开始逐步增加到 limit，步长为 increment，直到放入操作数栈中的 proc 操作数的值大于 limit(在正步长的情况下)或当此值小于 limit(在负步长的情况)时才结束。

如果在解释 proc 的过程中遇到 Exit 操作符，则 For 操作符的解释可以在达到 limit 之前终止。

13.4.4 ForAll 循环

ForAll 操作符取两个操作数：

〈proc：Procedure〉

〈X：VectorReference，DictionaryReference 或 OctetStringReference〉

不返回结果，列举 X 引用对象的元素，对 X 引用对象的每一个元素，解释一次 proc。此过程就象遇到 Execute 操作符一样重复执行。

在每次解释 proc 之前，X 的一个元素可作为 proc 的操作数被压入操作数栈。

如果 X 是一个 VectorReference，X 引用向量的元素从索引为 0 的元素开始顺序压入操作数栈。

如果 X 是一个 OctetStringReference，则从被引用的八位字节的第一个字符开始顺序压入操作数栈，每一个字符在操作数栈中用整数表示。

如果 X 是一个 DictionaryReference，则被引用的词典元素以任意的顺序压入操作数栈，对于每一个元素，与该元素相对应的标识符首先被压入操作数栈，接着压入词典中的是与该标识符相对应的值。

ForAll 操作符的解释在对 X 中的最后一个元素解释后终止。

如果对 proc 的解释过程中遇到 Exit 操作符，则 ForAll 操作符的解释可以在所有 X 的元素被列举之前结束。

13.4.5 GetDeviceDescription 取设备描述

GetDeviceDescription 操作符无操作数，返回一个结果：

〈devdict：DictionaryReference〉

其中 devdict 是 DeviceDescription 状态变量的当前值(见 12)。

13.4.6 If 如果

If 操作符取两个操作数：

〈proc：Procedure〉

〈cond：Boolean〉

不返回结果；但如果 proc 被解释，则它可能在操作数栈中留下结果。

如果 cond 的值为 true，则过程就象遇到 Execute 操作符一样被解释。

如果 cond 的值为 false，则过程不再解释。

13.4.7 IfElse 否则

IfElse 操作符取三个操作数：

〈proc 1：Procedure〉

〈proc 0：Procedure〉

〈cond：Boolean〉

不返回结果。但 proc 0 或 proc 1 可能在操作数栈中留下结果。

如果 cond 的值为 true，proc 0 就象遇到 Execute 操作符一样被解释。

如果 cond 的值为 false，proc 1 就象遇到 Execute 操作符一样被解释。

13.4.8 Loop 循环

Loop 操作符取一个操作数：

〈proc：Procedure〉

不返回结果，此过程就象遇到 Execute 操作符一样被重复地解释，直到在解释 proc 的过程中遇到 Exit 操作符为止。

13.4.9 Noop 空操作

Noop 操作符无操作数，也不回结果，这对虚拟机状态毫无影响。

13.4.10 Repeat 重复

Repeat 操作符取两个操作数：

〈proc:Procedure〉

〈n:Cardinal〉

不返回结果。此过程象遇到 Execute 操作符一样，顺序解释 n 次，或者直到遇到 Exit 操作符才结束。

13.4.11 RestoreGraphicsState 恢复图形状态

RestoreGraphicsState 操作符无操作数，也不返回结果。RestoreGraphicsState 操作符的解释结果将使整个图形状态，包括所有状态变量的值(见第 12 章)，恢复到上次解释 SaveGraphicsState 操作符所保存的图形状态。RestoreGraphicsState 操作符的解释结果隐含着对隐式图形状态栈的一次弹出。

13.4.12 RestoreGraphicsStateXCP 恢复除当前位置外的图形状态

RestoreGraphicsStateXCP 操作符无操作数，也不返回结果。对 RestoreGraphicsStateXCP 的解释结果将使整个图形状态，包括除了当前位置外的所有状态变量的值，恢复到上次解释 SaveGraphicsState 操作符所保存的图形状态。RestoreGraphicsStateXCP 操作符的解释结果隐含对隐式图形状态栈的一次弹出。

13.4.13 RestoreSavedGraphicsState 恢复保存的图形状态

RestoreSavedGraphicsState 操作符无操作数，也不返回结果。对 RestoreSavedGraphicsState 操作符的解释结果将使整个图形状态，包括所有状态变量(见第 12 章)的值，恢复到上次解释 SaveState 操作符所保存的图形状态，如果没有作为 SaveState 操作符解释结果的图形状态被保存，则恢复到最近的上层模块开始时的图形状态。

注：RestoreSavedGraphicsState 操作符不断弹出隐式图形状态栈，直到它遇到一个由 SaveState 操作符保存的图形状态时为止，然后，它隐式地执行 RestpreGraphicsState 操作符，以恢复此图形状态。

13.4.14 RestoreState 恢复状态

RestoreState 取一个操作数：

〈S:SavedObject〉

不返回结果，RestoreState 操作符的作用是将图形状态和所有词典内容恢复到保存 S 时的状态。

如果上下文栈或操作数栈包含对 S 建立后的对象的 ObjectReference，则将引起 InvalidRestore 异常。

注：RestoreState 操作符隐含执行 RestoreSavedGraphicsState 操作符，以恢复图形状态。

13.4.15 SaveGraphicsState 保存图形状态

SaveGraphicsState 操作符无操作数，也不返回结果，解释 SaveGraphicsState 操作符将使整个图形状态，包括所有状态变量(见 12 章)的值均被保存到一个隐含的堆栈中。

13.4.16 SaveState 保存状态

SaveState 操作符无操作数，返回一个结果：

〈s:SaveObject〉

SaveState 操作符的作用是保存当前状态和所有词典的内容。

注：SaveState 操作符通过逻辑执行 SaveGraphicsState 操作符来保存图形状态。这些语义实现的正确方法依赖于具体的实现。

13.5 属性和资源操作符

13.5.1 CheckIfExecutable 检查是否可执行

CheckIfExecutable 操作符取一个操作数：

〈X:VectorReference 或 Identifier〉

返回一个结果：

〈ex:Boolean〉

其中 ex 是 X 的可执行属性值，见 11.2.1 关于可执行属性的讨论。

13.5.2 CheckIfReadable 检查是否可读

CheckIfReadable 操作符取一个操作数：

〈X:DictionaryReference 或 OctetStringReference 或 VectorReference〉

返回一个结果：

〈r:Boolean〉

如果 X 的存取属性值是 Readwrite 或 ReadOnly，则 r 的值为 true。如果 X 的存取属性值是 ExecuteOnly，r 值为 false。见 11.2.2 关于存取属性的讨论。

13.5.3 CheckIfWriteable 检查是否可写

CheckIfWriteable 操作符取一个操作数：

〈X:DictionaryReference 或 OctetStringReference 或 VectorReference〉

返回一个结果：

〈W:Boolean〉

如果 X 的存取属性值是 ReadWrite，则 W 的值为 true。如果 X 的存取属性值是 ReadOnly 或 ExecuteOnly，则其值为 False。见 11.2.2 关于存取属性的讨论。

13.5.4 FindResource 查找资源

FindResource 操作符取一个操作数

〈restype:Name〉

〈ID:Identifier〉

返回一个结果。restype 指出要查找资源的类型，它必须是下列的字面名字之一：

——ColourSpace

——DataSource

——Filter

——FontObject

——Form

——GlyphIndexMap

——FontIndexMap

——Pattern

如果 restype 的值并不是上面这些名字中的一个，将引起 UndefinedKey 异常。ID 是一个 INTERRNAL IDENTIFIER 的值，在结构中它已通过 RESOURCE DECLARATION 使之与一个资源相对应。如果没有 restype 类型的资源与 ID 相对应，则引起 InvalidResource 异常。否则，返回结果依赖于与 ID 相对应的资源类型。对于每一种资源类型，返回结果的类型和特征在下面几条中论述。

13.5.4.1 FontObject 字型对象

如果 ID 对应于一个 FontObject 资源，FindResource 返回一个 FontObject 类型的对象。

13.5.4.2 FontIndexMap 字型索引映射表

如果 ID 对应于一个 FontIndexMap 资源，FindResource 返回 Vector 类型的 FontIndexMap 的对象，其格式和内容见第 16 章中的描述。

13.5.4.3 GlyphIndexMap 字形索引映射表

如果 ID 对应于一个 GlyphIndexMap 资源，FindResource 返回 Vector 类型的 GlyphIndexMap 的对象，其格式和内容见第 16 章中的描述。

13.5.4.4 ColourSpace 彩色空间

如果 ID 对应于一个 ColourSpace 资源，FindResource 返回一个 ColourSpace 的对象向量，其格式和内容见第 15 章中的描述。

13.5.4.5 Filter 过滤器

如果 ID 对应于一个 Filter 资源，则 FindResource 返回一个 Identifier 类型的对象，它是 Filter 的名

字。过滤器名在第 22 章中的描述。

13.5.4.6 DataSource 数据源

如果 ID 对应于一个 DataSource 资源，则 FindResource 返回与 DataSource 相联系的 StreamObject 类型的对象，第 17 章论述了不同种类的数据源和过滤器，如果有的话，FindResource 操作符将适用于每一种。

13.5.4.7 Pattern 图案

如果 ID 对应于一个 Pattern 资源，则 FindResource 返回一个对 Pattern 对象的词典引用，见第 20 章中的论述。

13.5.4.8 Form 模板

如果 ID 对应于一个 Form 资源，则 Findresource 返回一个对 Form 对象的词典引用，见第 23 章中的论述。

13.5.5 MakeReadOnly 置为只读

MakeReadOnly 操作符取一个操作数：

〈X：DictionaryReference 或 OctetStringReference 或 VectorReference〉

返回一个结果：

〈X：DictionaryReference 或 OctetStringReference 或 VectorReference〉

MakeReadOnly 降低其操作数的存取属性为只读。如果操作数的存取属性是只可执行(ExecuteOnly)，则将引起一个 InvalidAccess 异常，见 11.2.2 中关于存取属性的讨论。

13.5.6 MakeExcuteOnly 置成只可执行

MakeExecuteOnly 操作符取一个操作数：

〈X：DictionaryReference 或 OctetStringReference 或 VectorReference〉

返回一个结果：

〈X：DictionaryReference 或 OctetStringReference 或 VectorReference〉

MakeExecuteOnly 降低其操作数的存取属性为只可执行，见 11.2.2 中关于存取属性的讨论。

14 坐标变换操作符

本章将定义管理坐标变换的操作符，这些操作符影响 CurrentTransformation 成像变量。伪类型变换(Transformation)及其相关的系数已在 11.3.10 中定义。

定义变换操作符时需要下面的一些定义。若 T_1 和 T_2 是两个变换，则将 T_1 所代表的映像作用到 T_2 代表的映像上，其所产生的结果映像也属于在 11.3.10 中所定义形式。表示该结果映像的变换称为 T_1 与 T_2 的积(表示为 $T_1 \times T_2$)。

注：如果 T_1 和 T_2 是两个变换，它们的矩阵分别表示为：

$$M_1=\begin{bmatrix} a_1 & b_1 & 0 \\ c_1 & d_1 & 0 \\ e_1 & f_1 & 1 \end{bmatrix}$$

$$M_2=\begin{bmatrix} a_2 & b_2 & 0 \\ c_2 & d_2 & 0 \\ e_2 & f_2 & 1 \end{bmatrix}$$

则 $T_1 \times T_2$ 变换所对应的矩阵是乘积 $M_1 \times M_2$，它们被定义为：

$$M_1 \times M_2=\begin{bmatrix} (a_1 \times a_2 + b_1 \times c_2) & (a_1 \times b_2 + b_1 \times d_2) & 0 \\ (c_1 \times a_2 + d_1 \times c_2) & (c_1 \times b_2 + d_1 \times d_2) & 0 \\ (e_1 \times a_2 + f_1 \times c_2 + e_2) & (e_1 \times b_2 + f_1 \times d_2 + f_2) & 1 \end{bmatrix}$$

这些操作符及它们语义的说明包含了对可能引起内容异常的一些条件的说明，异常是由操作符的

解释结果所引发。内容异常和对异常的处理在第 24 章中定义。除了这些操作符特有的异常外，还有普通异常，这些异常几乎在任何操作符的解释期间均可被引发。这些普通异常和它们的语义将在 24.6.2 中描述。

14.1 ScaleT(比例变换矩阵 T)

操作符 ScaleT 接受两个操作数：

〈S_2:Number〉

〈S_1:Number〉

然后返回一个结果：

〈T:Transformation〉

这里，变换 T 所对应的矩阵为：

$$\begin{bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

注：作用后，变换结果是将用户坐标系在 x 轴方向放大 s_1 倍，在 y 轴方向放大 s_2 倍。

14.2 Scale(比例变换)

操作符 Scale 接受两个操作数：

〈S_2:Number〉

〈S_1:Number〉

没有结果返回，Scale 操作符的变换效果与下面的效果相同。

{S_1 S_2 ScaleT Concat}

14.3 TranslateT(平移变换矩阵 T)

Translate 操作符接受两个操作数：

〈y:Number〉

〈x:Number〉

然后返回一个结果：

〈T:Transformation〉

这里变换 T 对应的矩阵为：

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x & y & 1 \end{bmatrix}$$

注：作用后，变换结果是把坐标原点平移到当前用户坐标系的点(x,y)处。

14.4 Translate(平移变换)

操作符 Translate 接受两个操作数：

〈S_2:Number〉

〈S_1:Number〉

没有结果返回，Translate 操作符的效果与下列操作结果相同：

{S_1 S_2 Translate Concat}

14.5 RotateT(旋转变换矩阵 T)

操作符 RotateT 接受一个操作数：

〈t:Number〉

返回一个结果：

〈T:Transformation〉

这里变换 T 对应的矩阵为：

$$\begin{bmatrix} \cos t & \sin t & 0 \\ -\sin t & \cos t & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

注：作用后，变换结果是将用户坐标系沿逆时针方向绕原点旋转 t 度。

14.6 Rotate(旋转变换)

Rotate 操作符接受两个操作数：

〈S_2:Number〉

〈S_1:Number〉

没有结果返回，Rotate 操作符的作用效果与下列操作相同

{S_1 S_2 RotateT Concat}

14.7 ConcatT(矩阵乘法)

操作符 ConcatT 接受两个操作数：

〈T_2:Transformation〉

〈T_1:Transformation〉

然后返回一个结果：

〈T_3:Transformation〉

这里 $T_3=T_1\times T_2$

14.8 Concat(复合变换)

Concat 操作符接受一个操作数：

〈T:Transformation〉

没有结果返回。如果 T_0 是 CurrentTransformation 的值，执行 Concat 后，将当前变换值改为 $T\times T_0$。

14.9 SetTrans(置当前变换)

这个操作符接受一个操作数：

〈T:Transformation〉

没有结果返回。作用效果是将 CurrentTransformation 成像变量设置为 $T\times T_i$，这里的 T_i 是当前 PICTURE 的 CurrentTransformation 成像变量的初始值。这里的当前 PICTURE 由结构处理机建立。

14.10 GetTrans(取当前变换)

GetTrans 操作符不接受操作数，它返回一个结果：

〈T:Transformation〉

对这个变换 T 来说，{T SetTrans}将把成像变量 CurrentTransformation 设置为当前值。

15 彩色空间和彩色操作符

15.1 彩色和彩色空间模型

本章定义文件如何选择彩色空间和在所选彩色空间中指定特定彩色的方法。

成像操作能使用当前选定彩色的逻辑油墨在表示设备上画图。

术语油墨是指可由表示设备显示的彩色和可见纹理。油墨可以是任何单色，包括黑色、白色或任何灰度；油墨也可是由各种单色组成的图案。所有油墨逻辑上是不透明的，它可以遮盖住先前画在同一页面图像区域上的图像元素。

由于在通常使用过程中有各式各样表示彩色的方法，所以要求所有处理彩色的系统将其彩色数据都转换成单一的标准表示，以求得能使用 SPDL 语法和语义的做法是不合理的。基于以上原因，SPDL 允许通过一个标准接口，可使用索引或专门命名的彩色表示法以及多种标准彩色表示法。

注：本标准定义了一个表示 SPDL 文件的理想化结果。合适的表示处理不需要产生这样理想化的结果，而可代之以产生一个受成像技术限制的合理的近似结果。

15.2 彩色空间

某些彩色空间直接与它们在输出设备上的表示方法有关(灰度设备,RGB 设备,CMYK 设备,KX 设备)。另外一些彩色空间则与人的视觉有关(CIE 相关的彩色空间)。某些如命名彩色和彩色映射等特有的特征也被模型化为彩色空间。

SPDL 文件通过执行 SetColourSpace 操作符来选择彩色空间,该操作影响当前彩色空间(CurrentColourSpace)成像变量的内容,彩色空间可由带有彩色空间对象(ColourSpaceObject)操作数的 SetcolourSpace 操作符来识别。彩色空间对象采用向量形式,它包含彩色空间的名字和其他相应的操作数,这些操作数在后面被称为参数。

注:彩色空间对象并不表示 SPDL 对象类型,而只是一个便于应用的术语,用以指定一个向量类型的特殊使用对象,该对象可通过 FindResource 操作符求得。

彩色空间对象可通过执行 FindResource 操作符得到。该操作符把 INTERNAL IDENTIFIR 值作为它的操作数,而 INTERNAL IDENTIFIR 通过 RESOURCE DECLARATION 与彩色空间对象联系在一起。

彩色空间对象各个元素的语义在 15.3 到 15.5 中定义,它们用来标识 SetColourSpace 操作符的每个标准彩色空间。

一旦已选中某个彩色空间,在该彩色空间中指定彩色可以通过执行 SetColour 操作符来实现,该操作符的执行会影响当前彩色 Current Colour 成像变量的内容。操作符 SetColour 的操作数是在当前彩色空间中选定特定彩色时的彩色元素;操作数的个数和有效范围随着当前彩色空间的不同而变化。

注:操作符 SetColourSpace 的部分语义是将当前彩色成像变量的内容设置为标准初始值。

执行 SetColour 操作符,以在每个标准彩色空间中选择彩色时所需的彩色元素在 15.3 到 15.5 中定义。

下面几条定义了所有彩色方法均支持的各种标准彩色空间,也定义了各种彩色空间中操作符 SetColourSpace 所需参数的语义,及操作符 SetColour 所需的各彩色元素的语义。另外还定义了在执行 SetColourSpace 操作符后,当前彩色(CurrentColour)成像变量的初始值(可以说是操作符 GetColour 的返回值)。

每条开头的彩色空间名,将是在彩色空间对象中用以标识 SetColourSpace 操作符彩色空间的名字。

15.3 CIE 和 CIE 相关的彩色空间

CIE 彩色空间允许文件使用与人的视觉有关的方法来指定彩色。一旦给定了 CIE 彩色说明,应该在实现过程中得到一致的结果(在给定物理设备的限制范围内)。CIE 彩色空间是:

——CIELAB-CIE1976($L^*a^*b^*$)-空间

——CIELUV-CIE1976(L^*U^*V)-空间

——CIEBasedABC-两级,非线性变换 CIE1931(XYZ)-空间

——CIEBasedA-消色差的 CIEBasedABC 空间

注:CIEBasedABC 彩色空间可以参数化,以形成标准的 RGB 彩色空间,见 15.3.3。

15.3.1 彩色空间/CIELAB

CIELAB 彩色空间等于 CIE1976($L^*a^*b^*$)空间。在这个彩色空间中操作符的参数是:

〈Dict:Dictionary〉

注:DICT 的内容在 15.3.1.1~15.3.1.2 中描述。

CIELAB 彩色空间中选择彩色操作符 SetColour 所需的彩色元素是:

〈b:Number〉

〈a:Number〉

〈L:Number〉

这里 L 的值必须在 0~100 范围内,而 a 和 b 的值必须在单色激励(stimuli)的光谱轨迹(spectrum

locus)内。

在本彩色空间中,执行了操作符 SetColourSpace 后的当前彩色成像变量的初始值(可以说是 GetColour 操作符的返回值)是:

〈0.0:Real〉

〈0.0:Real〉

〈0.0:Real〉

Dict 中必须的和可选的输入及其语义将在下面进行描述。

15.3.1.1 白点 WhitePoint

必须输入〈WhitePoint:Vector〉是一个有三个数字型元素的向量,向量中三个元素用以说明 CIE 1931〈XYZ〉空间中漫射(diffuse)白点的三刺激值(tristimulus Value)。每个元素的值必须大于 0,Y 的值必须是 1。向量中三个元素的顺序是:

$[X_w \quad 1 \quad Z_w]$

15.3.1.2 黑点 BlackPoint

可选输入〈BlackPoint:Vector〉是一个有三个数字型元素的向量,向量中的三个元素用以说明 CIE 1931(XYZ)空间中漫射黑点的三刺激值,每个元素的值必须大于 0。向量中三个元素的顺序是:

$[X_b \quad Y_b \quad Z_b]$

如果不存在黑点输入,则默认值为

[0 0 0]

15.3.1.3 范围

必须的输入〈Range:Vector〉是一个有六个数字型元素的向量,它用来说明彩色空间中元素 L^*、a^*、b^* 的有效范围。向量 Range 的六个元素可解释为三对边界值,每对边界对应着三对彩色元素中的一对元素。

$[L_0 \quad L_1 \quad a_0 \quad a_1 \quad b_0 \quad b_1]$

三个彩色元素中的每一元素的有效值范围可定义为:

$L_0 \leqslant L^* \leqslant L_1, a_0 \leqslant a^* \leqslant a_1, b_0 \leqslant b^* \leqslant b_1$。

注:L_0 和 L_1 的值必须在 0～100 范围内。

15.3.2 彩色空间/CIELUV

CIELUV 彩色空间等同于 CIE 1976($L^*U^*V^*$)空间,该彩色空间中操作符 SetColourSpace 的参数是:

〈Dict:Dictionary〉

注:Dict 的内容在 15.3.2.1～15.3.2.2 中叙述。

在 CIELUV 彩色空间中选择彩色的操作符 SetColour 所需的彩色元素是:

〈V:Number〉

〈U:Number〉

〈L:Number〉

这里 L 的值必须在 0～100 的范围内,而 u 和 v 的值必须在单色激励的光谱轨迹内。

在该彩色空间中,执行完 SetColourSpace 后,当前彩色成像变量的初始值(可以说是操作符 GetColour 的返回值)是:

〈0.0:Real〉

〈0.0:Real〉

〈0.0:Real〉

Dict 中必须的和可选的输入及它们的语义在下面进行叙述。

15.3.2.1 白点 WhitePoint

必须输入〈WhitePoint:Vector〉是一个有三个数字型元素的向量，向量中的元素用以说明 CIE 1931 (XYZ)空间中漫射白点的三刺激值，每个元素的值必须大于 0，Y 的值必须是 1。向量中三个元素的顺序是：

$[X_w \quad 1 \quad Z_w]$

15.3.2.2 黑点 BlackPoint

可选输入〈BlackPoint:Vector〉是一个有三个数字型元素的向量，向量中的元素用以说明 CIE 1931 (XYZ)空间中漫射黑点的三刺激值，每个元素的值必须大于 0。向量中三个元素的顺序是：

$[X_b \quad Y_b \quad Z_b]$

若不存在黑点输入，则默认值是：

[0 0 0]

15.3.2.3 范围

必须输入〈Range:Vector〉是一个六个数字型元素的向量，其元素用来说明彩色空间中元素 L^*、U^*、V^* 的有效值范围。向量 Range 中的六个元素可解释为三对边界值，每一对对应着三对彩色元素中的一对元素：

$[L_0 \quad L_1 \quad U_0 \quad U_1 \quad V_0 \quad V_1]$

彩色空间中每个彩色元素有效值的范围定义为 $L_0 \leqslant L^* \leqslant L_1$，$U_0 \leqslant U^* \leqslant U_1$，$V_0 \leqslant V^* \leqslant V_1$。

注：L_0 和 L_1 值必须在 0～100 范围内。

15.3.3 彩色空间/CIEBasedABC

CIEBasedABC 彩色空间以一个简单的色视觉(ColourVision)带理论(Zone Theory)为模型，它由非线性三色(trichromatic)的第一级结合非线性对立色(opponent Colour)的第二级组成。CIE 1931(XYZ)彩色空间的这一变换，最普通的一种用法是保证采样图像中低亮度彩色可被数字化，且逼真度(fidelity)损失最小。该彩色空间中的三个元素(可任意地命名为 A、B、C)，可以根据彩色空间如何被参数化来表示任何三个互不相关的彩色元素。

注：在使人最感兴趣的彩色元素中，能用 CIEBasedABC 彩色空间的三个元素 A、B、C 所表示的有：

——标准 RGB 空间中的 R、G、B；

——CIE 1931(XYZ)空间中的 X、Y、Z；

——NTSC 空间中的 Y、I、Q；

——SECAM 和 PAL 空间中的 Y、U、V。

该彩色空间中操作符 SetColourSpace 的参数是：

〈Dict:Dictionary〉

注：Dict 的内容在 15.3.3.1～15.3.3.8 中描述。

在 CIEBasedABC 彩色空间中选择彩色的操作符 SetColour 的彩色元素是：

〈C:Number〉

〈B:Number〉

〈A:Number〉

在该彩色空间中，在刚执行完 SetColourSpace 后，当前彩色成像变量的初始值(可以说是操作符 GetColour 的返回值)是：

〈0.0:Real〉

〈0.0:Real〉

〈0.0:Real〉

注：如果彩色元素的有效值范围不包括 0.0，则可用最接近的有效值来代替该彩色元素的默认值。

Dict 中必须的和可选的输入及它们的语义在下面加以描述。参阅那些描述，从 A、B、C 值到 CIE 1931(XYZ)空间的两级映射变换可定义为：

$$L = D_A(A) \times L_A + D_B(B) \times L_B + D_C(C) \times L_C$$

$$M=D_A(A)\times M_A+D_B(B)\times M_B+D_C(C)\times M_C$$

$$N=D_A(A)\times N_A+D_B(B)\times N_B+D_C(C)\times N_C$$

$$X=D_L(L)\times X_L+D_M(M)\times X_M+D_N(N)\times X_N$$

$$Y=D_L(L)\times Y_L+D_M(M)\times Y_M+D_N(N)\times Y_N$$

$$Z=D_L(L)\times Z_L+D_M(M)\times Z_M+D_N(N)\times Z_N$$

15.3.3.1 白点 WhitePoint

必须的输入〈WhitePoint:Vector〉是一个有三个数字型元素的向量，向量中的三个元素用以说明CIE 1931(XYZ)空间中漫射白点的三刺激值。每个元素的值必须大于0，Y的值必须是1。向量中三个元素的顺序是：

$[X_w \quad 1 \quad Z_w]$

15.3.3.2 黑点 BlackPoint

可选输入〈BlackPoint:Vactor〉是一个数字型元素的向量，向量中的三个元素用以说明CIE 1931(XYZ)空间中漫射黑点的三刺激，每个元素的值必须大于0。向量中三个元素的顺序是：

$[X_b \quad Y_b \quad Z_b]$

若不存在黑点输入，则默认值是：

[0 0 0]

15.3.3.3 范围 RangeABC

可选输入〈RangeABC:Vector〉是一个有六个数字型元素的向量，这些元素用来说明彩色空间中元素A、B、C的有效值范围。RangeABC中的六个元素可解释为三对边界值，每对对应着三对彩色元素中的一对元素：

$[A_0 \quad A_1 \quad B_0 \quad B_1 \quad C_0 \quad C_1]$

彩色空间中三对彩色元素的每一个的有效值范围可定义为：

$A_0 \leqslant A \leqslant A_1, B_0 \leqslant B \leqslant B_1, C_0 \leqslant C \leqslant C_1$

若向量RangeABC的输入并不存在，则默认值为：

[0 1 0 1 0 1]

15.3.3.4 DecodeABC

可选输入〈DecodeABC:Vector〉是一个有三个过程类元素的向量，三个过程类元素将ABC值映射为与中间LMN表示相关的线性值。向量中元素的顺序为：

$[D_A \quad D_B \quad D_C]$

每个过程把A、B、C的值作为操作数进行调用，然后返回相对的线性值。

注：因为这些过程的调用与实现时间有关，所以它们必须作为纯函数(pure function)来操作，它们不应该与任何非缺省的上下文有关，也不应该影响当前上下文。

若向量DecodeABC并不存在，则默认值是一个空过程向量。这些过程不影响操作数栈的内容：

[{ } { } { }]

15.3.3.5 MatrixABC

可选输入〈MatrixABC:Vector〉是一有九个数字型元素的向量，这些元素用来说明彩色空间中经译码的元素A、B、C与中间表示LMN之间的线性关系。向量中元素次序为：

$[L_A \quad M_A \quad N_A \quad L_B \quad M_B \quad N_B \quad L_C \quad M_C \quad N_C]$

若向量MatrixABC并不存在，缺省的矩阵是：

[1 0 0 0 1 0 0 0 1]

15.3.3.6 RangeLMN

可选输入〈RangeLMN:Vactor〉是一个有六个数字型元素的向量，这些元素用以说明中间表示的三个分量L、M、N的有效值范围。向量RangeLMN可解释为三对边界值，每一对对应于三对彩色元素之一。

$[L_0 \quad L_1 \quad M_0 \quad M_1 \quad N_0 \quad N_1]$

每个彩色元素的取值范围可定义为：

$L_0 \leqslant L \leqslant L_1, M_0 \leqslant M \leqslant M_1, N_0 \leqslant N \leqslant N_1$。

若 RangeLMN 输入并不存在，则默认值是：

[0 1 0 1 0 1]

15.3.3.7 DecodeLMN

可选输入〈DecodeLMN:Vector〉是一个有三个过程类元素的向量，这些过程将 LMN 映射为与 CIE 1931(XYZ)空间相关的线性值。向量中元素的顺序为：

$[D_L \quad D_M \quad D_N]$

每个过程将 L、M 或 N 作为操作数进行调用，然后返回相应的线性值。

注：因为这些过程调用依赖于实现时间，所以它们必须作为纯函数来操作——它们不应该依赖于任何非缺省的上下文，也不应该影响当前上下文。

若向量 DecodeLMN 输入并不存在，则默认值是一个空过程向量，这些空过程不影响操作数栈的内容：

[{ } { } { }]

15.3.3.8 MatrixLMN

可选输入〈MatrixLMN:Vector〉是一个有 9 个数字型元素的向量，这些元素用来说明中间 LMN 表示的三个译码分量 L、M、N 相对于 CIE 1931(XYZ)空间的线性描述。向量元素的顺序为：

$[X_L \quad Y_L \quad Z_L \quad X_M \quad Y_M \quad Z_M \quad X_N \quad Y_N \quad Z_N]$

若 MatrixLMN 输入并不存在，则默认值是：

[1 0 0 0 1 0 0 0 1]

15.3.4 彩色空间/CIEBasedA

CIEBasedA 彩色空间是一维的，且通常是 CIEBasedABC 彩色空间的消色差模拟(achromatic analog)。在这个彩色空间中彩色值为单一分量(可任意命名为 A)，该分量根据 CIEBasedA 彩色空间的参数方式，可以表示各种彩色元素。

注：在使人最感兴趣的彩色元素中，可用 CIEBasedA 彩色空间中分量 A 所表示的有：

——标准灰度空间中的灰度元素；

——CIE 1931(XYZ)空间中的亮度(luminance)元素 Y；

——CIE 1976(L^*a^*b)空间心理测验光亮度(psychometric lightness)L^* 元素；

——NTSC、SECAM 和 PAL 空间中的亮度元素 Y。

该彩色空间中操作符 SetColourSpace 的参数是：

〈Dict:Dictionary〉

注：Dict 的内容在 15.3.4.1～15.3.4.8 中描述。

CIEBased 彩色空间中用来选择彩色的操作符 SetColour 所需的彩色元素为：

〈A:Number〉

在刚执行完成 SetColourSpace 后，当前彩色成像变量的初始值(可以说是 GetColour 的返回值)为：

〈0.0:Real〉

注：如果彩色元素有效值的范围不包括 0.0，则彩色元素的默认值可代是最接近的有效值。

Dict 中必须的和可选的输入及其语义在下面几条中描述。参阅那些描述，从 A 值到 CIE 1931(XYZ)空间的两级映射变换可定义为：

$L = D_A(A) \times L_A$

$M = D_A(A) \times M_A$

$N = D_A(A) \times N_A$

$$X=D_L(L)\times X_L+D_M(M)\times X_M+D_N(N)\times X_N$$

$$Y=D_L(L)\times Y_L+D_M(M)\times Y_M+D_N(N)\times Y_N$$

$$Z=D_L(L)\times Z_L+D_M(M)\times Z_M+D_N(N)\times Z_N$$

15.3.4.1 WhitePoint

必须的输入〈WhitePoint:Vector〉是一个有三个数字型元素的向量，这些元素用以说明 CIE 1931 (XYZ)空间中漫射白点的三刺激值。每个元素的值必须大于 0，Y 的值必须为 1。向量中元素的顺序为：

$[X_w \quad 1 \quad Z_w]$

15.3.4.2 BlackPoint

可选输入〈BlackPoint:Vector〉是一个有三个数字型元素的向量，这些元素用以说明 CIE 1931 (XYZ)空间中漫射黑点的三刺激值。每个元素的值必须大于 0。向量中元素的顺序为：

$[X_b \quad Y_b \quad Z_b]$

若 BlackPoint 输入并不存在，则默认值是：

[0 0 0]

15.3.4.3 RangeA

可选输入〈RangeA:Vector〉是一个有两个数字型元素的向量，这些元素用以说明彩色空间中分量 A 的有效值范围。Range 元素可解释为一对边界：

$[A_0 \quad A_1]$

A 的有效值范围可以定义为 $A_0 \leqslant A \leqslant A_1$。若 RangeA 并不存在，则缺省的范围是：

[0 1]

15.3.4.4 DeccodeA

可选输入〈Decode A:Procedure〉用来将 A 的值变换成中间表示 LMN 的线性值。该过程将 A 作为其操作数进行调用，然后返回对应的线性值。

注：因为该过程的调用依赖于具体实现，所以它必须作为一个纯函数进行操作——它不应该依赖于任何非缺省的上下文，也不应影响当前上下文。

若输入 DecodeA 并不存在，则默认值是一个空过程，它不影响操作数栈的内容。

15.3.4.5 MatrixA

可选输入〈MatrixA:Vector〉是一个有三个数字型元素的向量，这些元素用以说明彩色空间中译码分量 A 相对于中间表示 LMN 的线性描述。向量中元素的顺序是：

$[L_A \quad M_A \quad N_A]$

若 MatrixA 输入项不存在，默认值为：

[1 1 1]

15.3.4.6 RangeLMN

可选输入〈RangeLMN:Vector〉是一个有六个数字型元素的向量，这些元素用以说明中间表示的三个分量 L、M、N 的有效值范围。RangeLMN 的元素可解释为三对边界，每一对对应彩色分量中的一个：

$[L_0 \quad L_1 \quad M_0 \quad M_1 \quad N_0 \quad N_1]$

三对彩色分量的有效值范围定义为：

$L_0 \leqslant L \leqslant L_1, M_0 \leqslant M \leqslant M_1, N_0 \leqslant N \leqslant N_1$。

若 RangeLMN 输入并不存在，则默认值范围是：

[0 1 0 1 0 1]

15.3.4.7 DecodeLMN

可选输入〈DecodeLMN:Vector〉是一个有三个过程类元素的向量，它的三个过程用来将 LMN 的值变换为 CIE 1931(XYZ)空间的线性值。向量中元素的顺序为：

$[D_L \quad D_M \quad D_N]$

每个过程把 L、M 或 N 作为操作数进行调用，然后必须返回对应的线性值。

注：因为这些过程的调用依赖于具体实现，所以它们必须以纯函数进行操作——它们不应该依赖任何非缺省的上下文，也不应该影响当前上下文。

若 DecodeLMN 输入并不存在，可用的默认值是一个空过程向量，它们不影响操作数栈的内容：

[{ } { } { }]

15.3.4.8 MatrixLMN

可选输入〈MatrixLMN:Vector〉是一个有九个数字型元素组成的向量，这些元素用以说明中间表示 LMN 的三个译码分量相对于 CIE 1931(XYZ)空间的线性描述。向量中元素的顺序为：

[X_L Y_L Z_L X_M Y_M Z_M X_N Y_N Z_N]

若 MatrixLMN 输入并不存在，则默认值为：

[1 0 0 0 1 0 0 0 1]

15.4 设备彩色空间 Device Colour Spaces

设备彩色空间允许文件能直接采用与输出设备表示相关的彩色值。彩色值直接对应于(或通过简单变换)设备的颜料(Colourants)，即物理染料(physical_dyes)的数量或发光体的强度。由于缺乏标准化，因此它依赖于具体实现时彩色空间的性质，所以最终结果可能因实现方法不同而变化。设备彩色空间有：

——DeviceRGB

——DeviceCMYK

——DeviceKX

——DeviceGrey

15.4.1 彩色空间/DeviceRGB

DeviceRGB 彩色空间是一个原色为红、绿、蓝的加色模型(additive colour model)。当在执行 SetColourSpace 操作符时，ColourSpace/DeviceRGB 作为它的彩色空间名操作数，不再需要别的参数。

在 DeviceRGB 空间中，选择彩色的操作符 SetColour 所需的彩色元素是：

〈B:Number〉

〈G:Number〉

〈R:Number〉

这里每个元素的值必须在 0.0～1.0 之间。0.0 表示给定的原色不起作用，而 1.0 表示该原色的最大强度。

在刚执行完 SetColourSpace 后，当前彩色成像变量的初始值(可以说是 GetColour 的返回值)是：

〈0.0:Real〉

〈0.0:Real〉

〈0.0:Real〉

15.4.2 彩色空间/DeviceCMYK

DeviceCMYK 彩色空间是一个原色为青色、品红、黄色和黑色的减色模型(Subtractive Colour model)。当在执行 SetColourSpace 操作符时，ColourSpace/DeviceCMYK 作为它的彩色空间名操作数，不再需要别的参数。

在 DeviceCMYK 彩色空间中，选择彩色的操作符 SetColour 所需的彩色元素是：

〈K:Number〉

〈Y:Number〉

〈M:Number〉

〈C:Number〉

这里每个彩色元素的值必须在 0.0～1.0 之间。0.0 表示给定的原色无光线吸收(light absorption)，

而 1.0 表示该原色的最大吸收。

在刚执行完 SetColourSpace 以后，该彩色空间当前成像变量的初始值(可以说是 GetColour 的返回值)为：

〈1.0:Real〉
〈0.0:Real〉
〈0.0:Real〉
〈0.0:Real〉

15.4.3 彩色空间/DeviceKX

彩色空间 DeviceKX 是一个减色模型，其原色是黑色和一个未指定的强光色(highlightColour)。在执行 SetColourSpace 操作符时，若把彩色空间名 ColourSpace/DeviceKX 作为它的操作数时，不再需要别的参数。

在彩色空间 DeviceKX 中，选择彩色的操作符 SetColour 所需的彩色元素是：

〈K:Number〉
〈X:Number〉

这里每个元素的值必须在 0.0～1.0 之间。0.0 表示给定的原色没有光线吸收，而 1.0 表示该原色的最大吸收。

在该彩色空间中，刚执行完 SetColourSpace 后，当前彩色成像变量的初始值(可以说是操作符 GetColour 的返回值)为：

〈1.0:Real〉
〈0.0:Real〉

15.4.4 彩色空间/DeviceGrey

彩色空间 DeviceGrey 是一个可以指定灰度值的一维彩色空间。在效果上，它也是在当前执行上下文中第一次执行 SetColourSpace 操作符之前的初始彩色空间。在执行 SetColourSpace 操作符时，把彩色空间名 ColourSpace/DeviceGrey 作为它的操作数，不再需要别的参数。

在 DeviceGrey 彩色空间中选择彩色(灰度级)操作符 SetColour 所需的彩色元素是：

〈Grey:Number〉

这里 Grey 的值介于 0.0～1.0 之间。0.0 用于表示黑色。1.0 用于表示白色。

在该彩色空间中。刚执行完操作符 SetColour 时，当前彩色成像变量的初始值(可以说是操作符 GetColour 的返回值)是：

〈0.0:Real〉

15.5 特殊目的的彩色空间

某些特征亦被模型化为彩色空间，这包括为彩色变换(映射)作索引和为命名彩色而引入的彩色空间，这些特殊目的彩色空间有：

——Indexed：标准彩色空间映射对照表(彩色表)
——NameColour：命名的彩色

15.5.1 彩色空间/Indexed

由于效率上的原因，许多输入和编辑设备使用了专用设备特有的彩色对照表，所以必须要有一种通用的转换方法，把这些对照表或索引指出的值转换成一个标准彩色空间中的彩色。Indexed 彩色空间提供了一种把小整数(Small integers)映射为任何一个标准彩色空间中任意色的方法。在 Indexed 彩色空间中，操作符 SetColourSpace 的参数是：

〈Loolup:过程或八位字节串〉
〈HighValue:基数(Cardined)〉
〈BaseColourSpaceObject:Vector〉

参数 BaseColourSpaceObject 指出了索引所映射的标准彩色空间。

参数 HighValue 是索引空间的上界。

最后一个参数 Lookup 是一张彩色对照表，该张表提供了索引值和基本彩色空间的彩色值之间的映射关系。如果对照表是一八位字节串类，则它的长度必定为 m * (HighValue+1)，这里 m 是基本彩色空间中彩色元素数目。Lookup 可作为一个过程类型的值或作为一八位字节串类的值，其语义将在下面进行描述。

在 Indexed 彩色空间中，选择彩色的操作符 SetColour 所需的彩色元素是：

〈Index:Number〉

若 Index 是一实型类，则它被截成整数，如果它超出了 0～HighValue 的取值范围，则它取最接近的边界值。

如果 Lookup 是一个过程类，索引值则被压入操作数栈，然后调用 Lookup。Lookup 必须将所给的索引值转换为一组彩色元素，然后必须把这些彩色元素返回操作数栈。压入栈的格式和顺序必须与基本彩色空间中操作符 SetColour 相一致。然后 SetColour 操作符对这些返回值就象在基本彩色空间中一样进行操作。Lookup 应该作为一个没有副作用的纯函数进行操作，并且必须对任何索引值(0-HighValue 之间)返回彩色元素值。

另外，如果 Lookup 是八位字节串类，则 Index 要乘以基本彩色空间中彩色元素的数目(为了方便，可称为 Numcomp)，然后用此乘积作为查彩色表的索引。Lookup 中对应上述乘积位置上的 NumComp 个八位字节被解释为基数类型的编码值，该编码值对应于基本彩色空间中的 NumComp 个彩色元素，该编码值用 255 去除，以产生 0～1 范围内的元素值，操作符 SetColour 就象在基本彩色空间中一样对这返回值进行操作。

在 Indexed 彩色空间中，刚执行完操作符 SetColourSpace 后，当前彩色成像变量的初始值(可以说是操作符 GetColour 的返回值)是：

〈0:Integer〉

15.5.2 彩色空间/NamedColour

有时引用一个工业标准或专门的彩色规范来指定彩色是很有用的。NamedColour 彩色空间提供了用这种方法指定彩色的一般机制。在此彩色空间中操作符 SetColourSpace 的参数是：

〈TintToColour:Procedure〉

〈SelectColourSpace:Procedure〉

〈NamedColour:Identifier〉

参数 NamedColour 是专门颜料、点彩色、分色等等的名字，如果表示过程能直接使用 NamedColour，则另外两个过程类参数可忽略。

如果表示过程不能直接使用 NamedColour，则执行参数 SetColourSpace，以提供一个任意可替补的彩色空间的描述，NamedColour 可映射到该替补彩色空间。这个过程必须在操作数栈返回替补彩色空间所用的参数和彩色空间名操作数，在栈中的格式和顺序应与操作符 SetColourSpace 所需的相同。NamedColour 的色彩经随后执行的操作符 SetColour 后，将被解释为执行 TinToColour 过程参数后的替补彩色空间。正如下面所述。

在 NamedColour 彩色空间中，操作符 SetColour 所需的彩色元素是：

〈Tint:Number〉

这里 Tint 的范围是 0.0～1.0。值 0.0 代表颜料的量用得最少，值 1.0 代表颜料的量用得最大，而中间值表示使用中间数量的颜料。

如果表示过程不能在 SetColourSpace 操作中直接使用 NamedColour，则 Tint 的值被压入操作数栈，然后调用过程 TintToColour。TintToColour 必须把所给的 Tint 转换为一组替补彩色空间中的彩色元素，而该彩色空间是通过将 SelectColourSpace 参数赋于操作符 SetColourSpace 而得到的。TintToColour

必须将这些彩色元素返回操作数栈，在替补彩色空间中，这些返回栈中的值在形式和顺序上，必须是操作符 SetColour 可访问的，然后 SetColour 操作符在替补彩色空间对这些返回值进行操作。TintToColour 应该作为一个没有副作用的纯函数来操作，而且能对任何有效的 Tint 值返回相应的彩色元素值。

在彩色空间 NamedColour 中，刚执行完操作符 SetColourSpace 后，当前彩色成像变量的初始值（可以说是操作符 GetColour 的返回值）是：

〈1.0:实型〉

15.6 操作符 Operators

对操作符及其语义的描述包括了对某些条件的说明，而这些条件可能导致由操作符解释结果而引发的内容异常。内容异常和异常处理在第 24 章定义。除了这些操作符特有异常外，还有类属性异常（普通异常），这种异常几乎可在任何操作符解释时被引发。这些类属性异常和它们的语义在 24.6.2 描述。

15.6.1 SetColourSpace

操作符 SetColourSpace 接受下列操作数：

〈ColourSpaceObject:VactorReference〉

这里的 ColourSpaceObject 是由执行 FindResource 操作符得到。在 ColourSpaceObject 中，元素的顺序应该是这样的：如果把 ColourSpaceObject 作为操作数，执行 VectorLoad 操作符，则将在操作数栈返回下列结果：

〈param n:Any〉

⋮

〈param 1:Any〉

〈ColourSpaceName:Name〉

这里 ColourSpaceName 是被选彩色空间的名字。而其余的参数按其顺序相应地已在 15.3～15.5 中说明。

操作符 SetColourSpace 并不返回结果，执行 SetColourSpace 的影响是：

——使得由 ColourSpaceObject 定义的彩色空间成为当前彩色空间；

——将当前彩色空间（CurrentColourSpace）成像变量的值设置为 ColourSpaceObject；

——将当前彩色（CurrentColour）成像变量的值设置为相应的初始值（见 15.3～15.5）。

15.6.2 GetColourSpace

操作符 GetColourSpace 不需要操作数，它返回当前彩色空间成像变量的值。

如果操作符 SetColourSpace 还没有在当前执行上下文中执行，则 GetColourSpace 返回当前彩色空间成像变量的初始值：单个元素向量，该元素的值为：

〈DeviceGreyp:Name〉

15.6.3 SetColour

SetColour 操作符接受下列操作数：

〈Comp n:Any〉

⋮

〈Comp 1:Any〉

这里彩色元素操作数的值和语义依赖于 CurrentColourSpace 成像变量的值，它们已在 15.3～15.5 的有关条进行叙述。无结果返回，执行 SetColour 操作符的结果是去设置 CurrentColour 成像变量的值。

15.6.4 GetColour

GetColour 操作符无操作数，并在操作数栈返回 CurrentColour 成像变量的值，其格式为标记对象（Mark Object），紧接着为在当前执行上下文中最后一次执行 SetColour 或 SetPatternColour 的操作数，它们已经存放在操作数栈。

如果在当前执行上下文中没有执行 SetColour 或 SetPatternColour 操作符，则 GetColour 返回当前

彩色空间(见 15.3～15.5)CurrentColour 成像变量的初始值。

15.6.5 SetOverPrint

操作符 SetOverPrint 接收单个操作数:

〈OverPrint:Boolean〉

无结果返回,执行该操作的结果是把 CurrentOverPrint 成像变量的值设置成 OverPrint 操作数的值。

描绘一个图形元素需要一个表示过程,用以决定是否形成独立的分色(Coloursepratations)。在当前执行上下文中,若一个表示过程并没有形成分色,那么 CurrentOverPrint 成像变量的值是非法的;若形成分色,那 CurrentOverPrint 成像变量的值可以用来决定是否去描绘一个图形元素。

当形成分色时,CurrentOverPrint 成像变量的值可以用来决定是否在对应于其他分色区域画图形元素,与当前彩色空间无关。CurrentOverPrint 成像变量的初始值为假(false),致使所画图形元素去清除与当前彩色空间无关的分色区。

15.6.6 GetOverPrint

GetOverPrint 操作符无操作数,返回一个结果:

〈OverPrint:Boolean〉

这里 OverPrint 是 CurrentOverPrint 成像变量的值。

16 字符文本和字型的操作符

描绘字符文本的过程与字型的结构有着密切联系,在本标准中采用的字型结构的定义见 ISO/IEC 9541 信息处理:字型信息交换。字符文本和字型模型在 16.1 中建立,16.2 和 16.3 中定义了数据结构和操作符。如果在一个描绘字符文本的 SPDL 文件中,使用 ISO/IEC 9541 中指定属性和结构的字型,产生的结果可以通过这种数据结构和操作符来准确地定义。本章将讨论这一描述的基础——字型模型。

16.1 字符文本和字型的模型

字符文本和字型的模型由几个子模型组成。字型模型在 16.1.1 中建立,定义字形和字形序列的模型在 16.1.2 中建立,字型引用和表示的模型在 16.1.3 中建立。

16.1.1 字型

在本标准中采用的字型模型已在 ISO/IEC 9541 信息处理:字型信息交换中建立。

一个(抽象的)字型是一个具有相同基本设计风格的图像的集合,例如,Garamond Italic 字型,再加上一些描述这些图像共性和个性的信息。这些图像代表了特定的图形符号,称之为字形(Glyph)。用它们表示的信息适合于人们阅读。字形不依赖于表示它们的任何特定的图像,因此字型就是一组用来表示字形图像和关于这些图像信息的集合。

16.1.1.1 字型资源

ISO/IEC 9541 使用了"字型资源"这一术语来代表字型。选用一个特定字型来表示字符正文,换句话说也就是指定一个代表该字型的字(或字型)来表示文本。

字型资源是由字形描述的集合再加上与字形描述集合总体上有关的说明信息和字符的尺度信息(font metrics)组成。字形描述由形状信息再加上与各单个字形有关的属性组成。字库资源中的字形描述可通过 ISO/IEC 9541 中定义的结构化的名字来标识。

注:在现有应用系统中使用的字型一般并不完全根据 ISO/IEC 9541 的定义,但这些用来表示字符文本的字型属性基本上等价于 ISO/IEC 9541 中所描述的属性,因此就有可能生成结构和属性符合 ISO/IEC 9541 中所定义的字型资源,使用这种字型资源来表示字符文本会产生同样的效果。在文件中用那些不同于 ISO/IEC 9541 定义的字型来表示字符文本的过程,可以理解为是采用结构和属性完全符合 ISO/IEC 9541 的字型资源来定义逻辑上等价的字型来表示字符文本。

16.1.1.2 字符的大小和方向

所有表示字符文本的字形属性在一个字形坐标系中定义,这个坐标系的单位与字符的额定大小成

比例。因此通过对字形轮廓描述和字的尺度信息的适当变换，字型资源可以用来在任意方向上表示任意大小的字符文本。

16.1.1.3 字符的变比和旋转

特定字型资源中的字形形状是一个可以对它进行所有图形操作的基本图形轮廓，这些操作可通过坐标变换来实现。用某特定字型表示的文本可以在一定范围内变化，可以用肖像模式、风景画模式或者沿媒体的任意方向进行旋转。正因为把字形形状当作普通的图形来处理，所以可以在描绘整个文本字符串的时候，通过改变 Current-TransFormation 来对整个字符串进行变比和旋转。

16.1.1.4 粗体和斜体字

一个特定的字型只有一套(可变比、可旋转的)字形形状。要想表示粗体或斜体的字符正文，必须选择包含相应字形形状的字型。

16.1.1.5 写模式

在表示多国文字时通常要求字型资源中的字形轮廓能够使用不同的书写模式：包括从左到右、从上到下、从右到左的书写模式。一个具有确定大小和方向的字型可以用不同的书写模式来表示文本。

一个字型资源中只有一套字形形状信息，但可以有多种属性集合，每个集合对应于一种书写模式。SPDL 中与其重要属性相关的写模式是定点坐标(ISO/IEC 9541/PX 和 ISO/IEC 9541/PY)和转义(Escapment)点坐标(ISO/IEC 9541/EX 和 ISO/IEC 9541/EY)。

16.1.2 字形和字形映射

16.1.2.1 字形标识符

字型资源和字形标识符一起唯一地决定了要加入页面图像的字形描述。

对于字符文本中的每个元素，由构图和布局过程来选择表示该元素的相应的字形描述。用来表示字符文本的文件元素仅仅识别由用于这一目的的构图和布局过程所选择的字型和字母标识符。

16.1.2.2 索引

16.1.2.2.1 字形索引

在 SPDL 中要有效地表示字符文本，就要求用一种紧凑的方法来指定文件中的字形序列。在 SPDL 中，用字形串(一种基于八位字节串类型的伪类型)来指定字形序列，它提供了与当前字型中的字形标识符相对应的整型值，这些整型值称为字形索引。

16.1.2.2.2 字型索引

SPDL 能够把多种字型组合为一个称为复合字型(见 16.1.3.3.2)的层次结构。当当前字型是这样一种复合字型时，字形串所提供的某些整型值，对应着并且指定了这一复合字型中的某个字型，这些整型值称为字型索引。

16.1.2.3 串

字符文本的一个重要方面就是用表示它的字形顺序组织成一个称为“串”的字形序列，SPDL 用伪类型 Glyphstring 和几个操作符来反映这种结构，从而优化了表示这种字形序列的过程。

Glyphstring(字形串)是一个八位字节串，这个八位的序列用以衍生出一个基数(Cardinal)序列。这些基数可进一步解释为字形索引，且有时解释为字型索引，用以指示将显示的字形。

16.1.2.4 字形映射

SPDL 为每个字型对象建立了相关的字形映射。这种映射定义了一种算法，它把字形串中的八位字节解释为字形索引，当当前字型对象为复合字型时，就把它解释为字型索引。字形映射指出了字形标识符和字形索引之间的联系。对于复合字型，还说明了单个字型对象和字库索引之间的联系。

特定的字形映射和它们对于字形串的应用在 16.2 中定义。

注

1 字形串说明了采用特定字形映射变换的特种字型来描绘字形序列的过程。当解释执行以 OctetStringReference 为操作数的操作符时，根据成像变量 CurrentFont 的值建立用于描绘字形串的字型和映射变换。表示字形串的

方法在 16.2 和 16.3 中定义。

2 重要的一点是：必须强调 SPDL 语法仅仅规定了需要取得的效果，而不是指出用以产生这种效果的方法。本节的定义是基于用字形标识符和用字形索引与字形标识之间的明确的映射关系来唯一确定字型资源中的字形描述，但在实际应用中可能在内部对这些字形描述和字型信息进行组织，以便优化这一映射关系。许多用于表示 SPDL 文件的字型并不遵照信息交换格式，且在它们的内部表述中也隐含着假定的字母映射关系，只要能够达到本标准定义的效果，使用这种字型也是可行的。

16.1.3 字型的引用和描述

SPDL 定义了引用和描述字型的四种类型的数据：

——一字型的 RESOURCE DECLARATION(结构)　　(一字型的资源说明)

——一字型的 RESOURCE DEFINITION(结构)　　(一字型的资源定义)

——FONT DICTIONARY(SPDL 数据结构)　　(字型词典)

——FONTOBJECT(SPDL 数据结构)　　(字型对象)

下面的条中将讨论它们的定义。

16.1.3.1 字型资源说明

字型的 RESOURCE DECLARATIONS 已在第 7 章中定义。字型的 RESOURCE DECLARATIONS 用 REFERENCE NAME 来标识一个字型对象，并在上下文的解释过程中赋给它一个名字，然后在文件内容中可以用操作符 FindFont 来引用选中的字型资源。

16.1.3.2 字型资源定义

字型的 RESOURCE DEFINITIONS 在第 7 章中定义。一个字型的 RESOURCE DEFINITION 提供：

——包含在 SPDL 结构中的字型对象，可以是遵照标准附录 C 中定义的交换格式的字型对象出现，也可以以 16.2.1 中定义的字型词典形式出现，这样在文件结构中可以用字型 RESOURCEDECLARATION 来引用这种字型。

——改变一个特定字型对象的字形映射关系。

——引用一个 ISO/IEC 9541 字型资源。有可能附加一些信息，以便如果指定的字型资源无效的话，允许表示过程去选择一个替换的字型资源。

16.1.3.3 字型词典

字型词典是一种表示字型资源的结构和属性的数据结构，这里的字型资源的结构和属性完全按照 ISO/IEC 9541 的定义，且与 SPDL 内容记法完全一致。通过对标记(token)序列的解释可以构造一个字型词典，且该词典再用以创建在表示文本文件中使用的字型对象。

注：除了表示字符文本所必须的信息之外，字型词典还包括一些不为 SPDL 所用的字型和字形属性。例如，它可以包括 ISO/IEC 9541 所定义的任何附加属性。根据字型词典创建字型对象过程中任何这样的属性均是无定义的，并且在 SPDL 使用字型对象时它们也不起任何作用。

有两种字型词典：基字型词典和复合字型词典。从基字型词典中衍生出的字型对象称为基字型。从复合字型词典中衍生出的字型对象称为复合字型。

16.1.3.3.1 基字型词典

基字型词典是形式上把 ISO/IEC 9541 所定义的字型属性与 SPDL 联结起来的一种数据结构，它是用 SPDL 对象类型对 ISO/IEC 9541 所定义的字型资源的结果和属性所作的描述。任何结构和属性与 ISO/IEC 9541 一致的字型资源均可以由一个 SPDL 基本字型词典来表示。

用以描述 ISO/IEC 9541 中的字型属性的两种结构是结构化名字和特征表。特征表在 SPDL 中用下面两种对象类型之一来表示：

——一个向量类型的对象，该向量是一张特征值的有序表，或者

——一个词典类型的对象，其中：

——词典中有序对的标识符元素对应于标识 ISO/IEC 9541 特征表中属性的结构化名；

——有序对中的值元素是属性值，属性值本身可能是词典引用，向量引用或者是象数值这样的简单对象。

在 SPDL 基字型词典中可用三种途径之一来表示 ISO/IEC 9541 的结构化名：

——由于效率或历史上的原因，ISO/IEC 9541 中的有些结构化名在保持它们原语义的基础上往往进行截断或修改，以适用于 SPDL 的字型词典。在 SPDL 中它们以 Name 数据结构的形式出现。

——其他的结构化名在 SPDL 中采用 identifier 数据类型来表述。这种标识符与对一个包含结构化名文本的八位字节串按照结构化名的描述实施 ConvertToIdentifier 操作所得到的结果完全相符。

——ISO/IEC 9541 中登录的字形标识符在 SPDL 字型词典中以 afii:xxxx 形式的名字来标识。这里的 xxxx 是由一到八个值在 0～9、A～F 之间的 GB 1988 编码的字符序列组成，以一个非零数开头，是分配给字形标识符的标准注册号的十六进制表示。对一个对应于该国际注册号的基数进行 ConvertToIdentifier 操作得到的结果与这个名字是等价的。

除了 ISO/IEC 9541 定义的属性外，SPDL 字型词典还包括：

——一个初始变换，描述了与用户坐标系相关的字形坐标系的大小和方向，以及一个额定单元大小的字型（在应用任何 ScaleFont 或 TransformFont 之前）。

——字型的默认写模式。

——默认的字形映射。它将字形索引与字形标识符联系起来（见 16.2.1.1）。

16.1.3.3.2 复合字型词典

复合字型是一个层次数据结构，它把基字型（和其他复合字型）组合成大的复合字型。复合字型要求比基字型更复杂的字形映射关系。

复合字型词典通常是一个定义复合字型特性的数据结构。复合字型词典的内容和基本字库词典相似，但是具体的字形形状信息由各独立字型来描述。

16.1.3.4 字型对象

字型资源提供了一种用于字型信息交换的恰当的机制。用一个特定字型表示字符文本的过程需要一个更适合于表示过程的数据结构，这种数据结构元素仅需要那些在表示过程中真正需要的信息，且只需要那些用来达到基本目的的附加操作。

字型对象是一个 DictionaryReference 类型的对象，它指向一个包含用某个特定字型表示字符文本过程中所必需信息的词典。一个字型对象的词典至少要有：

——字形形状信息；

——使用字型资源时用以顺序定位字形形状的典型字形的尺度信息；

——描述用户坐标系和字形坐标系之间关系的变换，在用户坐标系中所用字型资源的大小和方向；

——与 FontObject 相关的写模式；

——一种将字形标识符与字形索引联系起来的字形映射关系，如果是复合字型，该映射将单个字型对象和字型索引联系起来。

字型对象可用 DefineFont 操作符来组织，也可以用 FindFont 操作符从表示过程环境的所有可用信息中选择，或者由那些修改已存在字型对象的操作符来产生。

注

1 SPDL 范围之外有一个字型安装过程。它接收一个遵守 ISO/IEC 9541 定义的用来进行信息交换的字型资源，从而产生一种可以用来创建为 SPDL 处理器所用字型对象的数据结构。这一过程完成的功能和 DefineFont 操作符类似，主要差别在于字型安装过程仅仅为被安装字型装配一种默认的变换和字形映射，而且字型安装过程必须保留足够的与被安装字型有关的属性信息，以便一旦该字型最符合于由字型 RESOURE DEFINITION 所指出的字型引用，表示过程能够顺利地选择到它。

2 文件中基字型对象的字形映射关系在结构上可以按字型 RESOURCE DEFINITION 进行修改，在内容上可以用 OpenFont 和 DefineFont 操作符来修改。复合字型对象的字母映射除了字型安装过程之外一般不可修改，然而可用字形索引去映射 RESOURCE DEFINITION，从而在结构上生成新的复合字型映射，或者通过对相应

的复合字型词典的有效操作从而在内容上生成新的映射。

3 本节中字型对象和字型词典的定义规定了它们的逻辑结构，并为随后一些小节中定义描绘字符文本所用字型打下基础。并不要求应用系统真的通过对应的字型词典来存取字型对象，制定本标准的意图在于推动该领域中优化技术的有效运用(因此就有了涉及字型对象的条文。见16.2.2)。本标准要求的仅仅是解释执行字符文本和字型操作符所达到的效果必须符合本标准定义。

16.2 数据结构

定义字符文本操作符需要定义两种专门用于表示字符文本的特殊数据结构，它们是字型词典和字型对象。在字型词典中还定义了有效的字形映射算法。

16.2.1 字型词典

字型词典是包含〈FontType:Integer〉和一些必要的附加项的词典，这些附加项依赖于FontType的值。在下面的小节中将详细说明每种字型词典的语义，本标准为FontType定义了下面几种允许的值：

——4：在SPDL内容记法中可以指定具有FontType 4的汉字字型词典。

——3：在SPDL内容记法(notation)中完全可以指定具有FontType3的字型词典，FontType 3字型词典的语义是所有基字型语义的基础。

——1：FontType 1的字型词典对应于利用标准附录C中定义的字型对象交换格式所创建的基字型对象(见16.2.2)。FontType 1字型词典的语义是根据FontType 3字型词典的语义定义的。

——0：FontType 0的字型词典对应于复合字型对象，FontType 0字型词典的语义定义了所有复合字型的语义。

16.2.1.1 基字型词典

16.2.1.1.1 FontType 4字型词典

包含下面一些必须的项和可选的项：

——〈FontType:Integer〉，必须项；

——〈FontType:Name〉，必须项；

——〈FontType:Tramsformation〉，必须项；

——〈FontType:VectorReference〉，可选项；

——〈Encoding:VectorReference〉，可选项。UCS代码(或GB代码)到系统内部编码的映射串。

可能的值为：

UCS-2，双八位 UCS码，默认值；

UCS-4，肆八位 UCS码；

GB；

——〈Metrics:Procedure〉，必须项。

该过程能够针对不同的写模式(WMode)和不同的Encoding来计算字符的定位点和宽度信息。

——〈Private:Dictionary〉，可选项。

其值由字型生产者规定，此词典用来支撑CharStrings项。

——〈Construct Glyph:Procedure〉，必须项。

其值由字型生产者规定，该过程为由用户自己描述的汉字。该项用来描述各个字符的轮廓信息。

——〈PaintType:Integer〉，可选项。可能的值为：

0：实心字

2：空心字

——〈WMode:Integer〉，可选项。可能的值为：

0：表示从左到右，默认值；

2：表示从右到左；

1：表示从上到下；

3：表示从下到上。

16.2.1.1.2 FontType 3 字型词典

FontType 3 字型词典中所必须的项除了〈FontType：Integer〉之外还有：

——〈FontMatrix：Transformation〉：这是根据字型词典创建的对应字型对象所用变换的初始值或默认值。

——〈FontBBox：VectorReference〉：FontBBox 的值是对于一个有四个数值向量的引用，它代表 ISO/IEC 9541/MAXFONTEXT 的值。顺序为(maxfontext-minx，maxfontext-miny，maxfont ext-maxx，maxfontext-maxy)。

——〈Encoding：VectorReference〉：Encoding 的值是一个向量引用，向量元素的类型是标识符，这些标识符构成了字形索引映射表(Glyph Index Map)，见 16.2.1.1.3。这个表是根据字型词典创建相应的字型对象时使用的默认值。

——〈Metrics：DictionaryReference〉：Metrics 的值是引用一个词典的 DictionaryReference，该词典包含有每个字形的范围、字型默认写模式下字形转义(Escapments)等信息。如下所示：

——被引用词典中的名字单元是字形标识符。

——每个项的值是对于一个向量的引用，该向量含有顺序的六个数：(wx，wy，minx，miny，maxx，maxy)。

——〈wx：Number〉的值是 ISO/IEC 9541/EX-ISO/IEC 9541/PX 之差。

——〈wy：Number〉的值是 ISO/IEC 9541/EY-ISO/IEC 9541/PY 之差。

——余下四个数的值是 ISO/IEC 9541/EXT 的组成部分，它表示字形的范围。

——写模式依赖于定位点(ISO/IEC 9541/PX 和 ISO/IEC 9541/PY)，在 FontType 3 字型中所有字形的缺省写模式将是(0，0)。

——〈ConstructGlyph：Procedure〉：ConstructGlyph 的值是一个负责构造字形表示的过程，它的语义在 16.3.16 中定义。

注：推荐的 ConstructGlyph 过程(见 16.3.13)将利用字型词典中另一个名字为〈GlyphProcs：DictionaryReference〉的项。GlyphProcs 的值指向一个包含每个字母表示过程的词典，如下所示：

——被引用词典中的名字元素是字形标识符。

——每项的值是一个画出由名字指出的字形表示过程。

在本标准中定义了语义的 FontType 3 的字型词典的可选择项有：

——〈WMode：Cardinal〉：WMODE 的值指出在使用这个字型中的字形描述时如果不采用默认写模式时所使用的写模式。如果该字型中不存在 WMODE，这个值默认为零，指出了默认的写模式。一个大于零的值 N，指出了第 N 种可替换的写模式，它在向量 OtherMetrics 的(N－1)项中作了说明(见下面)。如果这种字型并不支持由 WMODE 值指出的写模式，或者 WMODE 的值大于应用系统中 WMODE 值的最大限度值，那么文件内容解释器在用这种字型表述文本时将使用默认模式。

——〈OtherMetrics：VectorReference〉。OtherMetrics 的值是对 DictionaryReference 向量的引用。每个 DictionaryReference 将引用一个词典，而该词典包含有每个字形定位点的信息和该字型可替换写模式的转义信息，如下所示：

——被引用词典中的名字元素是字形标识符。对于该词典中的每一项，在 Metrics 字典中应该有一个具有相同名字元素的项。

——每个项的值是一个向量引用，该向量的四个值依次为(wx，wy，px，py)：

——〈wx：Number〉的值是 ISO/IEC 9541/PX 之差。

——〈wy：Number〉的值是 ISO/IEC 9541/PY 之差。

——px 和 py 是该写模式下该字形定位点(ISO/IEC 9541/PX，ISO/IEC 9541/PY)的 X、Y 坐标。

除了必需的项和为支持可替换的写模式而设置的可选择的项以外，字型词典应该包含所有用以支撑 ConstructGlyph 项的其他项以及其他可能需要的项，描绘字形不需要的项将被忽略。

最后，所有基字型词典必须有一个字形标识符为〈.notdef:Name〉的字形描述，每当指出需描绘的字形标识符在该字型中未定义时，也就是说该字形的形状描绘信息在该字型中不存在时，那就利用上述字形描述，与这个特殊字形标识符相关的字形描述由字型设计者确定。

16.2.1.1.3 FontType 1 字型词典

FontType 1 字型词典是参照 FontType 3 字型词典语义来定义的。FontType 1 字型词典中必须的项在标准附录 C 中给出。它们的语义如下：

——〈FontMatrix:Transformation〉，

〈FontBBox:VectorReference〉，

〈Encoding:VectorReference〉：这些语义和 FontType 3 字型词典的相同。

——〈PaintType:Cardinal〉：值必须是 0 或者是 2。0 表示字形描述将填充轮廓线(实心)；2 表示字形描述仅勾画轮廓线(空心)。

—〈Charstrings:DictionaryReference〉：指向一个词典，该词典包含每个字形的描述过程。如下所述：

——被引用词典中的名字元素是字形标识符。

——每个项的值指向一个八位字节串。该字节串可解释为一个按 ISO/IEC 9541.3 中定义的格式加密了的字形过程。

下面是出现在 FontType 3 字型词典中的项，在 FontType 1 字型词典中并不存在，它们的语义被置换成：

——〈Metrics:DictionaryReference〉：每个字形的范围和转义等信息，它包含在 CharStrings 词典的字形处理过程中；因此，在字形描绘过程中不需要在 Metrics 词典中查找字形尺度信息，字形处理过程中所包含的转义和定位信息能够由可选的 Metrics 词典(见下)信息所代替。但是这种词典项的语义和 FontType 3 字型词典中 Metrics 字典项的语义是不同的。

——〈ConstructGlyph:Procedure〉：字形创建过程是 FontType 1 字型词典中所固有的，且总是有如下语义(见 16.3.13)：

——使用字形标识符从这个字型的 CharStrings 词典中装入相应的字形过程；

——根据 ISO/IEC 9541.3 中定义的语义解释执行字形过程。

由本标准定义语义的 FontType 1 字型词典的可选择项有：

——〈Metrics:DictionaryReference〉：Metrics 的值引用一个包含每个字形信息的词典，这些信息能够取代字形过程中的某些信息。如下所述：

——被引用词典中的名字元素是字形标识符。这和字型词典中用 CharString 引用的词典中部分或全部字形标识符是一样的。

——每个项的值可能是如下之一：

——一个简单数:wx；

——一个对含有两个数值(sbx,sby)的向量的引用；

——一个对含有四个数值(sbx,sby,wx,wy)的向量的引用。

——FontType 1 字型词典中的 CharStrings 词典的每个字形过程，以一个 hsbe 或 sbe 字母过程操作符作为开始(见 ISO/IEC 9541.3 中对这些字形过程操作符语义的描述)，数值 sbx，sby，wx，wy 将取代对 hsbe 或 sbe 操作符的调用，如下所述：

——〈wx:Number〉的值是默认写模式的一个新的转义(x 分量)；这是 ISO/IEC 9541/EX-ISO/IEC 9541/PX 之差。

——〈wy:Number〉的值是默认写模式的一个新的转义(y 分量)；这是 ISO/IEC 9541/EY-ISO/IEC 9541/PY 之差。如果 wx 已指定而 wy 未指定，那么 wy 将赋值 0。

——〈sbx:Number〉的值是字形表示过程新的初始参考点的 x 分量。这个分量的不同值将引起字形在水平方向上以不同写模式进行显示，总数为(取代值-字形过程值)。如果未指定该值，则 sbx 值不会改变。

——〈sby:Number〉的值是字形表示过程新的初始参考点的 y 分量。这个分量的不同值将引起字形在垂直方向上以不同写模式进行显示，总数为(取代值-字形过程值)。如果 sbx 和 sby 都未指定，则 sby 值不会改变。如果指定了 sbx，但未指定 sby，那么 sby 被赋值 0。

注：一个字型若有从左到右这一默认的写模式，则 sbx 的值应该是字形的左边方向(left sidebearing)(这是 ISO/IEC 9541 EXT/MINX 与 ISO/IEC 9541/PX 之差)，而且 sby 值为 0，见 ISO/IEC 9541.3 中的 2.8.4.1。

——〈Metrics2:DictionaryReference〉:Metrics2 的值引用一个包含每一个字形信息的词典，这个词典为该字型的第一个可替换写模式，提供了可以取代字形处理过程中的默认字形定位点和字形转义值的有关信息，这个词典与 FontType 3 字型词典中 OtherMetrics 项引用的词典必须具有相同的格式。如果在这个字型中同时存在 OtherMetrics 向量，Metrics2 项的优先级比 OtherMetrics 向量中的第 0 项要高。如果字型中还有 Metrics 项，那么将参照应用 Metrics 词典信息修改过的字形信息对 Metrics2 词典进行解释。

——〈OtherMetrics:VectorReference〉:OtherMetrics 的值和 FontType 3 字型词典的类似。OtherMetrics 中的信息要比包含在字形过程中的相应信息的优先级要高，在 FontType 1 字型词典中，如果存在 OtherMetrics 项，那么 Metrics2 项也必定存在。如果 Metrics 项也在这个字型中，那么将参照应用 Metrics 词典信息修改过的字形信息对 OtherMetrics 词典进行解释。

——〈CDevProc:Procedure〉:CDevProc 的值是对一个过程的引用，该过程能够针对写模式 0 和 1 在算法上修改每个字形的转义和定位点的特性。如果 Metrics 和/或 Metrics2 词典在这个字型中也存在的话，那么只有在应用了这些词典信息以后才会应用 CDevProc。如果在字型中存在 OtherMetrics 项，那么 CDevProc 就不再出现。CDevProc 的语义如下所示：

——开头两种写模式(0 和 1)的数值字符的尺度信息将按顺序放入操作数栈：

wx0,wy0,minx,miny,maxx,maxy,wx1,wy1,px1,py1

——wx0,wy0,wx1,wy1 分别是写模式 0 和 1 的转义(ISO/IEC 9541/EX-ISO/IEC 9541/PX，等等)；

——minx,miny,maxx,maxy 分别是 ISO/IEC 9541/EXT 的各分量的值，它们是字形的范围值。

——px1,py1 是写模式 1 的定位点(ISO/IEC 9541/PX,ISO/IEC 9541/PY)的 x,y 两个分量。(默认写模式的定位点是(0,0)，通过 CDevProc 是不可修改的)。

——然后把所画字形的字形标识符放入操作数栈。

——CDevProc 过程被解释执行。它从栈中取出 11 个参数，并把语义与前十个参数类似的十个数据作为结果留在栈中。

——以后这十个数将作为相应字形的尺度属性的新值使用。

——〈WMODE:Cardinal〉:WMODE 的值和 FontType 3 字型词典的相似。

类似 FontType 3 字型词典，FontType 1 字型词典应该包含用以支持字形描述的其他项，且应该提供标识符为〈.notdef:Name〉的字形描述。字型词典应该包含所需要的任何项；描绘字形过程中用不到的项将被忽略。

16.2.1.1.4 基字型字形映射

如果当前字型是基字型，本节定义了字符文本描绘过程中实现字形映射的算法。从字母串(GlyphStrings)中衍生出的字形索引如下：

——字形串中的每个八位字节被认为是一个 0 到 255 之间的整数。

——得到的每个整数用来作为一个字形索引(Glyph Index)。

字型索引没有用处,也不能从字形串中衍生出来。

在基字型中字形索引和字形标识符之间的联系是通过 Encoding 向量来说明的。Encoding 指向一个字形标识符的向量,且这是所有的基字型词典必须包含的项,这个向量称为字形索引映射表(Glyph Index Map)。其中的每个标识符都是字形标识符,每个字形索引作为一个 Encoding 向量的索引,然后从字形索引映射表中选取字形标识符。当字形标识符决定之后,就象 16.3.13～第 17 条中指出的那样继续字形的描绘过程。

当试图通过字形索引查找字形标识符时,若字形索引大于字形索引映射表的大小(少一),则文件解释器将调用 RangeCheck 异常。

字形索引映射表中的每个字形标识符应该是该字型中的一个字形描述标识符,如果从字母索引映射表中选择了一个字形标识符,但在该字型中没有这个字形描述,那么将用字形标识符〈.notdef:Name〉来取代。

注:字型词典或编码向量中使用的字形标识符可以是任何标识符,所以,用于给定字型词典的字形索引映射表必须适合于该字型的所有字形描述,特别是,字形索引映射表中使用的标识符应该与字型词典中标识字形描述的标识符一致。

16.2.1.1.5 标准字形索引映射

本标准定义了标准的字形索引映射,其中有些映射在所有的 SPDL 表示系统中都有效,在下面各条中将讨论它们。其他的字形索引映射只用于特定的表示系统或者特殊的字型和字库对象词典。这种系统,字型或字型对象不必要与标准字形索引映射保持一致。

标准字形索引映射在 RESOURCE DECLARATIONS 中说明(见第 7 章),然后在文件内容的 DECLARATION 定义域中使用。

16.2.1.1.5.1 Latinl publishing 字形索引映射表

名字为 GlyphIndexMap/Latinl publishing 的字形索引映射在所有的 SPDL 表示系统中都是有效的,它是一个有 256 个名字的向量。每个名字的格式为 afii:xxxx,这里 xxxx 是 1 到 8 个 0～9,A～F 之间的 GB 1988 编码的字符序列,以非零数字开头,代表与每个 ISO 10036 相关的国际注册字形标识符的 16 进制表示,当然除了某些名字是特殊的名字〈.notdef〉以外。GlyphIndexMap/Latinl publishing 中名字的详细清单在附录 E 中给出。

16.2.1.1.5.2 Latinl publishing(A)字形索引映射表

本标准定义了名为 GlyphIndexMap/Latinl publishing(A)的字形索引映射表。它是一个 256 个名字的向量。

GlyphIndexMap/Latinl publishing(A)中的名字的详细说明在附录 E 中定义。

16.2.1.1.5.3 AFII 字形索引映射算法

字形索引映射族 GlyphIndexMap/AFII/nnnnnnnn 在所有的 SPDL 表示系统中都有效。每个 n 都是一个(0..9,A..F)之间的 GB 1988 字符,nnnnnnnn 是一个在 00000000 和 FFFFFF00 之间的 16 进制数。

这个族中的每个成员是一个 256 个名字的向量。每个名字的格式是 afii:xxxxxxxx,这里 xxxxxxxx 是 8 个(0..9,A..F)之间的 GB 1988 编码字符的序列,它可解释为一个数的 16 进制表示。对于这个族中的每个特定的实例 GlyphIndexMap/AFII/nnnnnnnn,每个名字 afii:xxxxxxxx,将在字形索引表中有一个十六进制值 xxxxxxxx,它相当于 nnnnnnnn(16 进制)加上索引(0 到 255 的十进制数或 00 到 FF 的十六进制数),这个索引代表了每个名字在向量中的位置。

16.2.1.2 复合字型词典

16.2.1.2.1 FontType 0 字型词典

除了〈FontType:Integer〉外,FontType 0 字型词典中必需的项有:

——〈FontMatrix:Transformation〉:这是从字型词典创建相应字型对象时使用的初始或默认的变换值。这个变换和复合字型中各个下属字型的 FontMatrix 有密切联系。

——〈Encoding:VectorReference〉:Encoding 的值指向一个整型向量,每个整型量是一个字型选择号,这是在字型映射过程中使用的将字型索引和字型选择号联系起来的字库索引映射(见16.2.1.2.2)。

——〈FDepVector:VectorReference〉:FDepVector 的值指向一个字型对象的向量。字型索引映射(Encoding 项,见上)中的字型选择号被用来在 FDepVector 中选择一个字型对象,这就是如何在字形映射时可以通过复合字型来存取各分字型。分字型可以是基字库对象,也可以是复合字型对象。

——〈FMapType:Integer〉:FMapType 的每个允许值代表了 16.2.1.2.2 中定义的一种不同的字形映射算法。其他的值在本标准中保留,以备将来之用。

根据 FMapType 的值,FontType 0 字型词典中还要求有其他项,它们在 16.2.1.2.2 中定义。

由本标准定义了语义的 FontType 0 字型词典的可选择项是:

——〈WMODE:Cardinal〉:WMODE 的值和基字型的一样。复合字型中的 WMODE 值可以取代该复合字型中所有分字型的 WMODE 值。如果复合字型中不存在 WMODE,则默认值为 0。

除了以上的项以外,字型词典中可以包含任何其他所需要的项,描绘字形时不需要的项将被忽略。

16.2.1.2.2 复合字型字形映射

如果当前字型是复合字型,本条定义了在描绘字符文本过程中进行字形映射的算法。下面定义了本条中使用的术语。

复合字型的层次结构被看作是一棵树,树的根结点是原复合字型,它是开始映射字形串时的当前字型,这个字型称为"根字型"。树的叶结点是实际包含字形形状信息的基字型,中间结点(如果有的话)称为附加复合字型。沿着从根到叶的方向移动称为树递降(ascending the tree),沿着从叶到根的方向移动称为树上溯(ascending the tree)。复合字型的任何分字型以及他们的任何级别的分字型称为下属字型(descendent font)。拥有至少一个下属字型的字型称为父字型(parent font)。

注:实现时,复合字型的嵌套深度可以有一定界限。

一般说来,字形串中的八位字节用以派生出一串字形说明符(Glyph Specifier)。这里每个字形说明符有一对(基字型,字形标识符)组成,然后就根据基字型去描绘每个字形标识符,好象该基字型就是描绘操作进程中的当前字型。

注:基字型中的过程,例如 FontType 3 字型中的 ConstructGlyph 过程或 FontType 1 字型中的 CDevProc 过程,可以用操作符 GetSelectedFont 和 GetRootFont 分别决定当前所选择的基字型对象和原来的复合字型对象。

在复合字型中,字型索引和 FDepVector 字型选择号之间的对应关系是有 Encoding 向量来说明的。Encoding 指向一个整数向量,并且它是所有的复合字型词典中必须的项,这个向量就是一个字型索引映射表。每个字型索引用来作为查找 Encoding 向量的下标,然后从字型索引映射表中选取一个字型选择号,进而再作为查找 FDepVector 的下标,以决定用来描绘字母的分字型或者进一步进行分字型的选择,如下所述:

有两类复合字型映射算法:模态的和非模态的(modal 和 non-modal)。对于非模态的映射算法,从字形串中每析取(extract)出一个字形说明符以后映射算法就重新开始;对于模态映射算法而言,算法保存分字型在层次树结构中的位置信息。具有非模态映射算法的复合字库可以是具有模态映射算法的复合字型的下属字型,但反过来并不成立。

对于非模态映射算法,决定字形说明符的算法有如下四个步骤:

1) 映射算法根据由 FMapType 值决定的算法从字形串中相继析取出字型索引和潜在的字母索引(正如下面所定义的)。

2) 字型索引用来作为查找复合字型的字型索引映射表的下标,来决定字型选择号,然后这个选择号作为 FDepVector 的下标来选择复合字型的分字型对象。

3) 如果中选字型对象是基字型,那么潜在的字形索引就是实际字形索引,它可以用来查找基字型

的字形索引映射表(见 16.2.1.1.3),以决定字形标识符,至此整个字形说明符也就确定了。

4) 如果中选字型对象本身也是一个复合字型,那么映射算法就以由中选字型对象的 FMapType 所指定的方式确定一个新的字型索引和字形索引。它将利用以前的潜在字形索引值,并且从字形串中析取更多信息。正象下面的条中所定义的那样,处理过程转到上面第 2 步继续执行。

这个过程进行到字形串中的所有八位字节均处理完毕为止。如果字形串的映射过程没有在算法的步骤 3 结束,则将引起 RangeCheck 异常。

对于模态映射算法,确定字形说明符的算法有如下三个步骤:

1) 映射算法从字形串中不断析取信息,以引导算法在分字型的树中递降或上溯,从而根据一定算法选择一个根字型的下属字型,这个算法依赖于当前中选字型的 FMapType 的值(在下面条中定义)。(在字形串的映射开始时,当前中选字型是根字型)。

2) a. 如果中选字型是一个基字型,那么字形串中的下一个八位字节被认为是关于中选基字型中的字形索引。这个过程一直进行到遇见表示选择新字型的八位字节时才结束(它是由父字型的 FMapType 值定义的特定的一个或多个八位字节),这个字节不从字形串中取出。

b. 如果中选字型是一个复合字型,而且他的映射算法是非模态的,那么字形串中的随后一个八位字节被用来析取指向中选字型的字形说明符,使用的就是上面所讲的四个步骤的非模态映射算法。这个过程一直进行到执行步骤 1)过程中,映射算法遇到表示选择新字库的八位字节时才结束(它是由父字型的 FMapType 值定义的特定的一个或多个八位字节)。这个字节不从字形串中取出。

c. 如果所选字型是一个具有模态映射算法的复合字型,那么映射过程将参照所选字库转到上面步骤 1)继续执行。

3) 如果步骤 2.a)或 2.b)结束,则映射过程将参照父字型转到上面步骤 1)继续执行。

整个过程一直进行到字形串中所有的八位字节全部处理完毕才结束。

根据 FMapType 的值从字形串中析取字形说明符信息的算法其详细说明在以下各条中定义。

16.2.1.2.2.1 8/8 映射

如果 FMapType 的值为 2,那么非模态字形映射的算法如下:

在步骤 1 中,从字形串中调入两个八位字节,第一个被认为是值在 0～255 之间的字型索引,第二个被认为是值在 0～255 之间的潜在字形索引。

在步骤 4 中,先前的潜在字形索引被认为是新的字型索引,再从字形串中调入一个八位字节,它被认为是一个值在 0～255 之间的新的潜在字形索引。

16.2.1.2.2.2 1/7 映射

如果 FMapType 的值为 4,那么非模态字形映射的算法如下:

在步骤 1 中,从字形串中调入一个八位字节,最高位被认为是值为 0 或 1 的字型索引,剩下的 7 位被认为是值在 0～127 之间的潜在字形索引。

在步骤 4 中,先前的潜在字形索引除以 128,结果舍入为小于它的最大整数,作为新的字库索引。先前的潜在字形索引的低七位被认为是一个值在 0～127 之间的新的潜在字形索引。

16.2.1.2.2.3 9/7 映射

如果 FMapType 的值为 5,那么非模态字形映射的算法如下:

在步骤 1 中,从字形串中调入两个八位字节,第一个乘以 2,再加上第二个八位字节的最高位,结果作为值在 0～511 之间的字型索引,第二个字节的余下七位被认为是值在 0～127 之间的潜在字形索引。

在步骤 4 中,再从字形串中取一个八位字节,先前的潜在字形索引乘以 2,再加上新的八位字节的最高位,结果作为值在 0～511 之间的新的字型索引,新的八位字节的余下七位被认为是值在 0～127 之间的新的潜在字形索引。

16.2.1.2.2.4 区间映射 Interval Mapping

如果 FMapType 的值为 6,那么在复合字型词典中必须有附加项:〈SubVector:OcterString〉。这个串

中的第一个八位字节被认为是0～255之间的整数，该数加1是〈sz:Integer〉，即说明符长度，它是组成说明符值的八位字节个数。

注：实现时可以有一个sz极限值。

SubVector八位字节串的余下部分定义了一串相继说明符值的范围。每个范围有sz个八位字节来定义，这些八位字节可解释为一个非零的整数，第一个八位字节为最高位，其结果就是说明符值的范围大小。所有的范围大小之和应该小于2的sz次幂，这是sz个八位字节所能表示的说明符值的总数。在这串显式范围的最后有一个内含的范围，它包含了所有可能的说明符值的剩余部分。

有了说明符值，映射算法按下述方法确定字型索引和潜在字形索引：

算法先确定这个说明符值所在范围，从零开始计数，在SubVector八位字节串中由范围所指定的位置被用来作为字型索引；说明符值减去这一范围的第一个值所得的差被认为是潜在字形索引。

如果FMapType的值为6，那么非模态字形映射的算法按如下的方法从字形串中获得说明符值：

在步骤1中，从字形串中调入sz个八位字节，它们可被认为是一个非零整数，第一个八位字节为最高有效位，其结果就是说明符值。

在步骤4中，先前的字形索引乘以256(sz-1)，然后从字形串中调入sz-1个八位字节，它们可被认为是一个非零整数，第一个八位字节为最高有效位，两者之和就是新的说明符值。

16.2.1.2.2.5 转义映射 Escape Map

如果FMapType的值为3，那么在复合字型词典中必须有附加的可选择项：〈EscChar:Cardinal〉，它的值在0～255之间，这个值是转义码，用于转义映射的映射算法。如果字型字典中没有EscChar项，默认值为255，根字型中字型对象的EscChar值，将取代各复合分字库对象中指定的EscChar值。

具有FMapType 3的字型对象只能是FMapType值为3或7的其他字型对象的分字型。

如果FMapType的值为3，那么模态字形映射的算法如下：

1) 从字形串中调入1个八位字节，它被认为是一个0到255之间的整数，如果它等于转义码，那么从字形串中再取一个八位字节，解释为0～255之间的一个整数。

如果第二个整数不等于转义码，那么就是字型索引，用它来确定字型选择号，进而找到分字型。

2) a. 如果中选字型是一个基字型，那么字形串中随后的直到转义码之前的八位字节可认为是指向中选基字型的字形索引。这个过程一直进行到遇见一个等于转义码的八位字节时才结束，这个八位字节不从字形串中取出。

b. 如果中选的分字型是一个使用8/8,1/7,9/7或区间映射算法的复合字型，那么字形串中的随后八位字节可用来产生指向被选复合分字型的字形说明符，就象上面在16.2.1.2.2.1～16.2.1.2.2.4中所定义的一样。这个过程一直进行到执行步骤1)时，从字形串中取出的第一个字形值等于转义码时才结束。这个字节以及随后的八位字节不从字形串中取出。

c. 如果所选分字型是一个具有转义映射的复合字型，那么映射过程将指向所选的复合分字型，转到上面的步骤1)继续执行。

3) 如果步骤2.a)或2.b)由于遇到一个转义码而结束时，映射过程将指向转义映射字库的父字型，转到上面的步骤1)继续执行。

4) 在步骤1)，如果第二个数等于转义码，映射过程将指向转义映射字型的父字型，转到上面的步骤1)继续执行。如果当前指向的字型没有父字型(即为根字型)，那么转义码本身就作为字型索引，用以确定字型选择号和分字型，映射过程转移到步骤1)继续执行。

5) 在步骤1)，如果第一个数不等于转义码，那么字型索引定为0，映射过程从步骤2)开始。

16.2.1.2.2.6 双转义映射 Double Escape Map

如果FMapType的值为7，那么在复合字型词典中必须有附加的可选择项：〈EscChar:Name〉。它是值在0～255之间的整数，这个值是转义码，用于双转义映射的映射算法。如果字库词典中没有EscChar项，默认值为255；根字型中字型对象的EscChar值可取代任何复合字库对象中指定的EscChar值。

一个 FMapType 的值为 7 的字型对象只能是根字型，它不可能是其他任何复合字型的分字库。

如果 FMapType 的值为 7，则模态字形映射算法除了步骤 4)以外，与 FMapType 3 的算法相同，而步骤 4)如下所述：

4) 在步骤 1)中：如果第二个整数等于转义码，那么从字形串中再取出第三个八位字节，它可解释为一个值在 0～255 之间的整数。该数加上 256，就是字型索引，它用来决定字型选择符，然后决定一个复合字型。映射算法转步骤 2)继续执行。

16.2.1.2.2.7 移出/移入映射(ShifOut/ShifIn Mapping)

如果 FMapType 的值为 8，则在复合字型词典中有两个附加项：

〈ShiftOut:Name〉和〈ShiftIn:Name〉，

它们的值在 0～255 之间。如果字型词典中没有 ShiftOut 项，可用默认值为 14；如果字型字典中没有 ShiftIn 项，可用默认值为 15。

具有 FMapType 8 的字型对象只能是根字型。它不可能是任何其他复合字型对象的分字型。

如果 FMapType 的值为 8，那么模态字形映射算法只支持两种分字型：

1) 从字形串中调入一个八位字节，它被认为是一个 0 到 255 之间的整数，如果它等于移入码，那么字型索引为 0。如果它等于移出码，那么字型索引为 1。该字型索引可用来确定字库选择号，进而找到分字型。

2) 字形串中的随后的八位字节可用来产生指向被选分字型的字形说明符，这个过程一直进行到遇到一个等于移入或移出码的八位字节时才结束。然后映射过程将转到上面的步骤 1)继续执行。

3) 在字形串的开头，如果第一个数不等于移入/移出码，那么字型索引为 0，映射过程转移到步骤 2)继续执行。

16.2.1.2.3 标准字型索引映射

本标准定义了在所有的 SPDL 表示系统中均可用的标准字型索引映射。在特定表示系统或特定字型和字型对象词典中可以使用其他字型索引映射。这种系统的字型或字型对象不需要用标准字型索引映射来统一。

标准字型索引映射可以在 RESOURCE DECLARATIONS(见第 7 章)中说明，然后可以在文件内容的 DECLARATION 的范围内使用。

16.2.1.2.3.1 字型索引映射算法

名为 FontIndexMap/Sequential/nnn 的字型索引映射族可用于所有的 SPDL 表示系统。每个 n 是在(0…9)之间的 GB 1988 字符。数字序列 nnn 代表一个≤512 的十进制数值。

这个族的每个成员都是一个有 nnn 个整数的向量。这个向量的元素是值在 0 到 nnn－1 之间的连续整数。

16.2.2 字型对象词典 FontObject Dictionary

FontObject 词典是一个语义完全由引用字型词典定义的词典。状态变量 CurrentFont 的值一定是指向字型对象词典的 DictionaryReference。因而，FontObject 词典是一个数据结构，实际上解释器正是利用它在表示媒体上描绘字形图像。

一个 FontObject 词典可以说它是对应于一个包含相同字的尺度信息、字形形状以及字形映射信息的字型词典。在表示字符文本过程中使用 FontObject 的结果可参照与它对应的字型词典，它由本条中的操作符定义。

注：本条仅指出了在 SPDL 解释器中使用 FontObject 所能得到的结果，得到这种结果的方法将依赖于具体实现。

FontObject 词典有下列性质：

——每个 FontObject 词典对应于一个字型词典。当对一个字型词典引用使用 DefinrFont 操作符时，结果就是对对应的 FontObject 词典的引用；当对一个 FontObject 词典引用使用 OpenFont 操作符时，结果就是对对应的字型词典的引用(除非 FontObject 的存取属性是只可执行的，见下)。

——所有的 FontObject 词典都有“只读”的存取属性，并且不可以修改，除非使用 16.3 中特别定义的操作符来修改 FontObject 词典。特别是，PUT 操作符不能用于 FontObject 词典，这种企图将引起 InvalidAccess 异常。

——具体实现时可以增加下面的附加限制：某些或全部 FontObject 有“只可执行”存取属性，在这种情况下，使用 OpenFont 和 Get 将会引起 IvalidAcess 异常。

注：上述限制的目的就是允许在具体实现时使用高度优化的内部格式，而这个对任何页面描述的操作都是不透明的。

——本标准中没有详细说明 FontObject 词典的内容。除了 FontObject 词典不是“只可执行”属性外，它必须包含足够的信息，以便使用 OpenFont 操作符去重新创建一个对应的字库词典。

虽然 FontObject 词典的内容没有指定，但是根据对应的字型词典用于字符文本表示的 FontObject 的语义是完全定义了的。本标准的本条定义了 FontObject 词典的语义，这些词典和字型词典在内容和性能上是一致的，而实际系统中可用另一种方法来定义 FontObject 词典的内容。

作为这种定义的方法的一部分，语句：

{fontobj/Key Get}

用来表示从 fontobj 引用的 FontObject 词典中取得信息的动作。这个信息对应于字型字典中作为名字元素 key 的值所存储的信息，即使 fontobj 可能有“只可执行”的属性，或没有名字元素为 key 的项也没有关系。同样地，语句：

{fontobj OpenFont}

用来表示取得字型词典的可修改拷贝的操作。这个字型词典对应于由 fontobj 所引用的 FontObject 词典，即使 fonyobj 属性是“只可执行的”，因而不能适用 OpenFont 操作符。

注：上面的表示方式只是为了表示方便，因为实际上操作符 Get 和 OpenFont 不能用于某些 FontObject 词典。

16.3 操作符

下面的操作符用来获得、修改和构造字型对象，设置成像变量 CurrentFont，显示字符文本等。

16.3.1 DefineFont(定义字型)

DefineFont 操作符取一个操作数：

〈FD:DictionaryReference〉

并返回一个结果：

〈fontobj:DictionaryReference〉

FD 必须指向一个有效的字型词典，返回值 fontobj 指向一个新的对应于 FD 的 FontObject 词典。

16.3.2 FindFont(查找字型)

FindFont 取一个操作数：

〈ID:identifier〉

这里的 ID 是一个 INTERNAL IDENTIFIER 的值，它与解释上下文中的 FontObject 相关。该操作符返回一个结果：

〈fontobj:DictionaryReference〉

这里的 fontobj 是与 ID 相关联的 FontObject 词典。

如果 ID 是当前解释上下文中一个有效的 INTERNAL IDENTIFIER 的值并且与一个字型对象相对应，那么解释 FindFont 的结果和解释 Find Resource 的结果是一样的。在这种情况下，{ID FindFont} 的结果等同于

{ID /FontObject FindResource}

注：用来选择 FontObject 词典的看法，最适合于没有指定字型 RESOURCE DECLARATIONS 的属性，但选择总是提供的情况。

16.3.3 GetPosition(取当前位置)

GetPosition 操作符无操作数，但返回两个结果：

〈y:Number〉

〈x:Number〉

这里的(x y SetPosition)将把成像变量 CurrentFont 的值置为当前值。

16.3.4 GetRootFont(取根字型)

GetRootFont 操作符无操作数，并返回一个结果：

〈fontobj:DictionaryReference〉

这里的 fontobj 指向成像变量 CurrentFont 的当前值所指定的 FontObject 词典(通常是由最近一次执行 SetFont 操作符所设置)。

16.3.5 GetSelectedFont(取被选字型)

GetSelectedFont 操作符无操作数，并返回值：

〈fontobj:DictionaryReference〉

这里 fontobj 指向一个 FontObject 词典，除了正在描绘字符文本外，这个值和 GetRoot Font 操作符的返回值是一致的。但是如果一个描绘字符文本的过程(例如字形处理过程)来执行 GetSelectedFont，并且 CurrentFont 的值是一个复合字型，那么它将返回一个指向当前中选基本字型词典的结果。(见 16.2.1.2.2)

16.3.6 OpenFont(打开字型)

OpenFont 取一个操作数：

〈fontobj:DictionaryReference〉

并返回一个结果：

〈FD: DictionaryReference〉

这里的 fontobj 指向一个 FontObject 词典，返回值 FD 指向一个新的，与 fontobj 对应的字库词典的一个可修改拷贝。

如果 fontobj 指向的 FontObject 词典的存取属性为“只可执行”，那么执行 OpenFont 将引起 InvalidAccess 异常。

16.3.7 PutWMODE(置写模式)

PutWMODE 操作符取两个操作数：

〈wm:integer〉

〈fontobj1:DictionaryReference〉

这里的 wm 是写模式的值，fontobj1 指向 FontObject 词典。它返回一个结果：

〈fontobj2:DictionaryReference〉

这里的 fontobj2 指向一个新的 FontObject 词典，它是下列操作的结果：

{fontobj1 OpenFont Dup 〈WMODE:Cardinal〉wm Put DefineFont}

16.3.8 ScaleFont(字型中字符的大小)

ScaleFont 操作符取两个操作数：

〈sc:Number〉

〈fontobj1:DictionaryReference〉

其中 fontobj1 指向一个 FontObject 词典，并且返回一个结果：

〈fontobj2:DictionaryReference〉

这里的 fontobj2 指向新的 FontObject 词典，它可以由下列操作实现：

{fontobj1 [sc 0 0 sc 0 0] TransformFont}

16.3.9 SetFont(置字型)

SetFont 操作符取一个操作数：

〈fontobj：DictionaryReference〉

其中 fontobj 指向一个 FontObject 词典，不返回结果。它把成像变量 CurrentFont 的值设置为 fontobj。

16.3.10 SetPosition(置当前点位置)

SetPosition 操作符取两个操作数：

〈y：Number〉

〈x：Number〉

不返回结果。它把成像变量 CurrentPosition 的值设置成当前用户坐标系中坐标为(x,y)的点。

16.3.11 SetPositionRelative(置当前点相对位置)

SetPositionRelative 操作符取两个操作数：

〈y：Number〉

〈x：Number〉

不返回结果。它把成像变量 CurrentPositiont 的值由当前用户坐标系中的坐标点(CPx,CPy)改成坐标为(CPx+X,CPy+Y)的点。

16.3.12 ShowGlyph(输出字形)

ShowGlyph 操作符的定义提供了字型结构和描绘 SPDL 字符文本之间的联系。所有其他的描绘字符文本的操作符根据 ShowGlyph 定义。

成像操作符 ShowGlyph 取一个操作数：

〈glyphid：Identifier〉

没有返回值。ShowGlyph 的执行结果依赖于成像变量 CurrentFont 的值 fontobj。如果 CurrentFont 的值是一个基字型，那么{glyphid ShowGlyph}的执行结果如下所示(用双横线"--"给出的是文本注解，而不是定义 ShowGlyph 的过程)。

```
{--首先，保存图形状态--
  SaveGraphicsState
--平移用户坐标系 UCS，从而原点在当前点--
  GetPosition Translate
--用 FontMatrix 以得到 CurrentTransformation
--将字形坐标系映射到用户坐标系 UCS--
  fontobj/FontMatrix Get Concat
--如果使用可替换的写模式需进行定位点位移--
  fontobj/WMode Get Dup 0 NotEpual
  {fontobj/OtherMetrics Get
  Exchange 1 subtract Get glyphid Get
  Dup 2 Get Negate Exchange 3 Get Negate
  Translate}
  (pop)
  ifelse
--字形成像--
  SaveGraphicsState
  fontobj PushContextStack
  glyphid/constructGlyph Getvalue Execute
  Pop ContextStack
  RestoreGraphicsState
--从 fontobj 中取出转义字的尺度信息--
```

```
fontobj /WMode Get Dup 0 Equal
{pop fontobj /Metrics Get}
{fontobj /OtherMetrics Get Exchange 1 Subtract Get}
ifElse
glyphid Get Dup 0 Get Exchange Get
--把当前点称到(Ex,Ey)--
SetPositionRelative
--恢复图形状态,不取消转义码--
RestoreGraphicsStateXCP
}
```

注：在上面定义的上下文中，假定存在 6.2.1.1.1 的注中所描述的 GlyphProcs 词典，那么推荐 ConstructGlyph 过程为：

```
{/GlyphProcs GetValue Exchange Get Execute}
```

如果当前字型是复合字型，那么 ShowGlyph 操作符将引起 InvalidFont 异常。

16.3.13 ShowString(输出字符串)

成像操作符 ShowString 取一个操作数：

〈s:GlyphString〉

没有返回值。如果 s 指出的八位字节串通过字形映射算法映射成一字形说明符序列(bf1,gid1),(bf2,gid2),…,(bfn,gidn),其中每个 bfx 是一个基本 FontObject,每个 gidx 是字形标识符，那么执行{s ShowString}的结果等同于对每个字形说明符(bfx,gidx)进行如下操作：

```
{bfx SetFont gidx ShowGlyph}
```

如果当前字型是基本 FontObject,那么所有的 bfx 将是当前字型。如果当前字型是复合 FontObject,那么使用 SetFont 只是象征性的，因为 GetRootFont 将继续返回成像变量 CurrentFont 的原始值。当然，在任何基本字型中 bfx 使用 GetSelectedFont 将返回 bfx。

16.3.14 ShowStringEscapedX

成像操作符 ShowStringEscapedX 取两个操作数：

〈v:VectorReference〉

〈s:GlyphString〉

其中 v 必须指向一个数值向量，它含有从字形串 s 中析取出来的字形说明符个数一样多的元素。该操作符没有返回值。

如果对于 x=0,1…,n

——(bfx,gidx)是从字形串中析取出来的某个字形说明符；

——〈vx:Number〉是 v 的第 x 个元素。

那么{s v ShowStringEscapedX}等同于对于 x=0,1,…,n 顺序执行以下语句：

```
{save GraphicsState
bfx SetFont gidx ShowGlyph
Restore GraphicsState
vx 0 Set PositionRelative}
```

如果当前字型是基本 FontObject,那么所有的 bfx 将是当前字型。如果当前字型是复合 FontObject,那么使用 SetFont 只是象征性的，因为 GetRootFont 将继续返回成像变量 CurrentFont 的原始值。当然，在任何基本字型中 bfx 使用 GetSelectedFont 将返回 bfx。

16.3.15 ShowStringEscapedY

成像操作符 ShowStringEscapedY 取两个操作数：

〈v:VectorReference〉

〈s:GlyphString〉

其中 v 必须指向一个数值向量，它含有从字形串 s 中析取出来的字形说明符个数一样多的元素。

该操作符没有返回值。

如果对于 x=0,1,…,n

—(bfx,gidx)是从字形串中析取出来的某个字形说明符；

—〈vx:Number〉是 v 的第 x 个元素。

那么{s v ShowStringEscapedY}等同于对于 x=0,1,…,n 顺序执行以下语句：

{save GraphicsState

bfx SetFont gidx ShowGlyph

Restore GraphicsState

0 vx SetPositionRelative}

如果当前字型是基本 FontObject，那么所有的 bfx 将是当前字型。如果当前字型是复合 FontObject，那么使用 SetFont 只是象征性的，因为 GetRootFont 将继续返回成像变量 CurrentFont 的原始值。当然，在任何基本字型中 bfx 使用 GetSelectedFont 将返回 bfx。

16.3.16 ShowStringEscapedXY

成像操作符 ShowStringEscapedXY 取两个操作数：

〈v:VectorReference〉

〈s:GlyphString〉

其中 v 必须指向一个数值向量，它含有从字形串 s 中析取出来的字形说明符个数两倍多的元素。该操作符没有返回值。

如果对于 x=0,1,…,n

—(bfx,gidx)是从字形串中析取出来的某个字形说明符；

—〈v2x:Number〉和〈v2x+1〉是 v 的第 2x,2x+1 个元素。

那么{s v ShowStringEscapedXY}等同于对于 x=0,1,…,n 顺序执行以下语句：

{save GraphicsState

bfx SetFont gidx ShowGlyph

Restore GraphicsState

v2x v2x+1 SetPositionRelative}

如果当前字型是基本 FontObject，那么所有的 bfx 将是当前字型。如果当前字型是复合 FontObject，那么使用 SetFont 只是象征性的，因为 GetRootFont 将继续返回成像变量 CurrentFont 的原始值。当然，在任何基本字型中 bfx 使用 GetSelectedFont 将返回 bfx。

16.3.17 TransformFont(变换字型)

TransformFont 操作符取两个操作数：

〈T2:Transformation〉

〈fontobj1:DictionaryReference〉

这里的 fontobj1 指向一个 FontObject 词典，且返回一个结果：

〈fontobj2: DictionaryReference〉

这里的 fontobj2 指向一个新的 Fontobject 词典。

如果 fontobj1 指向一个基字型，那么 fontobj2 指向一个新的基本字型，该字型是执行下列语句的结果：

{fontobj1 OpenFont Dup Dup/FontMatrix Get

/FontMatrix Exch T2 ConcatT Put DefineFont}

如果 fontobj1 指向一个复合字型，那么 fontobj2 指向一个新的复合字型，当利用该字型成像时，即使上述变换已作用于每一个下属的基字型，那么图像字形仍采用原来的字型。

注：无论在哪一种情况下，与变换有关的 T2 值不可能由谁来取代，但是可以通过与 T2 的联接来不断修改。

17 光栅图形操作符

17.1 光栅图形的模型

光栅图形指的是通过称为图形元素(Picture elements)或像素(pels)组成的矩形阵列来表示图形信息。这种信息可以使用两种方式中的任一种来给出：

——指出一个形状(shape)。这可通过指明组成形状内部像素的方式来实现。

——指出一个图像(image)。这可通过指明每一像素的黑白、灰度或颜色值来实现。

光栅图形的第一种类型称为位图(bitmap)的光栅图形。在 SPDL 中，位图光栅图形是通过 MaskBitMap 操作符提供的。光栅图形的第二种类型称为取样光栅图形(sampled rastergraphics)，取样光栅图形通过 ImageRasterElement 操作符提供。

两种类型的光栅图形都需要用相似的设备来获得和处理光栅图像数据，它们主要的区别在于，光栅图像数据用作点阵光栅图形时，对每一个像素给定一个布尔值，它指出此像素是否在所定义的形状之内；而光栅图像数据用作取样光栅图形时，可能包含更多的复杂的值，它们指出了每个像素的值和/或颜色。

光栅图形操作符和数据类型可以用来表示大型的光栅图形元素，而这些光栅图像数据在成像以前即使在打印机上要把它们都存储起来也是不可能的。它也提供了光栅图像数据格式上的灵活性，这样，各种类型的扫描仪可使用不同的数据压缩算法进行压缩输出。

注：重要的一点是 SPDL 指出了生成光栅图形元素的效果，而不是产生这种效果所使用的技术，真正实现时可以不必去实现在定义光栅成像操作符时所描述的所有中间步骤，这样描述的目的是为了便于将光栅图形操作的主要部分独立约束成这种成像的特殊实例，并允许对所需最终结果进行精确描述。

17.1.1 取样光栅图形 Sampled Raster Graphics

一幅图形或图像的取样光栅图形表示包括图像的很多取样点的颜色和/或值的信息，通过这些取样点可以重新构造出图像，信息的单元称为取样值(Sample Values)，一个特定图像取样值的集合称为此图像的取样数据。这样的光栅图像可由扫描仪通过测量每个取样格点存在的图形来产生，或由图形系统同步生成。

扫描一幅图形产生取样图像的过程主要包括两个步骤：

——测量图形的每一取样格点的颜色和/或值(照度—luminance 或亮度—lightness)，并且参照特定的彩色模型给出参量表示；

——将参量表示组装成压缩的数据结构，并常使用压缩技术进一步压缩此数据结构。

从取样数据重构图像需进行上述步骤的逆操作，光栅图像的重构和成像的模型包括 4 个步骤：

——解释取样数据以提供彩色元素矩形阵列或光栅网格的每一元素的取样值，每个取样值逻辑上可以认为由数值向量构成(如：由灰度值构成的单元素向量或由一组 RGB 坐标构成的三元素向量)。

——取样值被解释成表示矩形区域阵列的每个元素的颜色。

——矩形区域阵列的形状和大小在转换成所需的大小和方向时由应用程序进行修改。

——生成的图像加入到页面图像中，参照 SPDL 的成像模型，这种原始的成像操作包括：

——油墨(ink)是颜色区域的转换阵列；

——掩膜(mask)是被转换光栅区域的边界；并且

——裁剪区域是成像操作时当前的裁剪区域。

注：表示设备经常增加它们表面可见色的范围，这可以通过模拟色来实现，而这种颜色在使用其基色的离散点图案时不能直接生成。这种技术的一个普通应用是在黑白设备上用半色调技术来产生灰度等级。另一种用途是为光

栅图像生成设备产生图像数据，这些数据和用于扩充颜色的方法相适应。例如，这种设备可以产生两类光栅数据来表示灰度等级的照片图像。SPDL 光栅图形成像原语允许使用任何一种技术来表示光栅图像。

17.1.2 位图光栅图形 Bitmap Raster Graphics

位图光栅图形数据包括每一个像素的布尔值，它们指出了此像素是否在所定义的图形中。位图光栅图形数据的集合定义了用于原始成像操作的掩膜的形状，那些光栅图形数据是 0 的像素对应着掩膜的不透明部分，而那些光栅图形数据是 1 的像素则对应着要用油墨涂色的部分。

注：在上面的例子中，0 值对应掩膜中不透过的部分，而 1 对应着透过掩膜将被涂色的部分。这可以是任意的，因为掩膜的实际特性是由在 imagedictionary 操作数中的 Decode 项指出，它是 MaskBitMap 操作符的操作数。

虽然位图光栅图形数据逻辑上是两值性的，对光栅图形数据的处理所获得的最后用于定义掩膜的值，常常需要与取样光栅图像数据进行同样类型的操作。因此，SPDL 使用相同的设置来处理位图光栅图形数据和取样光栅图像数据。

17.1.3 光栅图像数据

光栅图形技术的三个重要特点是：

——用于表示颜色的多种方法；

——用于压缩取样数据的多种方法；

——表示光栅图形图像所需要的大量数据。

其中第一点只适用于取样图像，后两点同样适合于点阵光栅图形数据。

由于这些特点，我们没有理由要求所有处理光栅图形数据的系统，一定要将所有光栅数据转换成一个标准表示，以使用 SPDL 的语法和语义。因此 SPDL 定义了一个光栅图像处理的语法，它允许使用非标准和标准的颜色表示法和数据压缩方法。SPDL 定义了这些方法的标准值，它们将由每个实现版本来支持，但 SPDL 的语法也支持用于专门交换的非标准方法。

17.2 图像词典 Image Dictionaries

ImageRasterElement 和 MaskBitMap 操作符两者都取一个显式的操作数-图像词典(imageDictionary)；这个词典包括规定的内容。一个图像词典是一个对于光栅元素或掩膜点阵成像过程的自包含描述，图像词典的内容指出了下面几点：

——用于光栅图形图像的数据源；

——抽取和解释数据的方法；

——从当前用户坐标空间到成像坐标空间的映射变换。

17.2.1 图像词典的内容和语义

必选的和可选择的成像词典项有：

——〈DataSources:Vector〉是一个必选项，在单个和多个数据源的情况下 DataSource 内容所需的详细论述见 17.2.1.1。

——〈width:Cardinal〉是一个必选项，它指出了取样图像或掩膜点阵的宽度。

——〈Height:Cardinecl〉是一个必选项，它指出了取样图像或掩膜点阵的高度。

——〈BitsPerComponent:Cardinal〉是一个必选项，它标识用于表示每个颜色成分的位数。BitsPer Component 须指出怎样将光栅图形图像数据分离成从 0 到($2^{\text{BitsPer Component}}$)—1 的取样值。在分离数据时，ImageRasterElement 和 MaskBitMap 操作符按照从数据源获取数据的顺序读取八位字节数据，并从每个字节的高位开始读取位值。Bitsper Component 允许的值为 1,2,4,8 和 12。

注：如果图像词典是 MaskBitMap 操作符的操作数，则 BitsperComponent 项的值必须为 1。

——〈Decode:Vector〉是一个必选项。如果图像词典是 ImageRasterElement 操作符的操作数，则 Decode 向量指出了从取样值到相应彩色空间分量值范围映射关系。如果图像词典是 MaskBitMap 操作符的操作数，则 Decode 向量指出了掩膜的级性。关于 Decode 向量所需内容的详细论述，见 17.2.1.2 的各种性形。

——〈Interpolation:Boolean〉是可选项。如果此项未给出，则它的值默认为 false。它的值若为 true，表示将执行图像的插入，如果具体的实现可以作到这一点的话(见 17.2.1.3)。

——〈ImageTransfarm:Transformation〉是必选项。光栅图形图像有它自己的矩形图像坐标空间，其左下角是(0,0)，右上角是(width,Height)，ImageTrans form 指出从用户坐标空间到该图像坐标空间的映射。

17.2.1.1 数据源 Data Source

图像词典包含着几个有序对，它们指出了取样光栅图形图像的 width 和 Height。此取样或点阵掩膜值由光栅图形操作符在解码时获得，并被解释成一个矩形阵列，且被认为按固定的顺序获取：(0,0)到(Width-1,0)，然后从(0,1)到(Width-1,1)依次类推。按这种方式提供的八位字节数据是由图像词典中 DataSource 项指出的数据源的任务，以使得解码后能按上述的顺序得到正确的取样值。

如果成像词典中的 DataSource 项是单元素向量，则认为是指出了一个单数据源。如果当前彩色空间有一个以上的分量，则所有彩色分量数据可从数据源中按位方式顺序获取，间隔以每个取样基数。允许的数据源类型是：

——StreamObject(流对象)

——OctetString(八位字节串)

——Procedure(过程)

如果数据源是 StreamObject 类型，则必须有足够的数据以满足解码过程所需的取样数。StreamObject 在文本内容中可以由 Filter 操作符(见第 22 章)或由 FindResource 操作符(见第 13 章)建立。由 FindResource 操作符建立的 StreamObject 可以被过滤，这依赖于数据源资源与 FindResource 操作符的操作数 INTERNAL IDENTIFIER 的约束关系，见下面的论述；

——如果数据源在文件内部(见 17.2.1.1.1)，并且文件使用纯文本编码，则建立的 StreamObject 将使用 ASC Ⅱ 85Deocde 过滤器进行过滤。

——如果数据源是外部的，使用二进制或纯文本编码，或内含在一个二进制编码的文件中，则建立的 StreamObject 将不再过滤。

如果数据源是 OctetString 类型，则组成 OctetString 的八位字节被顺序提取，作为成像或点阵掩膜数据的八位字节。如果八位字节串的长度不足以提供取样所需的数目，则 OctetString 的数据被重复使用，直到取样所需的数据提取完毕。

如果数据源是 Procedure 类型，则当它执行时，必须返回一个 OctetString，包含图像或点阵掩膜数据的八位字节。如果在取样数据提取完毕之前，OctetString 中的数据已使用完毕，则 Prcedure 被重复调用，直到取样所需的数据均被提取完。Procedure 不能执行 ImageRasterElement 或 MaskBitMap 操作符。

注：此 Procedure 不能改变图像词典或图形状态的内容，进行这些操作的结果在本标准中没有定义，并且可能产生不可预见的和不确定的结果

如果图像词典中的 DataSource 项是一个多个元素的向量，则认为它指出了一个多数据源，在当前彩色空间中每个彩色分量有一个数据源，当前彩色空间中每个彩色分量的数据从不同的数据源中提取。Datasource 向量的所有元素必须为同一类型，并且向量中数据源的顺序必须与当前彩色空间的彩色分量相一致。

在多数据源的情形下，每个数据源必须适合上述单数据源情况下的要求，另外还要加上下面的几条：

——如果数据源是 StreamObject 类型，则它们必须为不同的 StreamObject。

——如果数据源是 Octetstring 类型，则它们的长度必须相等。

——如果数据源是 Procedure 类型，则它们每个在被顺序调用时，必须返回相同长度的 Octetstring 类型。

17.2.1.1.1 内含数据 In-Line Data

SPDL 提供了可过 DataBlock 内含标记类型在文件中包括光栅图形图像数据，这一功能称为内含数据的设置。这种标记类型，其值是一系列的八位字节，被定义成如第 25 章中给出的二进制和纯文本文件。在二进制文件中，这一系列的八位字节表示二进制光栅图形数据，在纯文本文件中，这一系列的八位字节表示 ASC Ⅱ 85-编码的光栅图形数据(见第 22 章)。

内含光栅图形图像数据应如下使用：

——在结构中，将(由本标准)预定义的内含数据源，Datasource/Document 通过 RESOURCE DECLARATION 结构元素与一个 INTERNAL IDENTFIER 的值相连编。

——在 RESOURCE DECLARATION 范围内的文件中，通过执行 Findresource 操作符，并使用 INTERNAL IDENTIFIER 的值作为操作数来获得一个 StreamObject。

——构造一个图像词典，如果合适的话，使用 StreamObject 作为 DataSource 向量的元素(或其中之一)。

——构造一个 TOKENSEQUENCE 或一系列的 TOKENSEQUENCE，其中包含一个将内含数据作为数据源的 ImageRasterElement 或 MaskBitMap 光栅图形图像操作符。

——紧接着解释 ImageRasterElement 或 MaskBitMap，操作符提供成像操作所需的全部数据，以 DataBlock 标记格式给出，后面紧跟着 0 个或多个连续的 DataBlockContinuation 标记。

内含数据应放在文件内容中，为的是在实际解释操作符时，可由光栅图形图像操作符存取这些数据。因此，如果光栅图形图像操作符在由循环操作符构成的过程中出现时(例如，For 操作符)，则循环中所有解释光栅图形图像操作符的内含数据必须紧跟着循环操作符，采用一个或多个 DataBlock 标记序列和零个或多个 DataBlockContinuation 标记格式。序列的顺序应与循环中光栅图形图像操作符的解释顺序一致，并且解释每个光栅图形图像操作符的数据应该是由 DataBlock 标记开始的一个新的序列。

在文件内容中遇到的 DataBlock 或 DataBlockContinuation 标记，如果不是成像操作符必须的数据源，则将引起 SyntaxError 异常。

17.2.1.2 译码向量 Decode Vector

在一个作为 ImageRasterElement 操作符的操作数的成像词典中，图像词典的 BitsPerCompenent 项指出了怎样将光栅图形图像数据分解成取样值。Decode 向量指出了从取样值到当前彩色空间的每个彩色分量相应值范围的一个线性映射关系，对于每个彩色分量，映射的最小和最大值可被指定，并且线性映射将是：

$$OutputValue = MIN + InputValue * (MAX - MIN)/N$$

其中：——OutputValue＝将作为最终彩色分量的映射值；

——InputValue＝在分解过程中从数据源提取的取样值；

——$N = (2^{BitsPerComponent}) - 1$；并且

——MIN 和 MAX 是 Decode 向量中指出的参数。

Decode 向量中的数被成对解释，每对对应于当前彩色空间的一个分量，因此，Decode 向量的长度必须为当前彩色空间的彩色分量数目的两倍。并且 Decode 向量中的 MIN/MAX 对的顺序必须与当前彩色空间的彩色分量相匹配。

注：如果某个 OutputValue 超出彩色分量所允许的范围，则 OutputValue 将被剪裁成最接近的允许值。

在作为 MaskBitMap 操作符的操作数的图像词典中，Decode 向量是一个两个元素的向量，它指出了掩膜的级性，在这种情形下 Decode 向量允许的两个值及它们的意义如下：

——〔0,1〕取样值为 0 的点则涂以当前色；取样值为 1 的点则对应于掩膜中不透明的部分。

——〔1,0〕取样值为 1 的点则涂以当前色，取样值为 0 的点则对应于掩膜中不透明的部分。

17.2.1.3 内插 interpolation

当源图像的分辨率比表示设备的分辨率低得多的时候，每个源取样点将占据了多个设备像素，这会导致很糟的输出，这种不良视觉效果可以通过在相邻取样值之间内插值来改善，图像词典中内插项的值

为 true，就表示实现时希望内插，不同设备上的内插算法的特性和实现超出了本标准的范围。

17.3 操作符

这些操作符及它们语义的说明包括了解释这些操作符时可能引起异常的条件的说明，内容异常和异常处理在第 24 章中定义。除了这些操作符特定异常外，还有一般性的异常，它们几乎在解释所有操作符时都可能引起，一般性异常及它们的语义在 24.6.2 中论述。

17.3.1 ImageRasterElement

ImageRasterElement 成像操作符接受一个显式的操作数：

〈imagedictionary:Dictionary〉

不返回结果。ImageRasterElement 操作符使用 ImageDictionary 中的信息来获得取样值，解释它们，并使用当前彩色空间生成图形图像。图像词典的内容和语义的详细论述见 17.2

17.3.2 MaskBitMap

MaskBitMap 成像操作符接受一个显式操作数：

〈imagedictionary:Dictionary〉

不返回结果，MaskBitMap 操作符使用 ImageDictionary 中的信息来获取点阵掩膜值，解释它们以产生掩膜，并且使用当前彩色和掩膜执行基本的成像操作，图像词典的内容和语义的详细论述见 17.2。

18 几何图形操作符

18.1 几何图形模型 Model for Geometric Graphics

形状被几何地定义成路径。路径由路径段构成，路径段又由路径元素构成。

一个路径元素是一个点或一条有向直线或曲线，每个路径元素在概念上有一个起点和一个终点。一系列的路径元素连结起来，除了第一个以外的每个路径段起点都与前一个路径段的终点一致，一个路径段包括一个或多个相连的路径元素，其第一个是由 BeginPathSegment 操作符建立。一个路径包括零个或多个路径段，空路径(Null Path)是没有路径段的路径。

一个封闭的路径(closed path)段是一个其最后路径元素的终点与第一个路径元素的起点相一致的路径段。一个封闭的路径包括封闭的路径段。

路径由基本操作符构造并且保持在成像变量 CurrentPath 中。一个路径段由 BeginPathSegment 开始，且可以由操作符 LineTo 和 CurvrTo 来扩展，CurrentPath 可以由 GetPath 来保存和用 SetPth 来恢复，并且可由 NewPath 或由成像和剪裁操作符将其清成空路径。

路径只在把它们用于指出一个掩膜或裁剪区域时才有几何解释，此时，定义路径的点被解释成用户坐标系中的几何形状。成像操作符 StrokePath、FillPath 和 FillPathEvenodd 以及裁剪操作符 ClipPath 和 ClipPathEvenodd，应用 CurrentTransformation 将此几何形状转换成引用坐标系统中的图形，执行这些操作符结果的几何形状的详细论述见操作符的定义。

18.2 路径的构造和处理操作符

本条论述用于构造和直接处理路径的操作符。所有的路径构造操作符在影响 CurrentPath 成像变量值的同时，也隐含地将 CurrentPosition 成像变量的值置成当前路径的终点。

这些操作符及它们语义的说明包括了对解释这些操作符可能引起异常的情况的说明，内容异常及异常的处理在第 24 章中定义。除了这些操作符特定异常以外，还有一些一般性的异常，它们在解释所有操作符时几乎都可能引起，一般性异常及它们的语义在 24.6.2 中论述。

18.2.1 AppendPath(添加路径)

AppendPath 操作符取一个操作数：

〈path:Path〉

不返回值，它把其操作数描述的路径添加到 CurrentPath 成像变量中。

18.2.2 ArcToClockWise(顺时针画弧)

ArcToClockWise 操作符取五个操作数：

〈t_2:Number〉

〈t_1:Number〉

〈r:Number〉

〈y:Number〉

〈x:Number〉

不返回结果，它建立一个包括有向圆弧的新路径段。其中：

——圆心为(x,y)；

——半径为(r)；

——起点的极坐标，相对于(x,y)是(r,〈t_1〉)；

——终点的极坐标，相对于(x,y)是(r,〈t_2〉)；

——方向为顺时针。

如果 CurrentPath 是空路径，则 ArcToClockwise 开始一个新的路径段，包含一个新路径元素的起点，紧接着是新的路径元素。

如果 CurrentPath 不是空路径，而且如果新路径段的起点与上次 CurrentPath 最后段的终点一致，则此路径元素被添加到路径段中。否则，如果新路径元素的起点与 CurrentPath 的最后段的终点不一至，则 ArcToClockwise 将一个线段添加到 CurrentPath 中，线段的起点是 CurrentPath 最后段的终点，线段的终点是新路径元素的起点，然后将新的路径元素添加到此线段之后。

18.2.3 ArcToCounterClockwise(逆时针画弧)

逆时钟方向画弧 ArcToCounterClockwise 操作符有与 ArcToClockwise 一样的语义(见 18.2.8)，但是它逆时针画一个圆弧。同样地，ArcToCounterClockwise 操作符取五个操作数：

〈t_2:Number〉

〈t_2:Number〉

〈r:Number〉

〈y:Number〉

〈x:Number〉

不返回结果，它建立一个包含有向圆弧的新路径元素，其中：

——圆心为(x,y)；

——半径为(r)；

——起点的极坐标，相对应于(x,y)是(r,〈t_1〉)；

——终点的极坐标，相对应于(x,y)是(r,〈t_2〉)；

——方向是逆时针。

如果 CurrentPath 是空路径，则 ArcToCounterColockwise 开始一个新的路径段，包含一个新路径元素的起点，紧接着是新的路径元素。

如果 CurrentPath 不是空路径，并且如果新路径的起点与 CurrentPath 的最后段的终点相一致，则此路径元素被添加到路径段中。否则，如果新路径元素的起点与 CurrentPath 最后段的终点不一致，则 ArcToCounterClockwise 将一个线段添加到 CurrentPath 中，线段的起点是 CurrentPath 最后段的终点，线段的终点为新路径元素的起点，然后将新的路径元素添加到线段之后。

18.2.4 BeginPathSegment(开始路径段)

BeginPathSegment 操作符取两个操作数：

〈y:Number〉

〈x:Number〉

不返回结果，它将一个新的路径段的起点添加到CurrentPath成像变量中，该起点为UCS坐标(x,y)的点。

18.2.5 BeginPathSegmentRelative(相对开始路径段)

BeginPathSegmentRelative 操作符取两个操作数:

〈y:Number〉

〈x:Number〉

不返回结果,如果(cpsx,cpsy)是 UCS 中的点,对应于当前路径的最后段的终点,则 BeginPathSegmentRelative 将一个新的路径段加到 CurrentPath 中,其起点为 UCS 坐标(cpsx+x,cpsy+y)的点。

如果 CurrentPath 是空路径,则引起 NoCurrentPoint 异常。

18.2.6 ClosePathSegment(闭合路径段)

操作符不取操作数,也不返回结果。如果当前路径段已经是封闭的,或如果 CurrentPath 是空路径,则 ClosePathSegment 不起作用。否则,如果(cpsb x,cpsb y)是对应于当前路径最后段起点的 UCS 坐标,则 ClosePathSegment 等价于(cpsb x,cpsb y LineTo)。除非它封闭当前路径段,在 ClosePathSegment 操作之后立即添加另一个段到 CurrentPath,从而开始一个新的路径段。

18.2.7 CurveTo(画曲线)

CurveTo 操作符取六个操作数:

〈y_3:Number〉

〈x_3:Number〉

〈y_2:Number〉

〈x_2:Number〉

〈y_1:Number〉

〈x_1:Number〉

不返回结果,它将一个包含 Bezier 曲线的新路径元素添加到 CurrentPath 成像变量的最后路径段中。此元素的起点是 CurrentPath 最后段的终点,并且新的路径元素在它的起点处和从最后路径段的终点到(x_1,y_1)的线段相切。路径元素的终点 UCS 坐标为(x_3,y_3),且路径元素与(x_2,y_2)到(x_3,y_3)的线段在最后点处相切。此元素包括两个 Bezier 控制点,它们的 UCS 坐标为(x_1,y_1)和(x_2,y_2)。

更确切地说,如果 CurrentPath 的最后路径段的终点为(x_0,y_0),新路径段包括的点应该在由等式表示的参数线上:

$$x=a_xt^3+b_xt^2+c_xt+d_x$$

$$x=a_yt^3+b_yt^2+c_yt+d_y$$

对于 $0\leqslant t\leqslant 1$,其中

$$a_x=x_3-3x_2+3x_1-x_0$$

$$a_y=y_3-3y_2+3y_1-y_0$$

$$b_x=3x_2-6x_1+3x_0$$

$$b_y=3y_2-6y_1+3y_0$$

$$C_x=3x_1-3x_0$$

$$C_y=3y_1-3y_0$$

$$d_x=x_0$$

$$d_y=y_0$$

如果 CurrentPath 是空路径,则引起 NoCurrentPoint 异常。

18.2.8 CurveToRelative(按相对坐标画曲线)

CurveToRelative 操作符取六个操作数:

〈y_3:Number〉

〈x_3:Number〉

〈y_2:Number〉

〈x_2:Number〉

〈y_1:Number〉

〈x_1:Number〉

不返回结果。若(cpsx,cpsy)是当前路径最后段的终点的 UCS 坐标,则 CurveToRelative 将一个包含 Bezier 曲线的路径元素加到 CurrentPath 成像变量的最后路径段中,此元素的起点是 CurrentPath 最后路径段的终点,且新路径段在它的起点与最后路径段的终点到(cpsx+x_1,cpsy+y_1)的线段相切,此路径元素的终点为 UCS 坐标(cpsx+x_3,cpsy+y_3),且路径元素在终点与的线段(cpsx+x_2,cpsy+y_2)到(cpsx+x_3,cpsx+y_3)相切。此元素包括两个 Bezier 控制点,其 UCS 坐标分别为(cpsx+x_2,cpsy+y_2)和(cpsx+x_3,cpsy+y_3)。此曲线的等式在 18.2.10 中给出。

如果 CurrentPath 是空路径,将引起 NoCurrentPoint 异常。

18.2.9 GetPath(取路径)

GetPath 操作符无操作数,返回一个结果:

〈Path:Path〉

其中 Path 是 CurrentPath 成像变量的值,GetPath 不影响 CurrentPath 的值。

18.2.10 GlyphToPath(沿路径输出字形)

GlyphToPath 操作符取两个操作数

〈fill:Boolean〉

〈glyphstr:Glyphstring〉

不返回结果。GlyphToPath 将一组路径段添加到 CurrentPath 中,这些路径段描述了字符的形状,就象它们由 ShowString 操作符表示一样(见 16 章)。如果 fill 的值为 true,则路径段构造成适合于用 FillPath 或 ClipPath 操作符之一来填充或裁剪,如果 fill 的值为 false,则路径段构造成适合于用 StrokePath 操作符来描画。

注:将 StrokePath 操作符作用于适合填充的路径,或将 FillPath 或 ClipPath 操作符作用于适合描画的路径,其结果在本标准中没有定义,且可能引起异常。

如果 CurrentPath 为空路径,则引起 NoCurrentPoint 异常。

18.2.11 LineTo(画线)

LineTo 操作符取两个操作数

〈y:Number〉

〈x:Number〉

不返回结果。它将添加由一条直线段组成的路径元素到 CurrentPath 成像变量中去,其起点是 CurrentPath 最后路径段的终点,其终点的 UCS 坐标为(x,y)。

如果 CurrentPath 是空路径,则将引起 NoCurrentPoint 异常。

18.2.12 LineToRelative(按相对坐标画线)

LineToRelative 操作符取两个操作数

〈y:Number〉

〈x:Number〉

不返回结果。若(cpsx,cpsy)是当前路径最后段的终点的 UCS 坐标,则 LineToRelative 将一新路径元素加到 CurrentPath 中,新路径元素包含一条直线段,其起点是 CurrentPath 最后路径段的终点,终点的 UCS 坐标为(cpsx+x,cpsy+y)。

如果 CurrentPath 是空路径,则将引起 NoCurrentPoint 异常。

18.2.13 NewPath(置路径为空)

NewPath 操作符无操作数,也不返回结果,它将成像变量 CurrentPath 置成空路径,而且使得 CurrentPosition 的值变成未定义。

18.2.14 SetPath

SetPath 操作符取一个操作数。

〈path:Path〉

不返回结果，它将 CurrentPath 成像变量的值置成 Path。

18.3 描画和填充操作符

本条论述用于描画和填充由路径描述的区域的操作符。这些操作符及它们语义的说明包括了解释它们可能引起内容异常的条件的说明，内容异常及异常处理在第 24 章中定义。除了这些操作符特定异常外，还有一些一般性的异常，它们几乎在解释所有操作符时都可能引起，这些一般性异常及它们的语义在 24.6.2 中论述。

18.3.1 FillPath(填充路径)

FillPath 操作符不接受操作数，也不返回结果，它使用 CurrentPath 成像变量定义的掩膜执行成像操作，并将 CurrentPath 置成空路径。

在执行成像操作过程中，FillPath 隐含地封闭所有断开于放的路径段，并且定义一个掩膜，它包括了由非零环绕规则决定的结果形状的内部和边界，然后用这个掩膜以及 CurrentColour 和 CurrentClipRegion 将一个图像元素加入当前页面图像。

当 CurrentPath 只包含一条直线段时，执行 FillPath 操作符的效果。等价于{0, SetStrokeWidth StrokePath}。

18.3.2 FillPathEvenOdd(奇遇规则填充路径)

FillPathEvenOdd 操作符不接受操作数，也不返回结果。它使用 CurrentPath 成像变量定义的掩膜执行成像操作，并将 CurrentPath 置成空路径。

在执行成像操作过程中，FillPathEvenOdd 隐含地封闭所有断开的路径段，并定义一个掩膜它包括了奇偶环绕规则决定的结果形状的内部和边界，然后用这个掩膜以及 CurrentColour 和 CurrentClipRegion 将一个图像元素加入当前页面图像。

当 CurrentPath 只包含一条直线段时，执行 FillPathEvenOdd 等价于{0, SetStrokWidth StrokePath}。

18.3.3 StrokePath(画路径)

StrokePath 操作符不接受操作数，也不返回结果，它使用后面将定义的 CurrentPath、CurrentDashPattern、CurrentMitreLimit、CurrentStrokeAdjust、CurrentStrokeWidth、CurrentStrokeJoin 和 CurrentStrokeEnd 等操作符定义的掩膜执行成像操作，并将 CurrentPath 置成空路径。

在执行成像操作过程中，StrokePath 在用户坐标空间中构造一个掩膜，用 CurrentTransformation 将其转换到引用坐标空间，并使用此掩膜、CurrentColor 和 CurrentClipRegion 执行一个基本的成像操作。在执行这些功能时，StrokePath 将：

——用 CurrentTransformation 的逆矩阵将 CurrentPath 转换到用户坐标空间；

——用下面叙述的方法在用户坐标空间构造图形；

——使用 CurrentTransformation 将此图形转换到引用坐标空间；

——执行一次基本的成像操作，使用：

——掩膜，其内部和边界是上述图形；

——CurrentColor(当前色)；

——CurrentClipRegion(当前裁剪区域)。

在第二步中构造的图形其每个路径段可能是实线或虚线，这由 CurrentDashPattern 的值决定，如果 CurrentDashPattern 的向量元素是空向量，则图形将如下构造：

——将每一个路径段扩展成统一的宽度 CurrentStrokeWidth，使得结果的宽线被原始路径段纵向地一分为二；

——用 CurrentStrokeEnd 指出的端点特性(如图 18-1 所示)作用于每一路径段;

——用 CurrentStrokeJoin 指出的线交特性(如图 18-2 所示)作用于每一路径段中的交点。并且可能时用 CurrentMiterLimit 修改(见 18.3.3.1)。

如果 CurrentDashPattern 的向量元素非空,则在用户坐标空间里构造一个新的路径,并由上述路径构造一个图形。

如果 CurrentStrokeWidth 成像变量的值为 0,则实际的描画宽度是表示设备可能产生的最小线宽。如果 CurrentStrokeAdjust 成像变量的值为 true,则实际的描画宽度可能与 CurrentStrokeWidth 不同,见 12 章中关于 CurrentStrokeAdjust 的描述。

如果 CurrentDashPatten 成像变量的值为:

〈V:Vector〉〈X:Number〉

则第一步中将 CurrentPath 转换到用户坐标空间的所得到的新路径将如下生成:

——向量 V 可被解释成沿 CurrentPath 向用户坐标空间的转换的一系列长度值,这些长度轮流地指出了虚线的长度及虚线间空白的长度。V 的内容可以循环使用,需要多少个循环就使用多少次值。

——数 X 可被解释成循环使用 V 时指出的一系列长度值的偏移量。在描画一路径以前,V 的元素循环地相加(虚线长度及空白长度),直到用完了总长度 X,生成虚线的长度是从循环处理 V 的后继长度中减去偏移量 X 后剩下的元素。

——新的路径包括一系列路径段,它们是从 CurrentPath 到用户坐标空间转换过程中通过 CurrentDashPattern 得到的虚线元素那些部分。

如果 CurrentPath 向用户坐标空间的转换结果不包括距离路径起点的偏移量,则称路径有零长度。描画一个零长度的路径不影响页面图像,因此,程序{NewPath x y BeginPathSegment Strokepath}在当前页面上不产生痕迹。

18.3.3.1 线交 CurrentMitreLimit 的效果

在任何给定的角点上(两条不相切的路径元素的交点),斜接(Miter)长度是所画线(由 StrokePath 操作符在用户坐标空间中构造)的内缘交点到所画线的外缘交点之间的距离,这距离随着路径元素之间的角度减小而增大。

当 CurrentStrokeJoin 成像变量的值指出了 StrokePath 操作符线交的斜接方式时,CurrentMitreLimit 成像变量值用来防止接的距离超过给定的最大长度。当线交的斜接长度与实际线宽的比例超过 CurrentMitreLimit 时,则 StrokePath 操作符构造交线规划线,就象 CurrentStrokeJoin 成像变量的值是平线交而不是斜交线,斜接长度与画线宽度的比例的公式是:

mitre length/stroke width=1/sin(angle/2)其中 angle 是两条路径元素的交线的角度。

18.4 成像变量操作符 Imaging Varible Operators

本条论述用于获取和改变成像变量值的操作符,它们控制画线和填充操作的各种参数。

18.4.1 GetDashPattern(取线型)

GetDashPattern 操作符无操作数,返回两个结果:

〈offset:Number〉

〈pattern:Vactor of Number〉

其中两个结果合起来构成了图形状态中 CurrentDashPattern 成像状态变量的当前值,该成像变量的语义在第 12 章中给出。

18.4.2 GetMitreLimit(取斜接的限定值)

GetMitreLimit 无操作数,返回一个结果:

〈limit:Number〉

其中 limit 是图形状态中 CurrentMitreLimit 成像状态变量的值,此成像变量的语义描述见第 12 章。

18.4.3 GetStrokeAdjust(取画线调整)

GetStrokeAdjust 操作符无操作数，返回一个结果：

〈adjust：Boolean〉

其中 adjust 是图形状态中 CurrentStrokeAdjust 成像状态变量的值，此成像变量的语义描述见第 12 章。

18.4.4 GetStrokeEnd(取当前线端)

GetStrokeEnd 操作符无操作数，返回一个结果：

〈end：Number〉

其中 end 是图形状态中 CurrentStokeEnd 成像状态变量的值，此成像变量的语义描述见第 12 章。

18.4.5 GetStokeJoin(取当前线交方式)

GetStrokeJoin 操作符无操作数，返回一个结果：

〈Join：Number〉

其中 jojn 是图形状态中 CurrentStrokeJoin 成像状态变量的值。此成像变量的语义描述见第 12 章。

18.4.6 GetSrtokeWidth(取当前线宽)

GetStrokeWidth 操作符无操作数，返回一个结果：

〈width：Number〉

其中 Width 是图形状态中 CurrentStokeWidth 成像状态变量的值，此成像变量的语义描述见第 12 章。

18.4.7 SetDashPattern(置线型)

SetDashPattern 操作符取两个操作数：

〈offset：Number〉

〈Pattern：Vector of Number〉

不返回结果。它将 CurrentDashPattern 成像变量设置成〈pattern〉〈offset〉，CurrentDashPattern 的作用由 StrokePath 操作符定义。

18.4.8 SetMitreLimit(置斜接限定值)

SetMitreLimit 操作符提取一个操作数：

〈x：Number〉

不返回结果。它将 CurrentMitreLimit 成像变量的值设置成 x，其中 x 的值必须大于或等于 1，CurrentMitreLimit 的作用在 18.3.3.1 中定义。

18.4.9 SetStrokeAdjust(置画线调整)

SetStrokeAdjust 操作符取一个操作数：

〈adjust：Boolean〉

不返回结果。执行 SetStrokeAdjust 操作符的结果是将 CurrentStrokeAdjust 成像状态变量的值设置成 adjust 值，此成像变量的语义描述见第 12 章。

18.4.10 SetStrokeEnd(置线端)

SetStrokeEnd 操作符取一个操作数

〈n：cardinal〉

其中 n=0，1 或 2，不返回结果，它将 CurrentStrokeEnd 成像变量设置成下面三者之一：

0——选择平的(butt)线端。在路径的端点为方形线，路径中没有突出的部分。

1——选择圆的(round)线端。在端点画一个直径与线宽相等的圆，并将其填充。

2——选择突出的方形(square)线端。端点为突出一半线宽的方形线端。

由 StrokePath 生成掩膜的 CurrentStrokeEnd 的效果在图 18-1 中给出：

18.4.11 SetStrokeJoin(置线交方式)

SetStrokeJoin 操作符取一个操作数：

〈n：Cardinal〉

其中 n=0,1,或 2,不返回结果,它将 CurrentStrokeJoin 成像变量设置成下面三者之一:

0——选择斜接线交,线的外缘被扩展,直到相交。

1——选择圆形线交,在线段的交点处画一个直径等于线宽的圆,并将其填充。

2——选择平线交,相交的路径段由 butt 线端结束,结果的缺口用一个三角形填充。

在由 StokePath 生成的掩膜中,CurrentStrokeJoin 的效果见图 18-2。

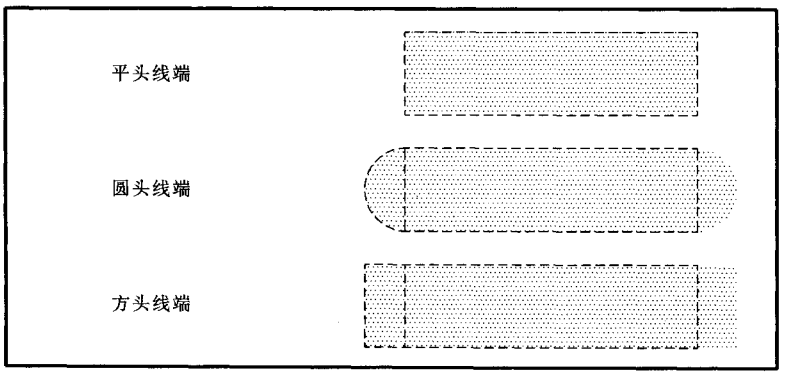


图 18-1 线端特性

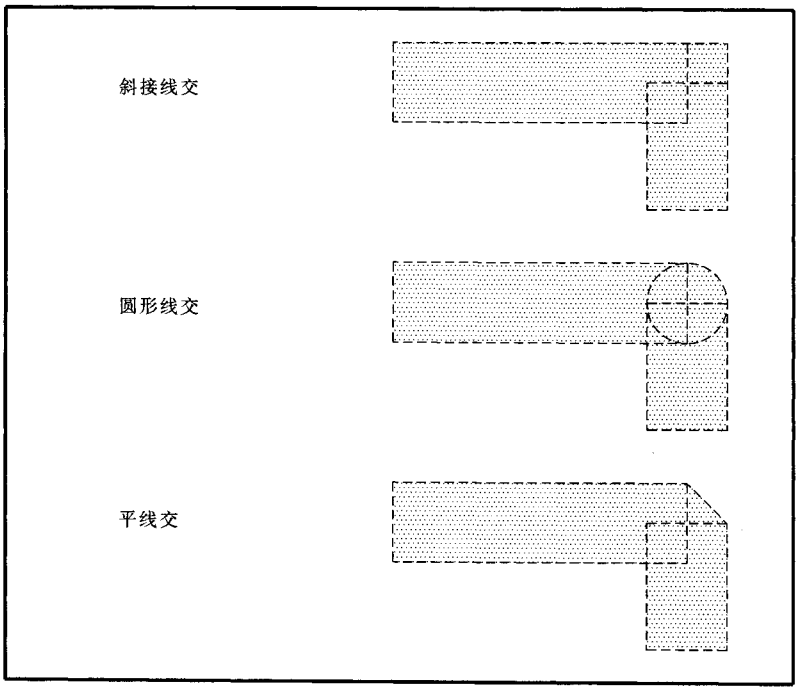


图 18-2 交特性

18.4.12 SetStrokeWidth(置线宽)

SetStrokeWidth 操作符取一个操作数:

〈x:Number〉

不返回结果,它将 CurrentStrokeWidth 成像变量的值设置成 x。CurrentStrokeWidth 的作用在 StrokePath 操作符中定义。

19 裁剪操作符

19.1 裁剪模型 Model for Clipping

成像操作符可以影响的页面图像区域称为裁剪区域。CurrentClipRegion 成像变量表示有效的裁剪区域,它由引用坐标系统表示。在页面的开始时,CurrentClipRegion 被设置成媒体的整个可成像的区域。

改变 CurrentClipRegion 的操作符可以通过 CurrentPath 派生出来的区域与其求交来实现。

19.2 操作符

这些操作符及它们语义的说明包括解释这些操作符可能引起内容异常的条件的说明。内容异常及异常处理在第 24 章中定义，除了这些操作符的特定异常外，还有一些一般性的异常，它们可能几乎在解释所有操作符时引起，这些一般性的异常及它们的语义在 24.6.2 中论述。

19.2.1 ClipPath(裁剪路径)

ClipPath 操作符无操作数出不返回结果。它使用 CurrentPath 成像变量来修改 CurrentClipRegion 成像变量，但不改变 CurrentPath 的值。

ClipPath 隐含地封闭所有开放的 CurrentPath 的子路径，并用非零变量规则判定结果形状的内部，然后将 CurrentClipRegion 成像变量设置成当前 CurrentClipRegion 和 CurrentPath 所决定的形状内部的相交部分。如果 CurrentPath 是空路径，则结果 CurrentClipRegion 的值是依赖于具体实现的非空的路径，它包含零区域。

19.2.2 ClipPathEvenOdd(奇遇规则裁剪路径)

ClipPathEvenOdd 操作符无操作操作数也不返回结果。它用 CurrentPath 成像变量来修改 Current-ClipRegion 成像数量的值，但不改变 CurrentPath 的值。

ClipPathEvenOdd 隐含地封闭所有开放的 CurrentPath 的子路径，并用奇/偶规则判定结果形状的内部。然后将 CurrentClipRegion 成像变量设置成当前 CurrentClipRegion 和 CurrentPath 所决定的形状内部的相交部分。如果 CurrentPath 是空路径，则结果 CurrentClipRegion 的值是依赖于具体实现的包含零区域的非空路径。

20 图案

当 FillPath、StrokePath 和 ShowString 等操作符用当前色画页面区域时，通常使用一种简单的颜色去覆盖区域，有时需要应用一种逻辑的墨水，它由重复图形而不是由简单的颜色组成，这样的重复图形称为图案(pattern)，本章论述了图案的建立和使用。

20.1 图案的建立和使用

用图案涂色意味着复制(或盖瓦片)一个小的图形(称为图案单元)，在 X 和 Y 方向上以固定间隔覆盖整个涂色区域。图案单元的式样可由任意的过程来定义，它可以包括，如填充区域、文字和取样图像等图形元素。图案单元的形状不一定是矩形。而且覆盖的间隔也可以不等于图案单元的大小。图案可以用来建立各种各样的图形纹理，如编织物、砖墙以及类似的几何模版。

注：虽然文件一般不能存取或知道任一给定实现设备的像素空间、图案的覆盖技术及相关的控制需要在设备像素空间中进行引用。

用图案进行涂色需要 4 个步骤：

——描述图案。这可以通过建立一个指定内容的特殊用途的词典来完成，它称为图案对象，包括图案的有关信息。词典中关键的一项是 PaintProc，执行该过程可绘出单个图案单元。这个词典可通过 RE-SOURCEDEFINITION 来构造，并通过 FindResource 操作符提供给文本。

——图案的实例化。MakePattern 操作符拷贝一个图案对象，并产生图案的一个实例(以一个只读图案词典的方式)，它被锁定于当前引用空间。换句话说，图案单元的大小和覆盖的相位与设备像素的关系由执行 MakePattern 时的 CurrentTransformation 来决定。

图纹本身在以后图形状态改变时不受影响。

——选择图案作为当前色。SetPatternColour 操作符将图案作为当前色。

——调用涂色操作符，如 FillPath，StrokePath，MaskBitMap 或 ShowString。通常以前用简单色涂色的所有区域将被图案单元覆盖。为了要得到这样的结果，将执行图案词典中的 PaintProc 过程，以得到图案单元，然后按需要的次数在当前页面上覆盖图案。具体实现时，希望使用一个高速缓存来保存图案单

元，以优化执行。

20.2 图案词典 Pattern Dictionary

图案词典中定义了：

——建立的图案类型；

——图案单元的过程；

——图案单元的裁剪；

——图案单元的覆盖控制。

图案单元在它自己的坐标系统-图案坐标系统中定义，它是在执行 MakePattern 时，由 MakePattern 操作数和 CurrentTransformation 共同定义的，覆盖和裁剪参数在图案坐标系统中解释，描绘图案单元的过程也在此坐标系统中执行。

下面的几条描述了图案词典中的必选项。图案词典还可以包含其他 PaintProc 所需的信息，除了 Implementation 以外的所有项均可以出现在图案对象中(MakePattern 操作符的操作数)。

20.2.1 PaintType(涂色类型)

必选项〈Painttype:Cardinal〉决定了怎样来指定图案单元的颜色。PaintType 的有效值有：

1) 彩色图案。PaintProc 本身指定了彩色空间和用于描绘图案单元的颜色，PaintProc 在执行任何成像操作以前必须选择一个颜色。

2) 掩膜图案。PaintProc 不指出任何彩色空间或颜色信息，而是在每次使用图案时将整个图案单元涂以指定的独立的颜色。在效果上，即 PaintProc 描述了一个掩膜，当前颜色的喷涂是透过它进行的。PaintProc 不能再执行任何颜色操作符(这包括第 15 章中的所有操作符)，如 SetTransfer 操作符、SetPatternColour 操作符或 Imageraster-Element 操作符等。使用 MaskBitMap 操作符是允许的，因为它不指定任何彩色信息。

20.2.2 PaintProc(涂色过程)

必选项〈PaintProc:Procedure〉指出了用于描绘图案单元的过程，PaintProc 取一个操作数：

〈PatternDict:Dictionary〉

其中 PatternDict 是包含 PaintProc 的图案词典。不返回结果，执行 PaintProc 的效果是画一个图案单元。

在执行 PaintProc 时的图形状态是图案通过执行 MakePattern 操作符而被实例化时的图形状态，包括 CurrentTransformation 被修改，像执行了 MakeRaster 和 MaskBitMap 操作符一样。

如果 PaintProc 执行了 ImageRasterElement 或 MaskBitMap 操作符，则操作使用的数据源均不能为 DataSource/Document。如果使用的数据源之一是从外部数据源派生来的 StreamObject，则必须执行 FindResource 操作符，以获取在 PaintProc 中必须出现的 StreamObject。

注：因为具体实现时将保留一个最近使用的图案单元的缓存以优化执行，PaintProc 可能在不可预测的文件中执行且执行不可预测的次数，PaintProc 应只依赖于系统词典中的操作符及它的 PatternDict 操作数的内容，且总产生相同的结果。关于执行 PaintProc 操作的进一步的限制可见 20.2.1 中关于 PaintType 的描述。

20.2.3 BBox

必选项〈BBox:Vector〉是一个 4 个类型为 Number 元素的向量，指出了图案单元的边框(bounding box)，它用图案单元坐标系统来度量。此边框用来裁剪图案单元，向量的 4 个元素分别指出边框的左下角的 x 和 y 坐标，右上角的 x 和 y 坐标：

〔llx lly upx upy〕

20.2.4 XStep

必选项〈XStep:Number〉指出水平方向上两图案单元间的距离，由图案单元坐标系来度量。

20.2.5 YStep

必选项〈YStep:Number〉指出垂直方向上两图案单元间的距离，由图案单元坐标系来度量。

注：XStep 和 YStep 的值可以和 BBox 项隐含的图案单元的尺寸不同，这就可以用不规则的图形来覆盖。

20.2.6 TilingType

图案单元放置的位置是基于一个关键图案单元的位置，然后根据 XStep 和 YStep 依次重复放置。关键图案单元的原点与坐标系统的原点一致，它是由执行 MakePattern 时的 CurrentTransformation 和 MakePattern 操作符的 Transformation 操作数共同定义的。因此，覆盖的相位可由 Trasformation 操作符的平移〈Translation〉分量来控制。这种覆盖是固定的，也就是说不论由 MakePattern 建立的图案词典在何时用于绘图，都使用相同的覆盖，而不管 CurrentTransformation 的变化。

必选项〈TilingType:Cardinal〉控制此覆盖的调节，以适应设备像素网络。TilingType 的有效值为：

1）常量间隔。图案单元有固定的间隔，也就是说，由多个设备像素组成。要得到此效果，MakePattern 操作符可能需要微小地变形图案，它通过对 XStep，YStep 及 MakePattern 操作符的 Transformation 操作数的微小调节来实现的，变形的数量不能超过一个设备像素。

2）不变形。图案单元不被歪曲，但是当在描绘图案时，图案单元在 X 和 Y 方向的间隔可能有最大为一个设备像素的差别，这使得 XStep 和 YStep 要求的间隔较为平均，但不是单个图案单元。

3）常量间隔及快速覆盖（类似 TilingTypel），但允许图案单元额外的变型，以便更加有效地实现。额外变形的数量依赖于具体实现。

20.2.7 实现 Implementation

〈Implementation:Any〉由 MakePattern 操作符加入图案词典，它是为了具体实现时用来存储信息，以帮助正确地覆盖且尽可能地高速存储图案。此项的类型和值依赖于具体实现，如果此项存在于 MakePattern 操作符的 PatternObject 中，则它的值可以由 MakePattern 操作符重写。

20.3 图案语义 Pattern Semantics

SetPatternColour 操作符所需操作数的数量和特性依赖于所使用的图案（彩色的或掩膜的）类型，对于每种图案类型的 SetPatternColour 操作符的语义相应在 20.3.1 和 21.3.2 中论述。

20.3.1 彩色化图案 Coloured Pattern

彩色化图案是一个色彩自包含的图案。作为描绘图案单元的一部分，PaintProc 显式地设置所有它描绘的图形元素的颜色。单个图案单元可以包含涂以不同颜色的元素，它也可能包含灰度或彩色的取样图像。

SetPatternColour 操作符选择一个彩色图案所需的操作数是：

〈PatternDict:Dictionary〉

其中 PatternDict 是执行 PaintType 为 1 的 MakePattern 操作符所得到的图案词典。

20.3.2 掩膜图案 Mask Pattern

掩膜图案是一个设有任何内在颜色的图案，而它的颜色在使用此图案时必须单独指出。这提供了在页面区域上覆盖以相同形状但不同颜色的图案单元的一种方法。图案的 PaintProc 不显式地指定任何颜色，它不能包含灰度或彩色取样图像（但它可以使用 MaskBitMap 操作符）。

SetPatternColour 操作符用来选择掩膜图案以及描绘掩膜图案颜色的操作数格式如下：

〈Patterndict:Dictionary〉

〈$Comp_n$:Any〉

…

〈$Comp_1$:Any〉

其中 PatternDict 是通过执行 MakePattern 操作符获得的图案词典，其中 PaintType 是 2，其余部分是在彩色空间中选择描绘图案的颜色（由在当前执行上下文中最近执行的 SetColourSpace 操作符来指定）。

20.4 为文件内容建立图案词典原型

作为 MakePattern 操作符的一个操作数的图案对象是通过执行 FindResource 操作符获得的。FindResource 操作符取一个 INTERNAL IDENTIFIER 的值作为操作数，它原先由 RESOURCEDECLARATION 约束成结构中的一个图案资源，且返回一个对图案对象的引用。

注：FindResource 操作符并不检查图案词典，以确定所需的项是否存在，且为正确的类型。这些功能由 MakePattern 操作符来完成。

20.5 操作符

这些操作符及它们语义的说明包括了解释这些操作符可能引起的内容异常的条件的说明。内容异常处理在第 24 章中定义，除了这些操作符特定异常外，还有一些普通的异常，它们在解释几乎所有操作符时均可能发生，这些普通的异常及它们的语义在 24.6.2 中论述。

20.5.1 MakePattern(建立图案)

MakePattern 操作符取两个操作数：

〈T：Transformation〉

〈PatternObject：Dictionary〉

返回一个结果：

〈PatternDict：Dictionary〉

图案坐标系由当前图形状态来构造，过程如下：

注：此论述只用于说明 MakePattern 操作符的语义，而不是说明在具体实现时如何与网纹语义相符。

——操作数 T 与当前图形状态中 CurrentTransformation 的一个拷贝相级联，所得结果的 Transformation 将被调整，以保证设备像素空间可用指定大小的图案单元正确地覆盖，这由 PatternObject 中的 TilingType 指定。

——CurrentPath 被设置成空路径，与执行 NewPath 操作符一样。

——CurrentClipRegion 被图案单元的界框所替代(由 PatternObject 中的 BBox 项定义)。

MakePattern 然后产生一个 PatternObject 的新拷贝，加入 Implementation 项，将 Dictionary 的存取属性至少降至只读，并将其压入操作数栈。

注：因为在一些具体实现中，MakePattern 操作符可能更多地改变词典内容，降低字典存取属性是为了防止以后的文本去修改词典内容。不允许任何后续文本存取词典内容(希望通过执行 SetPatternColour 来实现)，可以将词典的存取属性降为 ExecuteOnly。

20.5.2 SetPatternColour(置图案彩色)

SetPatternColour 的性质及所需的操作数依赖于已设置的图案类型(彩色的或掩膜的)。这些操作数在 20.3.1 和 20.3.2 中论述，SetPatternColour 不返回结果，执行 SetPatternColour 操作符的结果是去设置 CurrentColour 成像变量的值。

21 还原和还原控制的操作符

本章叙述了还原模型及控制图形元素和彩色如何在表示设备上还原的机制。

注：大多数控制还原过程的操作符在使用时要求知道表示设备的特性。如果要做到设备无关性，文件中不能使用本章中描述的任何操作符。

21.1 还原模型

几乎程序控制下的所有还原工具都与彩色的重新生成有关。彩色的还原模型在逻辑上是一个多步骤过程，它的语义在下面列出，根据当前彩色空间和表示设备的不同特点，并不总是需要做到所有的步骤：

——如果在 CIE—关联的彩色空间中指定了一种彩色，正如 15.3 中所说的，那么表示过程必须首先把它转换成适用于表示设备的某种设备彩色空间(DeviceRGB，DeviceCMYK，DeviceKX 或 DeviceGrey)，这种转换由彩色还原字典(ColourRenderingDictionary)来控制。

——如果在一个设备彩色空间中指定了一种彩色，正如 15.4 中所说的，而它与实际显示设备不匹配(例如用 DeviceRGB 彩色空间在灰度表示设备上指定一种彩色)，那么表示过程将调用一个彩色转换函数

——然后表示过程利用转换函数对每个彩色分量进行设备彩色值的映射变换(这一步有时也称为

伽马修正)。转换函数补偿了表示设备和人眼对彩色的非线性效应(response)。

——如果表示设备不能产生连续色调而只能表示特定的离散色彩(例如黑白像素),则表示过程调用半色调函数(halftone functions),它将通过像素图案来逼近所要求的彩色。

——最后,表示过程执行扫描转换(scan conversion),根据要求的彩色,在表示设备上将相应的像素涂色。

本章的其余部分讨论了 CIE——关联的彩色到设备彩色的转换过程中彩色还原字典的使用;不同设备彩色空间之间的转换算法;以及转换函数、屏幕、半色调函数的定义。21.10 中描述了提供这些功能控制的操作符。

注:本条仅给出了还原模型;而未指定基本的实现方法,但一个合格的实现系统应该提供其语义和 21.10 中的定义完全一致的操作符。

21.2 CIE 彩色到设备彩色的转换

把一个 CIE 彩色值转换为一个设备彩色值要求完成两个主要的操作:

——根据色域转换映射(gamut transfer mapping)调整 CIE 彩色值。色域(gamut)是彩色空间中所有可能彩色的子集,一个文件中使用的所有彩色构成了一个源色域(source gamut),表示设备上能够产生的所有彩色构成了一个设备色域(device gamut)。如果文件中指定了设备不能产生的彩色,本步骤便把该彩色从源色域转换到设备色域。

——根据彩色还原函数生成一个对应的设备彩色值。对于一个给定的 CIE 彩色值,该函数将计算出设备彩色空间中对应的彩色值。

当一个文件执行 SetColourSpace 选择了 CIE 彩色空间后,也就指定了源色域(Source Gamut),这种描述是与设备无关的。

设备色域和彩色还原函数由 Colour Rendering Dictionary 描述,这个字典是图形状态的一个参数,默认的彩色还原字典以及它的内容完全依赖于实现。SetColourRendering 操作符在图形状态中安装了一个新的彩色描述字典,GetColorRendering 操作符返回当前彩色还原字典,色域转换映射和彩色还原函数仅仅用于 CIE 彩色空间中的彩色值。根据定义,设备彩色空间中的彩色值直接控制设备彩色的各个分量。

注:21.2.1 引用了许多彩色还原字典中的内容,见 21.3 中对彩色还原词典的描述。

21.2.1 CIE 彩色到设备彩色的变换

彩色还原字典中所有的项定义了一个把 CIE 彩色值变换为设备彩色值的函数。还原函数的输出还需要进行进一步变换——设备彩色空间的转换、变换函数(transfer function)和半色调技术的使用。

每一个彩色还原字典中都必须有一个值的类型为基数的项 ColourRenderingType,它从整体上说明了彩色描述函数的结构,字典中的其他项都将依据这个值来解释。本标准仅仅定义了 ColourRenderingType 值为 1 的彩色还原字典的语义。一个实现系统的默认彩色还原字典的 ColourRenderingType 的值可以不为 1。

ColourRenderingType 为 1 的彩色还原字典代表的彩色还原函数的类型决定于 CIE 1931(XYZ)空间的两级非线性变换,这个空间称为还原彩色空间(render colour space),它的值按照下面两种途径之一处理:

——直接作为 DeviceRGB 或 DeviceGrey 彩色空间的彩色值使用;

——用作一张三个分量的彩色对照表(three-dimensional lookup table)的索引,这张表分别包含了可在 DeviceRGB 或 DeviceCMYK 彩色空间中解释的彩色值。

第一种方法最适合用于加色模型(additive)的线性彩色设备,而第二种方法适用于那些不能用一个简单公式来描述其彩色的设备,以产生高精度的显示。

从概念上来讲,从 CIE 彩色空间到设备彩色空间的转换包括如下几步:

1) 把基于 CIE 的彩色值从它的原始彩色空间转换到 CIE 1931(XYZ)-空间,这种转换依赖于彩色

空间的不同参数。

2) 调整 X,Y,Z 的值以解决源和设备中黑白点(white point 和 Black Point)之间的差异。根据彩色还原字典中 MatrixPQR 和 TransformPQR 项的值进行转换,力图保持彩色效果和视觉反差(visual contrast)。源的漫射白黑点是作为彩色空间的 WhitePoint、BlackPoint 参数给出的;设备的漫射白、黑点是作为彩色描绘字典的 whitePoint 项和 BlackPoint 项给出的。如果对应的项相等,那么本步便简化为恒等变换。

3) 根据彩色还原字典中的 MatrixLMN,EncodeLMN,MatrixABC,EncodeABC 项,将彩色值从 CIE 1931(XYZ)空间转换到还原彩色空间,产生三个分量 A,B,C。

注:这三个分量是随意命名的,不能与 CIE BasedABC 彩色空间的 A,B,C 三个分量相混淆。

4) 如果彩色还原字典中存在 Render Table 项,用 A,B,C 三个分量作为索引去查找三维的彩色对照表,产生一个内插值。根据表的定义方式的不同,这个值由 3 或 4 个彩色分量组成,每个分量由一个过程把它转换为设备彩色空间的一个彩色分量。如果有 3 个分量,它们认为是 DeviceRGB 彩色空间中的红、绿、蓝值;如果有 4 个值,它们定义了 DeviceCMYK 彩色空间中的青,品红,黄和黑的值。

如果没有 RenderTable,则 A、B、C 分量直接用作设备彩色值。如果设备自身彩色空间是 DeviceGrey,那么 A 分量表示灰度值,而 B,C 值被忽略,否则 A,B,C 分量分别定义了 DeviceRGB 彩色空间的红、绿、蓝三色的值。

21.3 彩色还原字典 Colour Rendering Dictionary

本条说明彩色还原字典各项的含义。

21.2 中已给出了关于这些项在 CIE 彩色到设备彩色转换过程中应用的高层次的描述。

必选项〈ColourRenderingType:Cardinal〉从整体上选择了彩色描述函数的结构,这一项在所有的彩色还原字典中都必须有。本标准中提供的唯一的标准 ColourRenderingType 是 1。ColourRenderingType 1 彩色还原字典中的必选项和可选项(除了必选项 ColourRenderingType 外)都在 21.3.1 中描述。

21.3.1 彩色还原类型 1 字典

21.3.1.1 WhitePoint(白点)

必选项〈WhitePoint:Vector〉是一个有三个数值型元素的向量,这三个元素指出了表示设备的漫射白点的 CIE 1931(XYZ)空间三刺激值(tristimulus)。

每个元素的值必须大于 0,Y 分量必须为 1,向量的元素顺序为:

[Xw 1 Zw]

21.3.1.2 BlackPoint(黑点)

可选项〈BlackPoint:Vector〉是一个有三个数值型元素的向量,这三个元素指出了表示设备的漫射黑点的 CIE 1931(XYZ)空间三刺激值。

每个元素的值必须大于 0,向量中的元素顺序为

[X_b Y_b Z_b]

如果 BlackPoint 项不存在,默认值为[000]

21.3.1.3 MatrixPQR

可选项〈MatrixPQR:Vector〉是一个有九个数值型元素的向量,这些元素说明了 CIE 1931(XYZ)空间的 x,y,z 分量相对于 PQR 中间表示的线性解释,正如 21.3.1.4 节所述。向量中的元素顺序为:

[Px Qx Rx Py Qy Ry Pz Qz Rz]

如果 MatrixPQR 不存在,那么默认的矩阵为:

[100 010 001]

21.3.1.4 TransformPQR

必选项〈TransformPQR:Vector〉是一个有三个过程类元素的向量,这些过程变换 PQR 中间表示的 P、Q、R 分量以补偿源和设备的漫射 WhitePoint 和 BlackPoint 之间的差别,从而保持彩色效果和视觉反

差。向量中的元素顺序为：

$[T_p \quad T_q \quad T_r]$

设 X_{Ws}, Y_{Ws}, Z_{Ws} 和 X_{Bs}, Y_{Bs}, Z_{Bs} 分别是源的漫射白点和黑点的 CIE 1931(XYZ)空间的三刺激值。设 X_{Wd}, Y_{Wd}, Z_{Wd} 和 X_{Bd}, Y_{Bd}, Z_{Bd} 分别是设备的漫射白点和黑点的 CIE 1931(XYZ)空间的三刺激值，源和表示设备的 CIE 1931(XYZ)空间的三刺激值分别为 X_s, Y_s, Z_s 和 X_d, Y_d, Z_d，它们通过 MatrixPQR 和 TransformPQR 项联系起来：

$$P_s = X_s \times P_X + Y_s \times P_Y + Z_s \times P_X$$
$$Q_s = X_s \times Q_X + Y_s \times Q_Y + Z_s \times Q_X$$
$$R_s = X_s \times R_X + Y_s \times R_Y + Z_s \times R_X$$

$$P_d = T_P(W_s, B_s, W_d, B_d, P_s)$$
$$Q_d = T_Q(W_s, B_s, W_d, B_d, Q_s)$$
$$R_d = T_R(W_s, B_s, W_d, B_d, Q_s)$$

$$X_d = P_d \times X_P + Q_d \times X_Q + R_d \times X_R$$
$$Y_d = P_d \times Y_P + Q_d \times Y_Q + R_d \times Y_R$$
$$Z_d = P_d \times Z_P + Q_d \times Z_Q + R_d \times Z_R$$

其中：

$$W_s = [X_{Ws} Y_{Ws} Z_{Ws} P_{Ws} Q_{Ws} R_{Ws}]$$
$$B_s = [X_{Bs} Y_{Bs} Z_{Bs} P_{Bs} Q_{Bs} R_{Bs}]$$
$$W_d = [X_{Wd} Y_{Wd} Z_{Wd} P_{Wd} Q_{Wd} R_{Wd}]$$
$$B_d = [X_{Bd} Y_{Bd} Z_{Bd} P_{Bd} Q_{Bd} R_{Bd}]$$

$$P_{Ws} = X_{Ws} \times P_X + Y_{Ws} \times P_Y + Z_{Ws} \times P_Z$$
$$Q_{Ws} = X_{Ws} \times Q_X + Y_{Ws} \times Q_Y + Z_{Ws} \times Q_Z$$
$$R_{Ws} = X_{Ws} \times R_X + Y_{Ws} \times R_Y + Z_{Ws} \times R_Z$$

$$P_{Bs} = X_{Bs} \times P_X + Y_{Bs} \times P_Y + Z_{Bs} \times P_Z$$
$$Q_{Bs} = X_{Bs} \times Q_X + Y_{Bs} \times Q_Y + Z_{Bs} \times Q_Z$$
$$R_{Bs} = X_{Bs} \times R_X + Y_{Bs} \times R_Y + Z_{Bs} \times R_Z$$

$$P_{Wd} = X_{Wd} \times P_X + Y_{Wd} \times P_Y + Z_{Wd} \times P_Z$$
$$Q_{Wd} = X_{Wd} \times Q_X + Y_{Wd} \times Q_Y + Z_{Wd} \times Q_Z$$
$$R_{Wd} = X_{Wd} \times R_X + Y_{Wd} \times R_Y + Z_{Wd} \times R_Z$$

$$P_{Bd} = X_{Bd} \times P_X + Y_{Bd} \times P_Y + Z_{Bd} \times P_Z$$
$$Q_{Bd} = X_{Bd} \times Q_X + Y_{Bd} \times Q_Y + Z_{Bd} \times Q_Z$$
$$R_{Bd} = X_{Bd} \times R_X + Y_{Bd} \times R_Y + Z_{Bd} \times R_Z$$

换句话说，CIE 1931(XYZ)空间中源彩色的 X_s, Y_s, Z_s 分量被当作一个三元向量，乘以 MatrixPQR（一个 3 乘 3 矩阵），结果产生了与 PQR 中间表示有关的源彩色的 P_s, Q_s, R_s 三个分量，用 TransformPQR 过程对这些分量分别进行变换，产生对应的设备彩色的 P_d, Q_d, R_d 分量。每个分量单独转换，相互

间没有影响。最后，设备彩色的 P_d，Q_d，R_d 分量被当作是一个三元向量，并乘以 MatrixPQR 的逆，结果便是设备彩色在 CIE 1931(XYZ)空间的 X_d，Y_d，Z_d 分量。

正象上面等式所示，每个 TransformPQR 过程都有五个操作数：

〈P_s 或 Q_s 或 R_s：Number〉

〈B_d：VectorReference〉

〈W_d：VectorReference〉

〈B_s：VectorReference〉

〈W_s：VectorReference〉

各操作数如上面等式所示，执行 TransformPQR 过程返回结果

〈P_d 或 Q_d 或 R_d：Number〉

这里的结果是其 Number 参数的单调函数。

注：既然这些过程在依赖于具体实现的场合下调用，那么它们应该作为纯函数来操作，不依赖于任何非默认的上下文，也不会影响当前上下文。

21.3.1.5 RangePQR

可选项〈RangePQR：Vector〉是一个有六个数值型元素的向量，它的元素说明了 P，Q，R 分量的有效值的范围。RangePQR 的元素可解释为三对边界，分别对应于三个彩色分量：

[P_0 P_1 Q_0 Q_1 R_0 R_1]

三个彩色分量的有效值范围定义为：

$P_0 \leqslant P \leqslant P_1$，$Q_0 \leqslant Q \leqslant Q_1$，$R_0 \leqslant R \leqslant R_1$

如果 RangePQR 项不存在，那么默认范围为：

[01 01 01]

21.3.1.6 MatrixLMN

可选项〈MatrixLMN：Vector〉是一个有九个数值型元素的向量，它们说明了 CIE 1931(XYZ)空间的 x，y，z 分量相对于 LMN 中间表示的线性解释，正如 21.3.1.7 所述。这个向量的元素顺序为：

[L_x M_x N_x L_y M_y N_y L_z M_z N_z]

如果不存在 MatrixLMN 项，默认矩阵为：

[100 010 001]

21.3.1.7 EncodeLMN

可选项〈EncodeLMN：Vector〉是一个有三个过程类元素的向量，这些过程将 LMN 中间表示的 L、M、N 分量进行编码。向量的元素顺序为：

[E_L E_M E_N]

将未编码的 L、M 或 N 的值作为操作数调用这些过程，则将返回对应的编码值。

注：既然这些过程在依赖于具体实现的场合下调用，那么他们必须作为纯函数来操作——不依赖于任何非默认的上下文，也不会影响当前上下文。

如果不存在 EncodeLMN 项，则使用的默认值是一个空过程向量，它不会对操作数栈产生任何影响：

[{} {} {}]

由 MatrixLMN 和 EncodeLMN 项定义的变换是：

$L = E_L(X \times L_X + Y \times L_Y + Z \times L_Z)$

$M = E_M(X \times M_X + Y \times M_Y + Z \times M_Z)$

$N = E_N(X \times N_X + Y \times N_Y + Z \times N_Z)$

21.3.1.8 RangeLMN

可选项〈RangeLMN：Vector〉是一个有六个数值型元素的向量，这些元素说明了中间表示的 L、M、

N 分量的有效值范围。RangeLMN 的元素可解释为三对边界，分别对应于三个彩色分量：

$[L_0 \quad L_1 \quad M_0 \quad M_1 \quad N_0 \quad N_1]$

三个彩色分量的有效值范围分别定义为：

$L_0 \leqslant L \leqslant L_1, M_0 \leqslant M \leqslant M_1, N_0 \leqslant N \leqslant N_1$

如果不存在 RangeLMN 项，默认的范围为：

[0 1 0 1 0 1]

21.3.1.9 MatrixABC

可选项〈MatrixABC:Vector〉是一个有九个数值型元素的向量，这些元素说明了 LMN 中间表示的 L,M,N 分量相对于还原彩色空间的线性解释，正如 21.1.3.10 所述。

向量中的元素顺序为：

$[A_L \quad B_L \quad C_L \quad A_M \quad B_M \quad C_M \quad A_N \quad B_N \quad C_N]$

如果 MatrixABC 项不存在，使用的默认矩阵为：

[100 010 001]

21.3.1.10 EncodeABC

可选项〈EncodeABC:Vector〉是一个有三个过程类元素的向量，它们是对描绘彩色空间的 A,B,C 分量进行编码的过程。向量中元素顺序为：

$[E_A \quad E_B \quad E_C]$

将未编码的 A,B,C 的值作为操作数来调用这些过程，则返回对应的编码值。

注：既然这些过程在依赖于具体实现的场合下调用，那么他们必须作为纯函数来操作，它不依赖于任何非默认的上下文，也不会影响当前上下文。

如果不存在 EncodeABC 项，使用的默认值是一组空的过程向量，它们不会对操作数栈产生任何影响：

[{} {} {}]

由 MatrixABC 和 EncodeABC 项定义的变换是：

$A = E_A(L \times A_L + M \times A_M + N \times A_N)$

$B = E_B(L \times B_L + M \times B_M + N \times B_N)$

$C = E_C(L \times C_L + M \times C_M + N \times C_N)$

21.3.1.11 RangeABC

可选项〈RangeABC:Vector〉是一个六个数值型元素的向量，它们指出了 A,B,C 分量的有效值范围。RangeABC 的元素被认为是三对边界，分别对应于三个彩色分量：

$[A_0 \quad A_1 \quad B_0 \quad B_1 \quad C_0 \quad C_1]$

三个彩色分量的有效值的范围分别为：

$A_0 \leqslant A \leqslant A_1, B_0 \leqslant B \leqslant B_1, C_0 \leqslant C \leqslant C_1$

如果没有 RenderTable 项，这些值的范围必须在 0 到 1 之间，因为这时还原彩色空间直接映射到设备彩色空间；如果有 RenderTable 项，这些值的范围定义了这三个分量的彩色表的边界。

如果没有 RangeABC 项，默认的范围是

[0 1 0 1 0 1]

21.3.1.12 RenderTable

可选项〈RenderTable:Vector〉是一个向量，它的元素描述了一张三维彩色对照表和一组相关的过程，这组过程通过表的查找和插入，将还原彩色空间的彩色值映射为设备彩色空间的彩色值。这个向量的形式为：

$[\langle N_A:\text{Cardinal}\rangle\langle N_B:\text{Cardinal}\rangle\langle N_C:\text{Cardinal}\rangle$

$\langle\text{table}:\text{Vector}\rangle\langle m:\text{Cardinal}\rangle\langle T_1:\text{Procedure}\rangle$

〈T_2:Procedure〉…〈T_m:Procedure〉]

其中彩色表的分量范围由 N_A,N_B,N_C 给出,这三个值必须大于 0。表中的每一项由 m 个编码的彩色分量值组成,m 为 3 或 4。对应于还原彩色空间分量 A,B,C 已编码的设备彩色值,它包含在表中由整数坐标(a,b,c)所指出的项中,其中

$$0 \leqslant a \leqslant N_A, 0 \leqslant b \leqslant N_B, 0 \leqslant c \leqslant N_C$$

并且有:

$$A = A_0 + a \times (A_1 - A_0)/(N_A - 1)$$

$$B = B_0 + b \times (B_1 - B_0)/(N_B - 1)$$

$$C = C_0 + c \times (C_1 - C_0)/(N_C - 1)$$

A_0,A_1,B_0,B_1,C_0,C_1 的值在 RangeABC 项中给出。

table 元素是一个由 N_A 个八位字节串构成的向量,它们定义了彩色表的内容,每个八位字节串应该含有 $m \times N_B \times N_C$ 个字符,对于该向量中的第 a 个字节串,从位置 $m \times (b \times N_C + C)$ 开始的 m 个八位字节,便是坐标(a,b,c)所定位的表项,这几个八位字节被解释为已编码的设备彩色分量 e_1,$e_2 \cdots e_m$。

元素 T_1,$T_2 \cdots T_m$ 是将编码的分量转换为实际设备彩色的分量值的一组过程,这种转换为:

$$d_1 = T_1(e_1/255)$$

$$d_2 = T_2(e_2/255)$$

$$\cdots$$

$$d_m = T_m(e_m/255)$$

换句话说,彩色还原函数把从表中获得的八位字节看成一个整数,将它除以 255(产生一个 0 到 1 之间的数),将结果压入操作数栈,然后调用相应的 T 过程,这个过程取出它的操作数,并产生一个值在 0 到 1 之间的结果。

注:既然这些过程在依赖于具体实现的场合下调用,那么它们必须作为纯函数来操作——不依赖于任何非默认的上下文,也不会影响当前上下文。

d_1,d_2,$\cdots d_m$ 的值组成了最终的设备彩色值,也就是如果 m 为 3,d_1、d_2 和 d_3 分别为红、绿、蓝分量,如果 m 是 4,那么 d_1、d_2、d_3、d_4 分别是青、品红、黄色和黑色分量

21.4 设备彩色空间之间的转换

典型的表示设备都有自己的设备彩色空间,对应于彩色空间 DeviceGrey、DeviceRGB 或 DeviceCYMK 之一。如果设备支持连续色调的输出,彩色的重新生成直接可以实现;否则可以通过半色调技术来实现。如果文件是在设备自身彩色空间中指定,那么不需要转换;否则,表示过程应该将文件中指定的彩色值转换为设备自身彩色空间指定的彩色值。

当文件在 CIE 彩色空间中指定彩色值时,通常不进行设备彩色变换;CIE 彩色还原函数直接映射到设备自身彩色空间(见 21.2)。因此本条仅仅描述了从一个设备彩色空间到另一个设备彩色空间的转换。本标准并未定义从 DeviceKX 彩色空间到其他设备相关的彩色空间以及从其他设备彩色空间到 DeviceKX 的转换过程。

下面几条中描述的转换过程并不包括转换函数或半色调函数的使用,只有当真正要在输出设备上还原彩色时,才会调用转换函数和半色调函数,以便对彩色转换操作的输出进行处理。

21.4.1 DeviceRGB 和 DeviceGrey 之间的转换

黑白以及不同浓淡的灰度可以认为是 DeviceRGB 彩色的特殊情况,DeviceRGB 彩色值与特定 DeviceGrey 彩色值之间的对应关系十分简单:

red=grey

green=grey

blue=grey

对于一个给定的 DeviceRGB 彩色值,可以根据 NTSC 显示标准公式来计算得到对应的 DeviceGrey

彩色值：

grey＝3×red＋59×green＋11×blue

21.4.2 DeviceCMYK 和 DeviceGrey 之间的转换

一般情况下，DeviceGrey 彩色值是对应的 DeviceCMYK 彩色值中 Black 分量的补数，因此，特定的 DeviceGrey 的彩色值简单地等价于 DeviceCMYK 的彩色值：

Gyan＝0.0

magenta＝0.0

yellow＝0.0

black＝1.0－Grey

从 DeviceCMYK 彩色值到 DeviceGrey 的转换，等价于先从 DeviceCMYK 转换到 DeviceRGB，再从 DeviceRGB 转换到 DeviceGrey 的过程，这两种转换结合起来表示为：

grey＝1.0－min(1.0,0.3×Cyan＋0.59×magenta＋0.11×yellow＋black)

其中 min 是一个函数，它返回值集中的最小值。

21.4.3 DeviceRGB 到 DeviceCMYK 的转换

从 DeviceRGB 彩色值到 DeviceCMYK 彩色值的转换分为两步，第一步可以通过表示红—绿—蓝以及青—品红—黄之间关系的等式来描述；第二步用黑色生成函数(black generation)和底色去除函数(under colour removal)来产生一个黑色分量，并且调整其他分量，以便更准确地表示原来的颜色。

相减基色(subtractive colour primary)青、品红和黄色与相加基色红、绿、蓝互为补色；所以在理论上，转换是十分简单的：

cyan＝1.0－red

magenta＝1.0－green

yellow＝1.0－blue

逻辑上，青(cyan)、品红(magenta)和黄色(yellow)足以产生任何打印色，因此，同样百分数的青，品红和黄色可以生成相同百分数的黑色。

21.4.3.1 Black Generation 函数和 Under Colour Removal 函数

实际上，彩色绘图油墨并不可能完全地混合；这种混合常常构成深棕色的阴影，因此，通常要用真正的黑油墨去取代彩色中的黑色成分，以获得更加真实的彩色还原。计算这个分量的值需要几个附加的步骤：

——当要显示一种特定颜色时，black generation 函数计算出使用黑色的数量。

——undercolur removal 函数减少青、品红以及黄色分量的数量，以补偿由 black generation 函数加入的黑色数量。

从 DeviceRGB 转换到 DeviceCMYK 彩色空间时，SPDL 只提供对于黑色生成(black generation)和底色去除(undercolour removal)的有限控制，需要更好地控制的应用程序必须在 CIE 彩色空间中指定彩色，并通过 CIE 彩色还原字典来控制到 DeviceCMYK 颜色的转换(见 21.2)。从 DeviceRGB 到 DeviceCMYK 的完整的转换过程如下所述。其中 BG(K)和 UCR(K)分别表示对 black generation 和 undercolour removal 函数的调用。

c＝1.0－red

m＝1.0－green

y＝1.0－blue

k＝min(c,m,y)

cyan＝min(1.0,max(0.0,c－UCR(K)))

magenta＝min(1.0,max(0.0,m－UCR(K)))

yellow=min(1.0,max(0.0,y−UCR(K)))

black=min(1.0,max(0,0,BG(K)))

其中 min 是一个函数，它返回值集中的最小值，而 max 是一个返回值集中最大值的函数。

black generation 和 undercolour removal 函数定义为过程，且分别保存在图形状态变量 CurrentBlackGeneration 和 CurrentUnderColourRemoval 中，操作符 SetBlackGeneration 和 SetUnderColourRemoval 用以设置图形状态中的这些参数。

注：状态变量 CurrentBlackGeneration 和 CurrentUnderColourRemoval 的默认值依赖于具体应用。

表示过程在进行 DeviceRGB 彩色到 DeviceCMYK 彩色的转换时调用这些过程，每个过程有一个数值操作数，并返回一个数值结果。因为这些过程在不可预测的场合下调用，它们必须是一个纯函数，没有副作用。

两个过程的操作数都是 K，即中间变量 c、m、y 值中的最小值。c、m、y 可由 1 减去原来的红、绿、蓝值而得。

一般情况下，K 是黑色的值，它可以从青、品红和黄色分量中移出，成为一个独立的黑色分量。black generation 函数计算出 K 值对应的函数值作为实际的黑色分量。

undercolour removal 函数计算出一个值，用中间变量 c，m 和 y 的值减去它，就生成最终的青、品红、黄色分量的值。函数返回值在−1 和 1 之间(闭区间)。如果它返回一个负数，就将这个返回值的绝对值加到各个分量上。经 black generation 和 undercolour removal 函数作用的分量值应该在 0 到 1 之间。如果有一个值在这个范围之外，那末将自动地用最接近的有效值来取代它，不产生错误提示，如上面的 min 和 max 函数所示。

21.4.4 DeviceCMYK 到 DeviceRGB 的转换

从 DeviceCMYK 彩色到 DeviceRGB 彩色的转换过程是一个简单的操作，它不包括 black generation 或 undercolour removal 函数：

red=1.0−min(1.0,cyan+black)

green=1.0−min(1.0,magenta+black)

blue=1.0−min(1.0,yellow+black)

其中 min 是一个返回值集中最小值的函数。

21.5 转换函数

转换函数调整彩色分量的值，以补偿表示设备和人眼之间的非线性效应。设备彩色空间的每个分量(例如，DeviceRGB 彩色空间的红色分量)总试图表现与其数值相称的可见亮度或强度。但很多设备并不体现这种线性效应，变换函数可以补偿设备的实际响应，这种操作有时称为伽马修正。

在处理彩色的一系列步骤中，显现过程在进行不同彩色空间的转换(如果必要的话)之后，但在调用半色调函数(如果必要的话)之前使用转换函数，独立的转换函数处理每个彩色分量。互相之间没有影响。转换函数在输出设备的自身彩色空间中操作，不考虑原指定彩色时所在的彩色空间。

21.5.1 指定转换函数

有三种方法可以用来指定转换函数：

——SetTransfer 操作符建立一个可用于所有彩色分量的转换函数。

——SetColourTransfer 操作符建立 4 个独立的转换函数，分别处理红、绿、蓝和灰度(或者是他们的补色青、品红、黄色和黑色)。RGB 彩色设备使用前三个，单色设备仅用灰度转换函数；CMYK 设备可以用全部四个函数。

—— SetHalftone 操作符可以通过半色调字典中的可选项来建立转换函数(见 21.7)，这是为 NamedColour 彩色空间的分量建立转换函数的唯一途径，半色调字典中指定的转换函数将取代由 SetTransfer 指定的函数。

转换函数可表示为过程，并保存在图形状态变量 CurrentTransfer 中。SetTranfer、SetColourTransfer

和 SetHalftone 操作符用以修改这个状态变量的值。

注：状态变量 CurrentTransfer 的默认值依赖于具体实现。

转换函数的操作数类型为 Number，范围在 0.0 到 1.0 之间(包括 0.0 和 1.0)，执行后的返回值也在这个范围内。这个操作数就是设备自身彩色空间中彩色分量的值(直接指定的或者从其他彩色空间转换而来的)，结果就是送给设备的转换过的值(如果需要的话，还要进行半色调处理)。

在解释转换函数的操作数和结果时，通常总认为各彩色分量是加色型的(红，绿，蓝或灰)，也就是说，值越大代表的彩色越亮。如果彩色分量是减色型的(青、品红、黄、黑)，表示过程在把它们送给转换函数前先由 1 减去它们，以便将它们转换为加色方式(additive form)。转换函数的结果通常是加色方式，并以这种方式传送给半色调函数，这一惯例同样适用于 NamedColour 分量。

注：既然这些过程在依赖于具体实现的场合下调用，那么它们必须作为纯函数来操作——不依赖于任何非默认的上下文，也不会影响当前上下文。

21.6 半色调函数

半色调技术就是用像素图案逼近连续色调彩色的一种处理方法。这组图案只能模拟有限几种离散的彩色，该技术最为人熟知的例子就是用黑、白像素来还原灰度。有些设备能直接产生连续色调的彩色，对于这种设备，不需要使用半色调技术。

如果要使用半色调技术，必须先用适当的变换函数对各彩色分量进行变换，这样，经过伽马修正后的表示设备自身彩色空间中连续色调的彩色分量就构成了半色调函数的输入。半色调函数的输出由设备确能生成的表示彩色的那些像素组成。经过变换函数的伽马修正后，彩色分量可直接传送给设备。

注：与大多数还原控制机制一样，半色调函数本身是针对特定设备的，并且只能用于某些文件中，在这个文件的已知的局部环境中确实要求使用这样的控制才能得到高质量的输出。那些在取得高质量输出时不要求使用半色调函数的文件可以不定义它们自己的半色调函数，转而使用那些在所有符合标准的应用系统中都存在的依赖于具体实现的默认半色调函数。

21.6.1 半色调屏幕模型

SPDL 支持的半色调函数基于半色调屏幕的使用。这个屏幕在概念上可如此定义：将一个由同样的半色调单元构成的正方形网格覆盖在设备像素阵列上，每个像素属于网格中的一个单元。每个半色调单元通常包含许多设备像素，屏幕网格完全在设备像素空间中定义，修改 CurrentTransformation 也不能改变它；这种性质保证相邻的两个半色调区域能够紧密地粘合而不留一点缝隙。

对于一个黑白设备，将半色调单元中的像素有的涂黑，有的涂白，这样就可以用它来逼近某种灰度色调。半色调单元的灰度级等于其白色像素数与像素总数之比，如果单元中有 n 个像素，那么它能够表现出 n+1 个不同的灰度级，要想得到一个特定的值在 0 到 1 之间的灰度值 g，只要将其中的 i 个像素涂白就可以了，其中 i=floor(g×n)。

对于构成像素的基色是全开或全关的(全黑或全白)彩色输出设备，前面的描述同样适用，对各彩色分量分别实施半色调技术，就产生这种彩色的对应灰度。

不管彩色分量原来是以加色(RGB 或灰度)方式还是以减色(CMYK)方式定义的，它都以加色方式进行半色调处理，彩色分量的值越大代表加色设备上颜色越亮，或者代表减色设备上油墨越少。变换函数以加色方式处理彩色值；见 21.4.1。

21.6.2 生成半色调屏幕

所有的实现过程中有两种生成半色调屏幕的基本方法：

——文件为每个彩色分量指定了屏幕频率、屏幕角度和点函数；这种方法是生成半色调屏幕的缺省方法，将在 21.6.2.1 中讨论。

——文件指定一个阀值(threshold value)向量，这组向量瓦片状(tiled)地覆盖整个设备像素空间；其中的阀值用来决定怎样还原中间彩色值，这种方法最适用于低分辨率的设备，将在 21.6.2.2 中讨论。

21.6.2.1 点函数 Spot Functions

注：为了便于说明，本条以在黑白设备上产生灰度级为例说明这种生成半色调屏幕的方法，同样的过程和算法也适

用于在彩色设备上还原中间彩色，即那些不能直接在设备上表示的彩色。

生成半色调空间的缺省方法要求为每个彩色分量设置频率、屏幕角度和点函数。半色调屏幕网格(grid)的属性有频率(每厘米的半色调单元的数目)和角度(网格线方向与设备像素坐标系的关系)。当一个单元想要表示灰度级在黑到白之间变化时，单元中各个像素的颜色按照已经定义好了的次序从黑到白变化。如果某种灰度包含一定的白色像素，那么亮一点的灰度必定包括更多的白色像素。对于递增的灰度级，像素从黑到白的改变次序在点函数中定义，这个函数定义为一个过程，点函数以一种间接方式描述了像素涂白的次序，这种方式与频率和角度的相互影响最小。

半色调单元本身有一坐标系：方块中央是原点，角点的 x，y 坐标为±1，在这个坐标系中，单元中每个像素坐标的 x，y 值都在－1 到 1 之间，把每个像素的坐标值作为操作数来调用点函数，返回一个－1 到 1 之间的数，它定义了该像素在序列中的位置。

点函数返回的值本身没有意义，像素涂白的顺序完全由不同像素的点函数值之间的关系来决定。当一个单元的灰度值在黑到白的范围内变化时，最先涂白的像素的点函数值是最小的(最负的)，下一个被涂白的像素的点函数值是次小的，依此类推。

注：如果两个像素具有相同的点函数值，它们的相互次序由具体实现决定。

21.6.2.2 阀值阵列 Threshold Array

生成半色调屏幕的另一种方法就是通过一个阀值阵列来直接控制半色调单元中的每个设备像素。这种方法适用于低分辨率设备，因为它提供了半色调还原的高层次控制，且允许半色调单元是一个矩形，而用点函数定义的半色调单元通常是正方形的。

一个阀值阵列就象一幅取样图像：它是像素值的矩形阵列。但是它完全在设备像素空间中定义，而且取样值通常是一些八位字节。像素值名义上代表灰度级：0 表示黑，255 表示白，重复使用阀值阵列就象用一片片瓦覆盖整个设备像素空间。这样，设备空间的每个像素映射到阀值阵列中的一个特定的取样值。

在每个像素非黑即白的二值设备上，使用阀值阵列的半色调技术是这样处理的：对于每个将要以一定灰度显示的设备像素，查找阀值阵列中与之对应的像素值，如果要显示的灰度值小于阀值阵列中的像素值，将该设备像素涂黑，否则涂白。0 到 1(闭区间)之间的灰度值对应于阀值阵列中 0 到 255(闭区间)之间的像素值。

这种模式可以很容易地推广到那些用多位来表示一个像素的单色设备上，例如，如果每个像素由 2 位来表示，那么每个像素可以直接表示四级灰度之一：黑，深灰，浅灰，白，分别以 0，1，2，3 编码。要想用设备像素能直接表示的四种灰度之外的一种中间灰度级来画一个设备像素，算法就查找阀值阵列的对应元素，以决定是用与之相邻的大的还是小的灰度级来表示它。

这种方法定义的半色调技术也可用于那些每个彩色分量都只有有限个值的彩色显示器。红、绿、蓝的值只是简单地分别当作灰度级来处理，对于各种色彩同样可使用阀值阵列。

21.7 半色调词典 Halftone Dictionary

Halftone Dictionary 是一个词典对象，它的各个项是半色调处理过程中的参数。图形状态中包括当前半色调词典，这个字典指出了成像操作符所使用的半色调过程。操作符 GetHalftone 的返回值为当前半色调词典；操作符 SetHalftone 将另一个半色调字典设置为当前字典，见 21.8。

每个半色调字典都必须有一个值类型为基数的 HalftoneType 项，它说明了半色调处理的主要类型，再根据类型来解释词典中的其他项。本标准中定义了三类标准半色调字典，下面一些条描述每个标准类型的半色调词典的内容。

注：本标准只说明了半色调字典中标准的必选项和可选项，具体实现时半色调词典中可以有本标准中未定义的其他项。

21.7.1 HalftonType 1 半色调词典

HalftonType 为 1 的半色调词典根据频率、角度和点函数定义了一个半色调屏幕(见 21.8)。通过操

作符 SetScreenFrequency、SetScreenAngle、SetSpotFunction 或 SetHalfton 可以把图形状态中的当前半色调词典赋值为一个 HalftoneType 为 1 的半色调词典。HalftoneType 1 半色调词典的内容在下面的条中描述。

21.7.1.1 HalftoneType(半色调类型)

必选项〈HalftonType:Cardinal〉的值必为 1。

21.7.1.2 Frequency(频率)

必选项〈Frequency:Number〉指出了在设备坐标系中每厘米的半色调单元数。

21.7.1.3 Angle(角度)

必选项〈Angle:Number〉指出了半色调屏幕相对于设备坐标系的旋转角度。

21.7.1.4 SpotFunction(点函数)

必选项〈Spotfunction:Procedure〉定义了为实现不同灰度/彩色值而调整半色调单元中的设备像素的顺序。这个过程取两个操作数:

〈y:Number〉

〈x:Number〉

并返回一个结果:

〈relval:Real〉

其中 x,y 是设备像素在半色调单元坐标系中的坐标值,relval 是一个在 −1.0 到 1.0 之间(包括)的实数,它定义了像素在序列中的位置,见 21.6.2.1 中有关于点函数语义的详细描述。

21.7.1.5 TransferFunction(变换函数)

〈TransferFunction:Procedure〉项可以修改状态变量 CurrentTransfer 的值,当 HalftonType 1 半色调字典作为 HalftoneType 3 半色调字典中一个对应于 NamedColour 彩色分量的元素时,该项是必选项,否则该项是可选项。这个过程取一个操作数:

〈d:Number〉

并返回一个结果:

〈t:Real〉

其中 d 是设备自身彩色(native colour)的一个彩色值,t 是真正传送给设备的值,见 21.5 中有关变换函数的详细描述。

21.7.1.6 ActualFrequency(实际频率)

可选项〈ActualFrequency:Number〉是一个其值由操作符 SetHalftone 设置的位置标志符〈placeholder〉。SetHalftone 可能对所要求的屏幕频率进行微小的调整,以保证包含像素的图案是固定不变的,把它作为一个屏幕单元重复地覆盖整个页面。

如果这一项在 HalftoneType 1 半色调字典中存在,对之执行 SetHalftone 操作符,则该操作符将用实际可以达到的频率来取代它。

21.7.1.7 ActualAngle(实际角度)

可选项〈ActualAngle:Number〉是一个其值由操作符 SetHalftone 设置的位置标志符。SetHalftone 可能对所要求的屏幕角度进行微小的调整,以保证包含像素的图案固定不变,把它作为一个屏幕单元重复地覆盖整个页面。如果这一项在 HalftoneType 1 半色调字典中存在,对它执行 SetHalftone 操作符,则该操作符将用实际可以达到的角度来取代它。

21.7.2 HalftoneType 2 半色调字典

HalftoneType 为 2 的半色调字典把屏幕定义为一个阀值阵列,这些阀值对半色调单元中的每个设备像素进行直接控制。用操作符 SetHalftone 可以将一个 HalftoneType 2 的半色调字典设置为图形状态中的当前半色调字典。HalftoneType 2 半色调字典的内容在下面条中描述。

21.7.2.1 HalftoneType(半色调类型)

必选项〈HalftoneType:Cardinal〉的值必为 2。

21.7.2.2 Width(宽度)

必选项〈width:Cardinal〉指出了以像素为单位阀值阵列的宽度。

21.7.2.3 Height(高度)

必选项〈Height:Cardinal〉指出了以像素为单位阀值阵列的高度。

21.7.2.4 Thresholds(阀值)

必选项〈Thresholds:String〉说明了阀值阵列,这个串必须包含有 width * Height 个元素,它从左下角开始(把这个矩阵阵列看作是铺在了设备像素空间上),沿着从左到右,从底向上的顺序描述了阀值阵列和阀值阵列中值的语义。

21.7.2.5 TransferFunction(变换函数)

项〈TransferFunction:Procedure〉可以修改状态变量 CurrentTransfer 的值,当 HalftoneType 2 半色调词典作为 HalftoneType 3 半色调词典的一个对应于 Named Color 彩色分量的元素时,该项是必选项;否则的话该项是可选项,这个过程取一个操作数:

〈d:Number〉

并返回一个结果:

〈t:Real〉

其中 d 是设备自身彩色的一个彩色值。t 是真正传送给设备的值,见 21.5 中关于变换函数的详细描述。

21.7.3 HalftoneType 3 半色调词典

HalftoneType 为 3 的半色调字典允许为任意数目的彩色分量定义单独的半色调屏幕。它包括一些项,其名字是设备彩色分量或 NamedColour 名,项的值是半色调字典。

设备彩色分量的标准名是 Grey、Red、Green、Blue、Cyuan、Magenta、Yellow 和 Black 中适当的几个名字。对于 NameColour 彩色空间,名字就是那些作为 SetColourSpace 操作符的 NamedColour 参数的名字。

项的值是类型 1 或 2 的半色调字典,它们为单彩色描述了半色调函数和变换函数,这种字典与那种适用于所有彩色分量的作为主半色调字典(即 SetHalftone 操作符的操作数)出现的字典形成了对照。

可以通过操作符 SetHalftone 将图形状态中的当前半色调字典设置为一个 HalftoneType3 的半色调字典,除了任意数目的可选择的彩色名/半色调字典对之外,HalftoneType 3 半色调字典必须含有两个必选项,它们在下面描述。

21.7.3.1 HalftoneType(半色调类型)

必选项〈HalftoneType:Cardinal〉的值必须为 3。

21.7.3.2 Default(默认值)

必选项〈Default:Dictionary〉是一个半色调字典,它为任何在 HalftoneType 3 半色调字典中没有自己项的彩色定义了半色调函数。

21.8 指定半色调函数

把当前半色调指定的信息保存在图形状态变量 CurrentHalftone 中;这个变量的值是一个半色调字典,它的 HalftoneType 和其他内容依赖于具体实现。这个状态变量的值可以由以下的一些操作符来改变,它们将在 21.10 中详细描述:

——SetScreenFrequency,SetScreenAngle,SetSpotFunction

如果当前半色调字典的 HalftoneType 为 1,这些操作符将修改字典中相应项的值。如果 HalftoneType 不为 1,那么状态变量 CurrentHalftone 的值被设置为一个 Halftone-Type 1 半色调字典。新的半色调字典的内容被设置成与具体实现有关的默认值,然后这些操作符就修改其中相应项的值。

——SetColourScreenFrequency,SetColourScreenAngle,SetColourSpotFunction

如果当前半色调字典的 HalftoneType 为 3,这些操作符将在字典中加入相应的项或者修改已存在

的相应项的值，这就象是在用操作符 SetScreenFrquenay、SetScreenAngle、SetSpotFunction 对于每个设备基色的基本半色调字典进行修改一样。如果当前半色调字典的 HalftoneType 不为 3，那么状态变量 CurrentHalftone 的值就设置为一个 HalftoneType 3 的半色调字典，新的半色调字典的内容根据不同的具体应用置为默认值，然后操作符增加并且/或者修改相应项的内容。

——SetHalftone

SetHalftone 操作符的半色调字典操作数被设置成新的当前半色调字典。

21.9 操作符

在定义这些操作符及其语义时，也说明了那些在解释执行操作符过程中可能会引起内容异常的情况。在第 24 章中定义了内容异常及异常处理，除了这些操作符特定异常外，还有一些几乎在所有操作符执行过程中都可能发生的普通异常，普通异常及其语义在 24.6.2 中描述。

21.9.1 CIE 彩色转换操作符

下面的操作符提供了从 CIE 相关彩色空间中的彩色到设备彩色空间中的彩色的转换过程控制。

21.9.1.1 SetColourRendering

操作符 SetColourRendering 取一个操作数：

〈crd：Dictionary〉

没有返回值。crd 应该是一个彩色还原字典，正如 21.3 所述。这个操作符的执行结果就是将成像状态变量 CurrentColourRendering 的值置为 Crd。

21.9.1.2 GetColourRendering

操作符 GetColourRendering 没有操作数，返回一个结果：

〈crd：Dictionary〉

其中 crd 是成像状态变量 CurrentColourRendering 的值，也就是图形状态中当前色的还原字典。

21.9.2 设备彩色转换操作符

下面的操作符提供了从 RGB 设备彩色到 CMYK 设备彩色的转换控制。

21.9.2.1 SetColourRemoval

操作符 SetColourRemoval 取一个操作数：

〈ucr：Procedure〉

没有返回值，ucr 应该是一个 UnderColourRemoval 过程（正如 21.4.3.1 中所述），该操作符的执行结果就是将成像状态变量 CurrentColourRemoval 的值置为 ucr。

21.9.2.2 GetUnderColourRemoval

操作符 GetUnderColourRemoval 没有操作数，有一个返回值：

〈ucr：Dictionary〉

其中 ucr 是成像状态变量 CurrentUnderColourRemoval 的值，见第 12 章。

21.9.2.3 SetBlackGeneration

操作符 SetBlackGeneration 取一个操作数：

〈bg：Procedure〉

没有返回值，bg 应该是一个 black generation 过程（正如 21.4.3.1 所述），该操作符的执行结果就是将成像状态变量 CurrentBlackGeneration 的值置为 bg。

21.9.2.4 GetBlackGeneration

操作符 GetBlackGeneration 没有操作数，有一个返回值：

〈bg：Dictionary〉

其中 bg 是成像状态变量 CurrentBlackGeneration 的值，见第 12 章。

21.9.3 变换函数操作符

下面的操作符提供了对还原模型中变换函数部分的控制。

21.9.3.1 SetTransfer(设置变换函数)

操作符 SetTransfer 取一个操作数:

〈t:Procedure〉

没有返回值,t 应该是一个变换函数的过程(正如 21.5.1 所述),该操作符的执行结果就是将构成成像状态变量 CurrentTransfer 的四个过程的值都赋为 t。

21.9.3.2 SetColourTransfer(设置彩色变换函数)

操作符 SetColourTransfer 取四个操作数:

〈k:Procedure〉

〈y:Procedure〉

〈m:Procedure〉

〈c:Procedure〉

没有返回值,操作数必须是变换函数过程(正如 21.5.1 所述),该操作符的执行结果就是将构成成像状态变量 CurrentTransfer 的四个过程的值置为它的操作数的值。

21.9.3.3 GetTransfer

操作符 GetTransfer 没有操作数,并有四个返回值:

〈k:Procedure〉

〈y:Procedure〉

〈m:Procedure〉

〈c:Procedure〉

其中 c,m,y,k 是图形状态中的当前变换函数过程(见 21.5.1)。在 CMYK 设备中这些过程分别指定了用于处理设备彩色 Cyan、Megenta、Yellow 和 Black 的变换函数,在 RGB 设备中,c、m、y 指定了用于处理红、绿、蓝色的变换函数。在灰度设备中,k 指出了用于处理灰度的变换函数。

注:如果成像变量 CurrentTransfer 的值是由操作符 SetTransfer 设置的,那么这四个返回值是一样的。

21.9.4 半色调函数操作符

下面的操作符提供了对还原模型中的半色调函数的控制。

21.9.4.1 SetScreenAngle(设置屏幕角度)

操作符 SetScreenAngle,取一个操作数:

〈a:Number〉

没有返回值。a 为所有半色调屏幕指定的一个半色调屏幕角。如果当前半色调字典的 HalftoneType 为 1,那么 SetScreenAngle 操作符修改字典中 Angle 项的值。如果当前半色调字典的 HalftoneType 不是 1,那么将状态变量 CurrentHalftone 的值置成一个 HalftoneType 1 的半色调字典,新的半色调字典的内容根据具体实现的不同置为默认值,Angle 项的值置为 a。

21.9.4.2 SetScreenFrequency(设置屏幕频率)

操作符 SetScreenFrequency 取一个操作数:

〈f:Number〉

没有返回值,f 为所有的半色调屏幕指定一个半色调屏幕的频率。如果当前半色调字典的 HalftoneType 为 1,那么 SetScreenFrequency 操作符修改字典中的 Frequency 项的值。如果当前半色调字典的 HalftoneType 不为 1,那么将状态变量 CurrentHalftone 的值置成一个 HalftoneType 为 1 的半色调字典。新的半色调字典的内容根据具体实现的不同置成默认值,然后将它的 Frequency 项置为 f。

21.9.4.3 SetSpotFunction(设置点函数)

操作符 SetSpotFunction 取一个操作数:

〈sf:procedure〉

没有返回值,sf 为所有的半色调屏幕指定了一个半色调屏幕点函数。如果当前半色调字典的 Halftone-

Type 为 1，那么 SetSpotFunction 操作符修改字典中的 Spotfunction 项；如果当前半色调字典的 HalftoneType 不为 1，那么将状态变量 CurrentHalftone 的值置成一个 HalftoneType 为 1 的半色调字典，新的半色调字典的内容根据具体实现的不同置为默认值，然后将它的 SpotFuntion 项置为 sf。

21.9.4.4 SetColourScreenAngle(设置彩色屏幕角)

操作符 SetColourScreenAngle 取 4 个操作数：

〈k:Number〉

〈y:Number〉

〈m:Number〉

〈c:Number〉

没有返回值，这四个操作数为每个设备基色指定了一个半色调屏幕角。操作符 SetColour-ScreenAngle 先确认当前的半色调字典的 HalftoneType 为 3，然后将它的操作数保存在相应彩色的 HalftoneType 1 半色调字典的 Angle 项中，对于 CMYK 设备，要被改动的 HalftoneType 1 半色调字典的项分别是 Cyan、Magenta、Yellow 和 Black；对于 RGB 设备，操作数 c、m、y 用来对 Red、Green、Blue 项进行修改；对于灰度设备，操作数 k 用来产生或修改项 Grey。

操作符 SetColourScreenAngle 完成如下逻辑功能：

——如果图形状态中的当前半色调字典的 HalftoneType 不为 3，则创建一个新的 HalftoneType 为 3 的半色调字典，根据具体实现不同设置它的默认值，并把它赋给成像状态变量 CurrentHalftone。

——对于 HalftoneType 3 半色调字典的每个相应项进行下面的操作：

——如果该项不存在，或者存在但它的值不是一个 HalftoneType 1 的半色调字典，那么创建一个项(如果需要的话)，然后将它的值置为一个新创建的由具体应用系统置成默认值的 HalftoneType 1 半色调字典。

——将这个 HalftoneType 1 半色调字典的 Angle 项的值置为适当的操作数。

21.9.4.5 SetColourScreenFrequency(设置彩色屏幕频率)

操作符 SetColourScreenFrequency 取 4 个操作数：

〈k:Number〉

〈y:Number〉

〈m:Number〉

〈c:Number〉

没有返回值，这四个操作数为每个设备基色指定了半色调屏幕频率。操作符 SetColourScreenFrequency 先确认当前的半色调字典的 HalftoneType 为 3，然后将它的操作数保存在相应彩色的 HalftoneType 1 半色调字典的 Frequency 项中；对于 CMYK 设备，要被改动的 HalftoneType 1 半色调字典的项分别是 Cyan、Magenta、Yellow 和 Black；对于 RGB 设备，操作数 c、m、y 用来对 Red、Green、Blue 项进行修改；对于灰度设备，操作数 k 用来产生或修改项 Grey。

操作符 SetColourScreenFrequency 完成如下的逻辑功能：

——如果图形状态中的当前半色调字典的 HalftoneType 不为 3，则创建一个新的 HalftoneType 为 3 的半色调字典，根据具体实现不同设置它的默认值，并把它赋给成像状态变量 CurrentHalftone。

——对于 HalftoneType 3 半色调字典的每个恰当的项进行下面的操作：

——如果该项不存在，或者存在但它的值不是一个 HalftoneType 1 的半色调字典，那么创建一个项(如果需要话)，然后将它的值置为一个新创建的由具体应用系统置成默认值的 HalftoneType 1 半色调字典。

——将这个 HalftoneType 1 半色调字典的 Frequency 项的值置为适当的操作数。

21.9.4.6 SetColourSpotFunction(设置彩色点函数)

操作符 SetColourSpotFunction 取四个操作数：

〈k:Procedure〉

〈y:Procedure〉

〈m:Procedure〉

〈c:Procedure〉

没有返回值，这四个操作数为每个设备基色指定了半色调屏幕点函数。操作符 SetColourSpotFunction 先确认当前的半色调字典的 HalftoneType 为 3，然后将它的操作数保存在相应彩色的 HalftoneType 1 半色调字典的 SpotFunction 项中；对于 CMYK 设备，要被改动的 HalftoneType 1 半色调字典的项分别是 Cyan、Magenta、Yellow 和 Black；对于 RGB 设备，操作数 c、m、y 用来对 Red、Green、Blue 项进行修改；对于灰度设备，操作数 k 用来产生或修改项 Grey。

操作符 SetColourSpotFunction 完成如下的逻辑功能：

——如果图形状态中的当前半色调字典的 HalftoneType 不为 3，则创建一个新的 HalftoneType 为 3 的半色调字典，根据具体实现不同设置它的默认值，并把它赋给成像状态变量 CurrentHalftone。

——对于 HalftoneType 3 半色调字典的每个相应的项进行下面的操作：

——如果该项不存在，或者存在但它的值不是一个 HalftoneType 1 的半色调字典，那么创建一个项（如果需要的话），然后将它的值置为一个新创建的由具体应用系统置成默认值的 HalftoneType 1 半色调字典。

——将这个 HalftoneType 1 半色调字典的 SpotFunction 项的值置为适当的操作数。

21.9.4.7 SetHalftone（设置半色调字典）

操作符 SetHalftone 取一个操作数：

〈htd:Dictionary〉

没有返回值，执行 SetHalftone 操作符之后成像状态变量 CurrentHalftone 的值被置成半色调字典操作数 htd，并将该字典置为只读。

注：如果 htd 中包含有用于设备基色的 TransferFunction 项，那么成像状态变量 CurrentTransfer 的值中相应的变换函数被更新为新的变换函数的值。

21.9.4.8 GetHalftone（取半色调字典）

操作符 GetHalftone 没有操作数，有一个返回值：

〈htd:Dictionary〉

其中 htd 是图形状态中的成像状态变量 CurrentHalftone 的值所指出的半色调字典。

注：用于设备基色的变换函数一般都存放在成像状态变量 CurrentTransfer 中，虽然 htd 中可能包括曾经用来设置成像状态变量 CurrentTransfer 值的项 TranferFunction，但是在文件中需要取得当前变换函数时，应当用操作符 GetTransfer 来得到正确的当前变换函数。

22 过滤器

在特殊的数据转换过程中，过滤器（Filter）非常有用，特别是光栅图形图像操作中源数据的压缩。在 SPDL 中，这种类型的数据转换以过滤器的方式提供。过滤器的一个重要特性是分层能力，或者说流水线处理。流水线过滤器之间的连接以流对象（STREAM OBJECTS）格式提供。本标准指定过滤器、流对象、标准过滤器的语义，并且给出了构造新的流对象和流水线过滤器的 FILTER 操作符的语义。

22.1 过滤器模型

过滤器是通过 FILTER 操作符作用于一个文件的，这个操作符把一个数据源（见 22.1.1 中的数据源定义，这不是一个数据源资源）、过滤器所用参数（如果需要的话）和过滤器名作为操作数，然后 FILTER 操作符返回过滤了的流对象，它能作为随后的 FILTER 操作符或其他操作符的操作数（见第 17 章）。

22.1.1 过滤器的数据源

当一个操作符为了得到数据去访问一个过滤后的流对象时，可以把与该流对象相联系的过滤器流水线中的第一个数据源作为访问数据。数据按序通过流水线中的所有过滤器，最终过滤出的数据提供给存取流对象的操作符。

过滤器有三种类型的数据源——流对象、过程和八位字节串。下面的条将叙述它们是如何用作为一个过滤器数据源的。

22.1.1.1 流对象 StreamObjects

通常希望文件中过滤器最普通的数据源是流对象。流对象可以通过以下两种方式之一获得：

——FindResource 操作符能在结构中查找说明的数据源资源，并返回与这个数据源相关的流对象。

——由先前运行 Filter 操作符所建立的流对象可作为随后运行的 Filter 操作符的操作数。

由 Filter 操作符返回的流对象可作为数据转换的输出结果，又可作为随后执行的 Filter 操作符的流对象操作数，因此过滤器的流水线处理是可行的。所以，一个过滤器的输出又可作为另一个过滤器的输入。

22.1.1.2 过程

过滤器的数据流可以是过程(Procedure)。当过滤器要求更多数据转换时，它就调用过程。过程必须在操作符数中返回包含任意数个字节数据的字节串类型的对象，然后过滤器从栈中弹出这一八位字节串，并把它的内容作为过滤器的输入。

这一过程将重复执行，直到过滤器遇见数据结束条件(见 22.1.2)时为止。最后一个八位字节串中的剩余数据将被丢弃。过程将返回长度为零的八位字节串，它指明不再有有效数据。

22.1.1.3 八位字节串 OctetString

若过滤器的数据源是一个八位字节串，则过滤器简单地将其作为编码数据。若过滤器遇见结束条件(见 22.1.2)，则将丢弃剩余的字节串。否则，它将继续直到用完八位字节串的全部内容。

22.1.2 数据结束 End-of-Data

过滤器遇见不能继续过滤数据的状态，就称该状态为数据结束条件(EOD-End-Of-Data)。多数过滤器能识别数据中经编码的 EOD 标记。该标记的性质依赖于过滤器。在某些情况下，EOD 条件依赖于预定信息，例如以一个字节计数或扫描行计数来代替编码数据中的显示标记。每个过滤器的 EOD 条件在 22.2 的过滤器描述中指定。

22.1.3 过滤器名

不同于在结构中已参数化的 ColourSpaceObjects，过滤器须在运行时对内容进行参数化。通过执行 Filter 操作符，由程序控制过滤器参数化。这就允许予定义的过滤器资源在具有不同参数组的内容中重复使用多次。过滤器的名字可以通过执行带有一个操作数的 FindResource 操作符来获得，而该操作符是一个 INTERNAL IDENTIFIER 的值，它是由先前的 RESOURCE DECLARATION 把它同过滤器资源结合在一起的。下面一些条所描述的各个过滤器名是在 RESOURCE DECLARATION 中与 INTERNAL IDENTIFIER 值相关的。

22.2 标准过滤器 Standard Filters

下面一些条将叙述由本标准定义的标准过滤器的语义、名字和参数。某些过滤器的算法已标准化(如 CCITTFAX 译码)，另一些过滤器的算法是基于计算机工业广泛使用的事实上的标准(如 ASCII85 译码)。合适的工具可提供本标准尚未定义的附加过滤器。

22.2.1 过滤器/ASCII16 进制译码 Filter/ASCIIHexDecode

ASCIIHerDecode 过滤器不需参数。该过滤器对八位字节的 ASCII 编码进行译码，并生成二进制数据。对每一对 ASCII 编码的 16 进制数(0～9 和 A～F 或 a～f)，它将第一个数解释为一个八位字节的最高有效 4 位，第二个数解释为最低有效 4 位。这样，每二个八位字节输入，只产生一个八位字节输出。所有空白字符(空格，Tab，换行，回车，换页，空白)将被忽略。ASCII 字符“>”表示为 EOD。任何别的字符将使过滤器产生 DataError 异常。

当已读过奇数个 16 进制数，若过滤器遇到了结束符 EOD，过滤器将认为读到了一个附加的“0”数字。

22.2.2 过滤器/ASCII 85 译码 Filter/ASCII85Decode

ASCIIDecode 过滤器不需参数。该过滤器对基于 ASCII85 编码的数据进行译码，并生成二进制数——每四个八位字节的输入数据生成 5 个二进制的字节串。输入数据编码如下：

二进制数据的八位字节串以 4 元组(4-tuples)编码(4 个一组)。每个 4 元组用以生成一个 ASCII 码字符的 5 元组。若二进制 4 元组为($b_1b_2b_3b_4$)且 5 元组为($c_1c_2c_3c_4c_5$)，则它们之间的关系为：

$$(b_1\times256^3)+(b_2\times256^2)+(b_3\times256)+b_4=(c_1\times85^4)+(c_2\times85^3)+(c_3\times85^2)+(c_4\times85)+c_5$$

换句话说，四个二进制数据的八位字节可解释为一个基为 256 的数，且再转换成基为 85 的数。这个数字的 5 位数 $c_1c_2c_3c_4c_5$，通过每个数加 33(表示 ASCII 码“!”)，转换成 ASCII 码字符。因此，从！到 u 范围内的 ASCII 码字符均可使用，！表示值 0，u 表示值 84。作为一个特殊的情况，若所有 5 位数都为 0，那可用单个字符 z 来代替!!!!!。

所有空白字符(空格，Tab，回车，换行，换页，空)将被忽略。若过滤器在输入过程中遇到“～”字符，下面字符一定为“>”，过滤器将遇到结束符 EOD。任何别的字符将使过滤器产生 DataError 异常。而且在 ASCII85 编码中不可能组合成的任何字符序列，将产生输入/出错(ioerror)。

22.2.3 过滤器/行程译码 Filter/RunlengthDecode

Filter/RunlengthDecode 过滤器不需参数，该过滤器以行程编码格式译码数据。编码数据由一系列运行元素(runs)组成，每一运行元素(run)由后面跟着 1 到 128 个八位字节数据串的一长度字节组成。若长度在 0～127 之间，则长度＋1 个(在 0 和 128 之间的)八位字节串将被逐个拷贝到过滤器的输出中，若长度范围在 129 到 255 中，则紧跟的单个八位字节将重复拷贝 257—长度次数(在 2 和 128 之间)到输出中。长度为 128 的运行元素 run，则标志结束状态 EOD。

22.2.4 过滤器/CCITT 传真译码 Filter/CCITTFaxDecode

CCIIIFaxDecode 过滤器需一系列参数，它们以词典形式给 Filter 操作符。过滤器支持组 3 和组 4 (Group3 & Group4)的编码方案，它们在 CCITT 蓝皮书(卷 V11.3，1988)中的建议 T.4 和 T.6 中说明。编码数据格式和译码算法也在这些建议中说明。若过滤器遇到不正确的编码源数据，将引起 DataError 异常。过滤器不进行任何错误纠正。

注：SPDL 支持建议 T.4 和 T.6 仅局限于数据译码，不包括实际传真机上进行通信所要求的连接初始化和应答协议。过滤器的目的是为 SPDL 文件提供有效地传真二值(bi-level)取样图像的能力。

22.2.4.1 K

可选项〈K：Integer〉指示用来编码源数据的编码表。负数表示二维编码(Group4)，0 表示一维编码(Group3，1-D)，正数表示一维和二维混合编码(Group3，2-D)。若 K 没有在词典中指明，则默认值为 0。

22.2.4.2 行结束

可选项〈EndOfLine：Boolean〉指出编码后的每行源数据前面是否已冠以行结束位(end-of-line bit)。若在词典中并不存在 Endofline，则过滤器所用缺省值为假。

22.2.4.3 EncodedByteAlign

可选项〈EncodedByteAlign：Boolean〉指出编码后的源数据每行前面是否有一个附加 0 位。若在词典中并不存在 EndOfLine，则默认值为假。

22.2.4.4 列

可选项〈Columns：Cardinal〉指出图像未来宽度。若词典中不存在 Columns，则过滤器所用默认值为 1728。

22.2.4.5 行

可选项〈Rows：Cardinal〉指出图像扫描行的高度。若此参数为 0，则图像高度未事先确定。结束块位模式(end-of-block bit pattern)或过滤器的 EOD 条件将中止编码数据。若词典中不存在 Rows，则默认值

为 0。

22.2.4.6 块结束

可选项〈EndOfBlock:Boolean〉指出编码源数据是否附加了块结束位。EndOfBlock 的值为真，将忽略行(Rows)参数。若词典中不存在 EndOfBlock，则默认值为真。

块结束方式是适合于 K 参数的 CCITT 传真块结束(EOFB--end-of-facsimile-block)或控制结束(RTC--return-to-control)。

22.2.4.7 Blackls 1

可选项〈Blackls 1:Boolean〉，若值为真，则使值为 1 的位转换为黑色像素；值为 0 的位转换为白色像素。若词典中不存在 Blackls 1，则默认值为假。

22.2.5 过滤器空译码

Filter/NullDecode 过滤器不执行数据转换。输入结果就是输出结果。因为它不需要检测 EOD 标志，所以这种过滤器能够处理任意数据源(过程或八位字节串)。

NullDecode 过滤器需要两个参数：

〈EODstring:Octetstring〉

〈EODcount:Cardinal〉

它们指出了过滤器识别 EOD 的条件。若结束条件计数值(EODcount)为正(且 EODstring 的长度不为 0)，则过滤器将允许数据通过，直到遇见 EODstring 的所有具体值也已通过过滤器。

若 EODcount 为 0(且 EODstring 的长度不为 0)，则 EODstring 的第一个出现的值就产生 EOD。这种情况下，过滤器使用 EODstring，但它不通过过滤器。

在各种情况下，EODstring 的重叠情况不能被识别。例如，在输入 XeeeX 时，eee 的 EODstring 仅能识别一次 EODstring 的长度也可为 0，在这种情况下，若 EODcount 的值为正，则 EOD count 个八位字节的任意数据将简单地通过过滤器，并作为它的输出。若 EODcount 的值为 0，则禁止检测 EOD 标记。过滤器不会遇到 EOD，这主要用于以过程或八位字节串作为数据源的情况。

22.3 操作符

Filter 操作符和它语义的说明包含了在操作符的解释过程中可能引起内容异常的条件的说明。内容异常和异常的处理在第 24 章中定义。除了这些操作符特定异常外，还有几乎所有操作符的解释过程中均可能产生的普通异常。这些普通异常和它们的语义在 24.6.2 中叙述。

22.3.1 Filter

Filter 操作符取两个或更多的操作数：

〈filter:Name〉

〈$param_n$:Cardinal〉

…

〈$para_1$:Cardinal〉

〈$param_0$:Carinal〉

〈datesource:StreamObject 或 Procedure 或 OctetString〉

并返回一个结果：〈stream:StreamObject〉

这里的 filter 是通过执行 FindResource 所得到的过滤器名，Datasource 就是相应过滤器编码的输入数据源，插入参数的个数和属性(如果存在的话)依赖于过滤器。结果是根据 Filter 的值和 filter 操作符的其他参数过滤出的一个新的 StreamObject。本标准定义的标准过滤器和它们的参数(如果存在的话)在 22.2中定义。

23 模板

模板(Form)是一种任意图形、文本或取样图像的自包含描述(Self-contained)，它们可以成像多次

——每次在不同的页面上，或者在一个页面的不同位置上。

23.1 模板模型

模板由 RESOURCE DEFINITION 在结构层定义，并且在一个 RESOURCE DEFINITION 中与 INTERNAL IDENTIFIER 的值相联系。模板的外部特性由执行 imagingoperators 的过程来描述，模板内容可通过执行 FindResource 操作符来指出，该操作返回模板对象 FormObject，并把图案对象作为操作数去执行 ExecuteForm 操作符，从而形成图像。

下面将说明模板词典的内容、模板执行的语义和 ExecuteForm 操作符的语义。

23.2 模板词典

模板由模板对象 Form Object 来定义，一个具有指定内容的专用词典完整地叙述了模板。模板在它自己的坐标系——模板坐标系中重定义，该坐标系由模板对象中与每次执行模板 PaintProc（模板对象中成像过程）时的 CurrentTransformation 相关的 Transformation 的值来定义。模板对象中的 BBOX 参数可以在模板坐标系中进行解释；PaintProc 也在该坐标系中执行。

下面一些条叙述模板对象的内容。除了叙述的内容以外，模板对象也可以包含 PaintProc 所需要的其他（常量）信息（见 23.2.3）。

23.2.1 BBox

必选项〈BBox：Vector〉是一个类型为数值的 4 个元素的向量，它们指出了在模板坐标系中模板的边界范围（Bounding Box），该边界可以用来对模板进行裁剪。向量的 4 个元素指出了模板边界的左下角坐标 X，Y（lower left X，lower left Y）和右上角的坐标 X，Y（upperright X，upper right Y）：

23.2.2 Transformation

必选项〈Transformation：Transformation〉是与当前变换〈CurrentTransformation〉相关的一个变换〈Transformation〉，它用来形成模板坐标系中的变换（Transformation）。

23.2.3 PaintProc

必选项〈PaintProc：Procedure〉指出了一个用来涂色模板的过程。PaintProc 取一个操作数：

〈FormDict：Dictionary〉

这里 FormDict 是一个包含 PaintProc 的模板对象，无返回结果。执行 PaintProc 的结果是去画一个模板实例。执行 PaintProc 的上下文在 23.4 中叙述。

如果 PaintProc 执行 ImageRasterElement 或 MaskBitMap 操作符，对这些操作符没有任何潜在的数据源 DataSource/Document。如果任何潜在数据源是来自于一个外部数据源的 StreamObject，那么为求得 StreamObject 执行 FindResource 操作符必须在 PaintProc 中出现。

注：由于在许多实现过程中希望通过保留最近所用模板的高速缓存（cache）来优化执行，PaintProc 可以在无法预测的时间和环境中执行，因此 PaintProc 将仅仅依赖于 SystemDict 中的操作符和 FormDict 操作数中的内容，且总是形成同样的结果。它没有超出涂色模板的其他附带效应。

23.2.4 UniqueID

可选项〈UniqueID：StructureName〉用来唯一标识可由多个文件使用的模板，以便进行高速存储。

注：为了避免在高速存储的实现过程中可能浪费存储器资源，在所有单个文件中和指定模板的模板对象中将不包括这一项。

23.2.5 Implementation

项〈Implementation：Any〉是由 ExecuteForm 操作符加入到模板对象（FormObject）中。通过使用 Implementation 来存储信息以便模板的高速存储。这一项的类型和值与具体实现有关。如果一个名字为 Implementation 的项又通过定义词典的 TOKENSEQUENCE 来把它放入模板对象，它的值可以通过执行 ExecuteForm 操作符来重写。

23.3 为文件内容建立模板词典

用来作为操作符 ExecuteForm 的操作数的模板词典，可以通过执行 FindResource 操作符来求得。

FindResource 操作符把 INTERNAL IDENTIFIER 作为它的操作数，而该值事先在结构中由一个 RESOURCE DECLARATION 把它与模板资源联系在一起了，并返回一个模板对象。

注：FindResource 操作符并不去检查模板对象，以保证所需要的项是存在的且类型正确；而这一功能由 ExecuteForm 操作符来实现。

23.4 模板成像

通过执行 ExecuteForm 操作符可以使模板成像，而它们是在模板对象中通过不断执行 PaintProc 实现的。在执行 ExecuteForm 操作符以前，一个文件首先应该在图形状态中设置相应的参数；特别是它应该改变 CurrentTransformation，以便控制在用户坐标系中模板的位置，大小和方向。

在执行 PaintProc 以前，下面的一些成像状态变量被置成它们的默认值（对于每一个默认值请见第 12 章）：

——CurrentColor（当前色）

——CurrentColorSpace（当前彩色空间）

——CurrentDashPattern（当前线型）

——CurrentMitreLimit（当前斜接的限定值）

——CurrentPath（当前路径）

——CurrentPosition（当前点位置）

——CurrentStrokeEnd（当前线端）

——CurrentStrokeJoin（当前线交方式）

——CurrentStrokeWidth（当前线宽）

执行环境的其他部分与在执行 PaintProc 时的具体实现有关。

注：在某些实现过程中，执行环境的部分或所有部分均可以从执行 ExecuteForm 操作符时的当前上下文中继承。

23.5 操作符

ExecuteForm 操作符和它们语义的说明包括了对那些操作符的解释结果可能产生异常的条件的说明。内容异常和异常处理在第 24 章定义。除了这些操作符特定异常外，还有在几乎所有的操作符解释过程中均可能产生的普通异常，这些普通异常和它们的语义在 24.6.2 中叙述。

23.5.1 ExecuteForm

ExecuteForm 操作符接受一个操作数：

〈FormObject：Dictionary〉

无结果返回。ExecuteForm 所生成的图像输出由模板对象的 PaintProc 过程（见 23.2.3）来定义。如果对于一个模板是第一次调用 ExecuteForm，那 ExecuteForm 首先证明词典包含所需要的信息，然后增加 Implementation 项（见 23.2.5），且把词典的访问属性至少降为 ReadOnly。

注：由于在某些实现过程中，ExecuteForm 操作符可以改变词典的内容，词典访问属性的降低可以防止后继的访问对词典内容的修改。Implementation 希望不允许任何后继操作对词典内容进行访问（除了执行 ExecuteForm 以外），可以把词典的访问属性降为 ExecuteOnly。

当 ExecuteForm 需要去调用 PaintProc 过程（而不是使用高速存储的图像模板描述）时，它把 FormObject 压入操作数栈，然后执行以下一些操作：

```
SaveGraphicsState
SetStateVariables          % Reset appropriate state variables
                           % to their default values(see 20.4;
                           % Procedure SetStateVariables not
                           % expanded here)
Dup/Transformation Get Concat
                           % Concatenate form's Transformation
```

```
                        % with CurrentTransformation
NewPath
Dup /BBox Get           % Get form bounding box(BBox)
BBoxToPath              % Use BBox to construct rectangular
                        % path (Procedure BBoxTOPath not
                        % expanded here)
ClipPath                % Clip to form bounding box
NewPath
Dup/PaintProc Get Execute       % Execute PaintProc
RestoreGraphicsState
```

注：PaintProc 过程希望去使用词典操作数且执行一系列对模板进行涂色的成像操作符。在给出了相同的图形状态参数以后，PaintProc 必须生成相同的结果，它与调用的次数无关，且与（例如）UserDict 的内容无关。文档将不希望任何特定的执行 ExecuteForm 将去运行指定的 PaintProc。

24 异常处理

在结构处理和内容处理过程中，可能出现各种异常条件。本章将定义可能出现的一组异常以及 SPDL 表示过程中处理异常的机制。

结构异常的产生和处理在 Block 层，可能引起的异常在 24.1 中定义，结构异常的处理在 24.2 中定义。内容异常在 24.3 中定义，内容异常处理在 24.4 中定义。有关内容异常和内容异常处理的操作符在 24.5 中定义。解释器错误可在内容中引起一种特定类型异常条件，这些在 24.6 中定义。

24.1 结构异常

结构异常出现在一个 Block 内部，可能是以下几种情况之一引起：

——隶属于 Block 的一个结构出现语法错；

——隶属于 Block 的一个结构出现非法描述；

——隶属于 Block 的一个结构出现 DPI 限制的违反；

——隶属于 Block 的一个结构出现结构警告；

——隶属于 Block 的一个结构出现实现限制的违反；

——隶属于 Block 的内容中出现未处理内容异常；

——隶属于 Block 的一个 Block 中出现未处理结构异常。

上述任何一种情况出现，则在出现上述情况的结构的直接上级 Block 中将会引起一个异常。

24.1.1 结构中的语法错

当结构处理器在隶属于 Block 的结构中遇到了本标准中规定的语法违反（见 25.2 和 25.4）时，则 Block 产生语法错。

24.1.2 结构中的非法描述

当结构处理器在一个隶属于 Block 的结构中遇到了本标准中不允许出现的特性值时，则产生非法描述异常。

24.1.3 DPI 限制的违反

当结构处理器在一个隶属于 Block 的结构中遇到 DPI，且由它指定的特性值与上级 Block 中 DPI 指定的值相冲突时，则一个 Block 将产生 DPI 限制违反。

24.1.4 实现限制的违反

当结构处理器在隶属于 Block 的结构中遇到了在实现过程中并不支持的特性值时，则产生实现限制违反。

注：在一个合适的实现过程中，仅仅在遇到本标准所定义的特性值时（见第 26 章）才会出现这一异常。

24.1.5 内容中的未处理内容异常

当内容处理器在隶属于 Block 的内容中遇到一个未处理内容异常时(见 24.3 和 24.4 的内容异常和内容异常处理),则 Block 产生未处理内容异常。

24.1.6 下属 Block 中的未处理结构异常

当在一个 Block 中出现了上述五种结构异常中的任何一种时,如果该 Block 未处理这个异常(见 24.2)。则最直接的上级 Block 中也将出现同样的结构异常。

若未处理结构异常出现在一个 DOCUMENT 中(该文档没有上级 Block),那么该异常就采用 24.2.2中指定的方法处理。

24.2 结构异常的处理

每个 Block 有一个其名字为 DPI/AbortPolicy 的 DPI 特性,它有 onWarning(警告值)onError(错误值)或 StruggleOn(冲突值)(见第 8 章)。(注意:连同别的 DPI,在 PAGE 以下将省略这个 DPI 特性值的显式说明。)

如果一个结构警告出现在 DPI/AbortPolicy 的值为 onError 的 Block 中,则在典型情况下,它不会出现结构异常,且根据具体实现方式将错误信息打印出来。

如果一个结构警告出现在 DPI/AbortPolicy 的值为 onWarning 的 Block 中,则除了打印错误信息以外,还会产生结构异常。

下面的一些条将叙述结构异常的处理。

24.2.1 struggleOn 冲突

如果一个结构异常出现在 DPI/Abort Policy 的值为 struggleOn 的 Block 中,则该异常已被处理并发出警告信息。

注:这一警告信息可通过打印规程进行传递,或可打印在被中断的页上或标题页上。

隶属于 Block 的内容和结构不再由表示过程进一步处理。然而,如果出现异常的 Block 是一个 PAGE,则 PAGE 就被表示。在页面上所表示的图像将与具体实现有关。

紧跟出现过异常的 Block 的结构处理就立即恢复。如果异常出现在下属于一个 PAGE 的 Block 中,那逐渐积累的结果页面图像将与具体实现有关。

注:希望结果页面图像尽可能地与在文档中并不存在异常的表示过程中所产生的结果一样地好。

24.2.2 onError,onWarning

如果一个结构异常出现在 DPI/AbortPolicy 特性值是 onError 或 onWarning 的 Block 中,则说异常未被处理,发出一个与具体实现有关的警告信息。

注:这一警告信息可通过打印规程进行传递,或可打印在被中断的页上或标题页上。

隶属于 Block 的内容和结构不再由表示过程进一步处理。然而,如果出现异常的 Block 是一个 PAGE,则 PAGE 就被表示。在页面上表示图像与具体实现有关,先于该页的所有页面将有选择地表示出来。就象 PAGE 中引发的异常并不存在一样。

当在一个 Block 中出现未处理异常时,则在最直接的上级 Block 中也将引起同样的结构异常,因此在上级 Block 中也将产生由本条所定义的效果。

如果未处理结构异常出现在一个 DOCUMENT 中(该文档中无上级 Block),则文档处理就结束。如果 DOCUMENT 是一简单 PAGE,则就表示页面图像;在该页面上表示图像将与具体实现有关。

24.3 内容异常

内容异常出现在内容处理过程中,是执行下述任何一种操作符的结果所引起的:RaiseException 和(在某些情况下)RaiseError 或 RaiseWarning。

执行 RaiseException 操作符会立即引发内容异常,该异常可以在内容中(见 24.4)被处理或不处理。如果未被处理,则在最直接上级 Block 中就出现未处理内容异常的结构异常。

执行 RaiseError 操作符本身并不引发内容异常。而是从一个名字为 ErrorDict 的词典中选择一个

error 过程，然后执行这一 error 过程。Error 过程一般由执行 RaiseException 操作符来结束，这样就会引发一个内容异常（见 24.6 中有关 error 过程和它们的使用的完整说明）。

执行 RaiseWarning 操作符一般并不引发内容异常，而是以与具体实现有关的方式发出一个警告信息给打印请求者。然后，每个 Block 有一个命名为 DPI/AbortPoliy 的 DPI 特性，其值可以是 onWarning 或 onError。在 DPI/AbortPolicy 特性值为 onWarning 的上下文中，在发出了警告信息 OntentWarning 以后，解释器出现错误，从而引发异常（见 24.6）。

24.4 内容异常的处理

内容异常是由执行 RaiseExcection 操作符的结果产生的。根据执行这一操作符的上下文情况，异常可能被处理或未被处理。如果在解释一个由 ExecuteTrapped 操作符所调用的过程中出现了异常，则异常将被处理，否则它就不被处理。

24.4.1 异常处理

ExecuteTrapped 操作符提供了在一个上下文中执行一个过程的机制，而在该上下文中内容异常将作为陷阱来处理。一个异常出现，所有过程和子过程的执行将停止，内容处理将对紧跟在 Execute-Trapped 操作符以后的词法继续进行分析，在操作数栈中将设置一个布尔值 true，以表示该异常为陷阱。（如果一个过程中无异常发生，且已被完整地解释，则在操作数栈中设置的布尔值为 false。）

在 ExecuteTrapped 操作符返回的布尔值后面可以紧跟一个处理异常的过程，而该过程由文档进行定义，且当一个异常出现时，可以有条件地被执行。

如果一个异常是陷阱且以这种方式进行处理，则其结果完全在文档的控制之下。

24.4.2 未处理异常

如果在一个上下文中出现的内容异常并未作为异常陷阱，则内容异常就未被处理，那就执行来自 ErrorDict 的过程 ReportErrorInfo，然后就在直接上级 Block 中出现一个结构异常。

24.5 内容异常操作符和过程

这些操作符和他们语义的描述包括了对某些条件的描述，而这些条件在操作符的解释过程中可能会引发内容异常。内容异常和异常处理在第 24 章中定义。除了这些操作符特定异常外，还有一些几乎所有操作符的解释过程中都能引发的普通异常，这些普通异常和它们的语义在 24.6.2 中叙述。

24.5.1 RaiseException（引发异常）

RaiseExeception 操作符无变量，且不返回结果。它可以不管词法关系，动态地中止包含陷阱上下文（trapped context）实例的程序的执行。陷阱上下文是一个由 ExecuteTrapped 操作符调用的一个过程。陷阱上下文的使用在 24.4.1 中讨论，ExecutiveTrapped 操作符本身在 24.5.5 中叙述。

24.5.2 RaiseError（引发错误）

RaiseError 操作符取一个操作数：

〈error：Name〉

无结果返回。执行 RaiseError 操作符的结果是去执行一个在 ErrorDict 中的与 error 有关的过程。

24.5.3 RaiseWarning（引发警告）

RaiseWaring 操作符取一个操作数：

〈waring：OctetString Reference〉

无结果返回，执行 RaiseWarning 操作符的结果是去执行一个操作数为 Warning 的 Print 操作符，然后有条件地引起一个 ContentWarning 解释器错。是否引起解释器错由当前的 DPI/AbortPolicy 决定；见 24.3。

24.5.4 Print（打印）

Print 操作符取一个操作数：

〈message：OctetString Reference〉

无结果返回。它能使所发信息以与具体实现有关的方式传送给打印的请求者。

24.5.5 ExecuteTrapped(执行陷阱)

ExecuteTrapped 操作符取一个操作数:

〈p:procedure〉

并返回一个结果:

〈trapped:Boolean〉

ExecuteTrapped 执行过程 P(类似 Execute 操作符)。如果运行 P 到正常结束,则 ExecuteTrapped 在操作数栈返回一个结果值 false。如果 P 过早地结束,它类似于执行 RaiseException 操作符的结果,ExecuteTrapped 在操作数栈返回结果 true。不管结果如何,xecuteTrapped 以后,解释器将按正常顺序恢复执行下一个对象。

24.5.6 StoreErrorinfo(存储错误信息)

在 ErrorDict 中的 StoreErrorInfo 过程取一个操作数:

〈errorname:Name〉

返回一个结果:

〈errorname:Name〉

当出现一个解释器错误时,解释器就执行这一过程;它记录有关 error 和出现该 error 上下文的有关信息入 ErrorInfoDict(见 24.7)。

缺省的 StoreErrorInfo 过程将置 newerror 的值为 true,然后设置 errorname,command,ostack 和 cstack 为相应值(见 24.7)。文档可以重新定义这一过程的值,但必须保证能正确解释后继的文档错误,新的 StoreErrorInfo 过程必须在 ErrorInfoDict 中设置 newerror 项为 true,并必须保证在执行新的 StoreErrorInfo 过程后,在堆栈中留下 errorname。

如果缺省的 StoreErrorInfo 过程由文档在内容中显式执行,则就出现一个解释器错误。

注:由于所希望的 StoreErrorInfo 总是可以在当前执行上下文中访问到,所以 ErrorDict 中的 StoreErrorInfo 项的精确拷贝可以由解释器把它保存到系统词典 SystemDict 中。

如果 StoreErrorInfo 的值被重定义,那么重新定义 ReportErrorInfo 过程的值是必要的。

24.5.7 ReportErrorInfo(报告错误信息)

在 ErrorDict 中的 ReportErrorInfo 过程无操作数,且不返回结果。对于文档未处理的异常,解释器将执行这一过程,它设定 ErrorInfoDict 中 newerror 项的值为 false,然后根据 ErrorInfoDict 中的信息,以与具体实现有关的方式发出一个错误信息给打印请求者(见 24.7)。

一个文档可以重新定义 ReportErrorInfo 的值,但必须保证能正确解释后继的文档错误,新的 ReportErrorInfo 过程必须设置 EerrorInfoDict 中的 newerror 项为 false。

如果执行一个并非来自内容异常处理过程的缺省 ReportErrorInfo,则其结果与具体实现有关,且毫无意义,但也不会出现解释错误。

注:由于所希望的 ReportErrorInfo 总是可以从当前执行上下文中访问到,所以 ErrorDict 中的 ReportErrorInfo 项的精确拷贝可以由解释器把它保存到 SystemDict 中。

24.6 解释器错误

解释器错误仅仅在解释一个内容中类型为操作符的对象时才可能出现。解释器错误可能是由各种情况引起的,例如除数为 0 或语法错。

当一个解释器错出现的时候,内容处理器首先试图把操作数栈的内容恢复到解释执行引起错误的操作符以前的状态,这种企图的结果与具体实现有关。其次,它把操作符对象压入操作数栈。最后,在操作数栈装入解释器错的名字(见 24.6.2),并执行 RaiseError 操作符。RaiseError 操作符直接由解释器执行,并不要对名字 RaiseError 执行任何查找工作。

内容处理器的后继活动是在由 RaiseError 操作符所调用的 error 过程控制下进行的。

24.6.1 Error 过程和词典

所有默认的 error 过程将在操作栈中放入相应解释器错的名字，然后调用来自 ErrorDict 的 StoreErrorInfo 过程，最后执行 RaiseException 操作符。

ErrorDict 的默认内容包括 StoreErrorInfo 过程、ReportErrorInfo 过程和所有 error-specific 过程（见 24.6.2 所有 error 的一览表）。ErrorDict 本身作为一个 SystemDict 项而被保存。

ErrorInfoDict 的默认内容和它们的语义在 24.7 中叙述，ErrorInfoDict 本身作为一个 SystemDict 而被保存。

24.6.2 可能的内容错误

下面是可能出现的内容错误和它们语义的一览表。每一错误出现所引起的异常的名字与下表中的错误名相同。

纵观这一标准，操作符语义的描述包括了对在执行这些操作符过程中可能产生异常的条件的描述。此外，还有几乎在执行任何操作符的过程中均可能引起异常的普通错误，这些普通错误是 InvalidAccess，LimitCheck，RangeCheck，StackOverflow，StackUnderflow，TypeCheck 和 noMemory 等。

下面是由本标准所定义的可能出现的内容错误一览表：

——ContentWarning——在一个 DPI/AbortPolicy 值为 onWarning 的 Block 中执行 RaiseWarning 操作符。

——ContextStackOverflow——试图在一个已满的上下文栈中执行 PushContexStack。

——ContentStackUnderflow——试图在一个已没有可移动词典的上下文栈中执行 PopContextStack。

——DictionaryFull——试图在一个已满的词典中定义一个新的项。

——InvalidAccess——试图去违反一个对象的访问属性。

——InvalidExit——在任何循环结构外执行 Exit 操作。

——InvalidFont——FindFont 有一个无效操作数。

——InvalidID——FindResource 有一个无效的 INTERNAL IDENTIFIER 操作数。

——InvalidRestore——当上下文栈或操作数栈中包括一个对象的 ObjectReference，且这个对象是在 RestoreState 的 SaveObject 操作数已经建立以后所创建的，则再试图去执行 RestoreState 操作符。

——IOError——一个与具体实现有关的错误已经出现，而再去读出或写入一个流对象 StreamObject。

——LimitCheck——一种实现限制（对于这种限制没有定义专门的异常过程）已经超出。

——NoCurrentPosition——若当前路径为空路径，则试图去执行一个路径构造操作，而该操作又不是由下面一些操作所实现：BeginPathSegment，SetPosition，ArcToClockwise 或 ArcToCounterClockwise。

——NoMemory——没有足够的存储空间以便可以成功地执行操作。

——RangeCheck——一个操作数的值不在执行操作符所允许的范围以内。

——StackOverflow——有关操作数栈深度的实现限制已被超出。

——StackUnderflow——栈中无足够的操作数以致操作符无法执行。

——SyntaxError——所遇到的文档内容并不符合内容交换格式的要求。

——Timeout——用来执行一个文档的默认实现时间限制或文档中由 DPI 所指定的时间限制已被超过。

——TypeCheck——一个操作数具有错误对象类型。

——UndefineKey——具有一个指定 DictionaryKey 的有序对，而在上下文栈中找不到它的名字元素，或一个操作数，其值是名字列表中一个所希望的值，但该值超出了范围。

——UndefineResource——没有指定类型的资源与所提供的名字操作数相对应。

——UndefinedResult——上溢/下溢或无意义结果。例如除数为 0 去执行 Divid 操作符，或一个负数开平方根（SquareRoot）。

——UnmatchMark——希望的 Mark 对象并不在操作数栈中。

24.7 Error InfoDict(错误信息词典)

下面一些条叙述缺省 Error InfoDict 的项和它们的语义。必选的项在所有实现过程中均存在;可选项在某些实现过程中并不存在,这时未给出项的默认值由异常处理函数设定。

24.7.1 newerror(新错误)

必选项〈Newerror:Boolean〉用于解释器去跟踪错误处理。项的默认值由 StoreErrorInfo-过程设置为 true,且由 ReportErrorInfo 过程设置成 false。改变这些过程值的文档必须保留包含 newerror 的原始语义(见 24.5.6 和 24.5.7)。

24.7.2 错误名(errorname)

必选项〈errorname:Name〉是产生的错误名。错误名的全部列表在 24.6.2 中给出。

24.7.3 command

必选项〈command:Operator〉是错误发生时解释器执行的操作符。

24.7.4 ostack

必选项〈ostack:VectorReference〉是对包括在错误发生前全部操作数栈转储的向量的引用,就象由 MakeandStoreVector 操作符所存储的一样。

24.7.5. dstack

必选项〈dstack:VectorReference〉是对包括在错误发生前全部上下文栈转储的向量的引用,就象由 ContexStack 操作符所存储的一样。

24.7.6 recordstacks

可选项〈recordstack:Boolean〉控制缺省的 StoreErrorInfo 过程(见 24.5.6)是否记录 ostackd 和 dstack 的转储。值为真表示栈的内容被记录。若 ErrorInfoDict 中此项没有给出,则缺省的 StoreErrorInfo 过程当作此项已给出,且值为真一样执行。

25 交换格式 Interchange Format

通过多种过程可以很容易来分析 SPDL 结构。它们包括强制过程、从 SPDL 文件中抽取页面或图形的过程以及表示过程等。为了支持在多种环境下通过多种过程去分析 SPDL 结构,SPDL 结构的 ANS.1 和 SGML 编码都已标准化。这两种编码完全等价:一种编码表示的函数可以通过一个简单句法转换变为另一种编码表示。仅一种表示在其实现时需要遵守本标准。

SPDL 文件结构允许过程在不需要中断描述页面和图形图像的标记序列的情况下,能从 SPDL 文件中抽取元素和相关的属性,并决定外部资源要求。除表示过程以外,无其他过程需要去分析标记序列。本标准指定了构成内容的两种编码标记序列的格式是:压缩二进制编码和可读纯文本编码。为了解释的简洁和高效,使用了易于解释和简洁的优化表示方法。

SPDL 内容的二进制编码要连同 SPDL 结构的二进制编码一起使用。SPDL 内容的纯文本编码要连同 SPDL 结构的纯文本编码一起使用。

25.1 二进制结构交换格式

二进制结构交换格式基于 ASN.1。本章将定义其编码。

25.1.1 SPDL 模式

```
SPDL{1 10180 10}      DEFINITIONS  ::=BEGIN
EXPORTS Pageset,Picture,Resource-Def,Name,
Reference-Name
IMPORTS          Font-Reference FROM ISO9541-2-FONTS
                 {1 9541 2 1 0}
                 Structured-Name FROM ISO9541-2-FONTS
```

```
                {1 9541 2 1 0}
——参考 GB/T 16262 中的简单类型,定义以下类型
Name            ::=[APPLICATION O]PrintableString
                (FROM(A..Z|a..Z|"_"|"."
                |":"|0..9))
Reference-Name          ::=CHOICE{
                        [APPLICATION O]Name,
                        [UNIVERSAL 6]OBJECT IDENTIFIER
                            }
——高层结构元素
SPDL—Instance           ::=CHOICE{
document                    Document,
resource-def                Resource-Def
                                }
Document            ::=CHOICE{
  pageset               Pageset,
  picture               Picture,
                            }
——SPDL 结构语法区别的高层结构元素是全部应用类型
Pageset             ::={APPLICATION 1}IMPLICIT
                    SEQUENCE{
spdl-id             [UNIVERSAL 6]OBJECT IDENTIFIER
prologue            [0] IMPLICIT Prologue OPTIONAL
body                [1] IMPLICIT SEQUENCE OF CHOICE
                        {
                        [APPLICATION 1]Pageset
                        [APPLICATION 2]Picture
                            }OPTIONAL
                        }
Picture             ::=[APPLICATION 2]IMPLICIT
                    SEQUENCE{
spdl-id             [UNIVERSAL 6]OBJECT IDENTIFIER,
content-type-id     [0] IMPLICIT OBJECT IDENTIFIER,
prologue            [1] IMPLICIT Prologue OPTIONAL,
body                [2] IMPLICIT SEQUENCE OF CHOICE
                        {
                        [APPLICATION 2]Picture
                        [APPLICATION 3]
                        TokenSequence
                            }OPTIONAL
                        }
  ——两个附加应用类型
TokenSequence           ::=[APPLICATION 3]IMPLICIT OCTET
```

```
                                STRING
Comment                         ::=[APPLICATION 4]IMPLICIT PrintableString
——Prologue 结构元素
    Prologue                    ::=SEQUENCE{
external-dec                        [0] IMPLICIT SEQUENCE OF
                                        External-Dec OPTIONAL,
informative-dec                     [1] Informative-Dec OPTIONAL,
resource-def                        [2] IMPLICIT SEQUENCE OF
                                        Resource-Def OPTIONAL,
resource-dec                        [3] IMPLICIT SEQUENCE OF
                                        Resource-Dec OPTIONAL,
dpi-dec                             [4] IMPLICIT SEQUENCE OF
                                        DPI-Declaration OPTIONAL,
context-dec                         [5] Context-Dec OPTIONAL,
dictionary-gen                      [6] IMPLICIT SEQUENCE OF
                                        Dictionary-Gen OPTIONAL,
setup-proc                          [7] IMPLICIT SEQUENCE OF
                                        Setup-Proc OPTIONAL
                                    }
Informative-Dec                 ::=SET OF Hint
Hint                            ::=SEQUENCE{
  hint-name                         [0] Reference-Name,
  hint-value                        [1] ANY DEFINED BY hint-name
                                    }
Context-Dec                     ::=SEQUENCE OF Name
Dictionary-Gen                  ::=SEQUENCE{
  dictionary-id                     [0] IMPLICIT Name,
  size                              [1] IMPLICIT INTEGER,
  dictionary-generator              [2] IMPLICIT SEQUENCE OF
                                        TokenSequence OPTIONAL
                                    }
Setup-Proc                      ::=SEQUENCE OF TokenSequence
——外部参考
External-ID                     ::=[APPLICATION 5]CHOICE{
                                    [APPLICATION 0]Name,
                                    [UNIVERSAL 6]OBJECT IDENTIFIER
                                    }
  External-Dec                  ::=SEQUENCE{
  external-id                       [APPLICATION 5]External-ID,
  structure-type-id                 [0] IMPLICIT Structure-Type-ID,
  structure-id                      [1] Structure-id
                                    }
```

```
Structure-Type-ID          ::=ENUMERATED{
                                Pageset(1),
                                Picture(2),
                                Resource Definition(3),
                                Prologue(4),
                                Resource Declaration(5),
                                DPI Declaration(6),
                                Context Declaration(7),
                                Dictionary Generator(8),
                                Setup Procedure(9),
                                Tokensequence(10),
                                Informative Declaration(11),
                                Structure Identifier(12)
                                Resource Reference(13),
                                Resource Specification(14),
                                Font Object Reference(15),
                                Font Reference(16),
                                Property List(17),
                                Property(18),
                                Run Vector(19)
                                }
Structure-id               ::=CHOICE{
    object-id                   [UNIVERSAL 6]OBJECT IDENTIFIER,
    octet-string                [UNIVERSAL 4]OCTET STRING
                                }
```

——资源定义和说明

```
Resource-Dec               ::=SEQUENCE{
  internal-id                   [0] IMPLICIT Name,
  resource-type-id              [1] IMPLICIT Resource-Type-ID,
  resource-id                   [2] Reference-Name
                                }
Resource-Type-ID           ::=ENUMERATED{
                                FontObject(1),
                                Glyph Index Map(2),
                                Font Index Map(3),
                                Color Space(4),
                                Data Source(5),
                                Filter(6),
                                Pattern(7),
                                Form(8),
                                }
Resource-Def               ::=[APPLICATION 6]IMPLICIT
```

```
                                SEQUENCE{
  spdl-id                       [UNIVERSAL 6]OBJECT IDENTIFIER,
  resource-id                   [0] Reference-Name,
  resource-type-id              [1] IMPLICIT Resource-Type-ID,
  function-id                   [2] IMPLICIT Function-ID,
  resource-spec                     Resource-Spec OPTIONAL
                                }
Function-ID                     ::=ENUMERATED{
                                      Define(1),
                                      Undefine(2)
                                      }DEFAULT Define(1)
Resource-Spec                   ::=CHOICE{
  font-object                   [3] Font-Object-Spec,
  glyph-index-map               [4] Glyph-Map-Spec,
  font-index-map                [5] Font-Map-Spec,
  color-space                   [6] Color-Space-Spec,
  pattern                       [7] Pattern-Spec,
  form                          [8] Form-Spec
                                }
Font-Object-Spec                ::=CHOICE{
  font-reference                [0] IMPLICIT Font-Ref-Spec,
  font-object                   [1] IMPLICIT Base-Font—Spec,
  named-font                    [2] IMPLICIT Named-Font-Spec,
  composite-font                [3] IMPLICIT Composite-Font-
                                    Spec
  constructed-font              [4] IMPLICIT Constructed-Font-
                                    Spec,
                                }
Font-Ref-Spec                   ::=SEQUENCE{
  font-reference                [0] Font-Reference,
  glyph-index-map-id            [1] Reference-Name
                                }
Base-Font-Spec                  ::=OCTET STRING
Named-Font-Spec                 ::=SEQUENCE{
  font-obj-id                   [0] Reference-Name,
  glyph-index-map-id            [1] Reference-Name OPTIONAL
                                }
Composite-Font-Spec             ::=SEQUENCE{
  fmap-type                     [0] IMPLICIT INTEGER,
  fmap-param-list               [1] IMPLICIT FMap-Param
                                    OPTIONAL,
  font-index-map-id             [2] Reference-Name,
```

```
    font-object-list            [3] IMPLICIT SEQUENCE OF Font-
                                    Object-Spec
                                }
FMap-Param                  ::=SEQUENCE{
    escchar                     [0] IMPLICIT INTEGER DEFAULT
                                    255,
    shiftout                    [1] IMPLICIT INTEGER DEFAULT
                                    14,
    shiftin                     [2] IMPLICIT INTEGER DEFAULT
                                    15,
    font-object-list            [3] IMPLICIT INTEGER OPTIONAL
                                }
Constructed-Font-Spec       ::=SEQUENCE OF TokenSequence
Glyph-Map-Spec              ::=SEQUENCE{
    map-size                    [0] IMPLICIT INTEGER,
    glyph-id-list               [1] IMPLICIT SEQUENCE OF Glyph-
                                    ID OPTIONAL,
                                }
Glyph-ID                    ::=SEQUENCE{
    glyph-id                        Glyph-Identifier DEFAULT
                                    not-defined,
                                }
Glyph-Identifier            ::=CHOICE{
    name                        [0] IMPLICIT Name,
    structured-name             [1] IMPLICIT Structured-Name,
    iso-10036-id                [2] IMPLICIT Name
                                }
not-defined Name            ::=".notdef"
Font-Map-Spec               ::=SEQUENCE{
    map-size                    [0] IMPLICIT INTEGER,
    index-list                  [1] IMPLICIT SEQUENCE OF
                                    INTEGER OPTIONAL,
                                }
Color-Space-Spec            ::=SEQUENCE{
    color-space-name            [0] IMPLICIT OBJECT IDENTIFIER,
    primary-color-id            [1] IMPLICIT OBJECT IDENTIFIER
                                    OPTIONAL,
    coor-space-spec             [2] IMPLICIT TokenSequence
                                }
Pattern-Spec                ::=TokenSequence
Form-Spec                   ::=TokenSequence
DPI-Declaration             ::=SEQUENCE{
    dpi-name                    [0] IMPLICIT OBJECT IDENTIFIER,
```

```
  dpi-value                    [1] ANY DEFINED BY dpi-name
                                   }

END
```

25.1.2 指定 DPI 说明

```
DPI Declarations{1 10180 101}DEFINITIONS
                               ::=BEGIN
IMPORTS                        Name
                                   FROM SPDL{1 10180 10}
                               Reference-Name
                                   FROM SPDL{1 10180 10}
Number                         ::=CHOICE{
                                   [UNIVERSAL 2] INTEGER,
                                   [UNIVERSAL 9] REAL
                                       }
Property-List                  ::=SET OF SEQUENCE{
  property-name                    [0] IMPLICIT OBJECT IDENTIFIER,
  property-value                   [1] ANY DEFINED BY property-
                                       name
                                       }
Run-Vector                     ::=SEQUENCE OF SEQUENCE{
  run-length                       [0] IMPLICIT INTEGER,
  run-value                        [1] CHOICE{
                                       [APPLICATION 0]Name,
                                       [UNIVERSAL 2] INTEGER
                                           }
                                           }
```

——指定文本产生指令

——以下文本产生指令的结构与 DPI 说明中的一样。它们定义了与 DPI 名字的特别值相对应的值的类型。

——Medium Document Production Instruction

```
Medium-DPI                     ::=SEQUENCE{
  dpi-name                         [0] IMPLICIT OBJECT IDENTIFIER
                                       (dpi-medium),
  dpi-value                        [1] Medium-Spec
                                       }
dpi-medium                     OBJECT IDENTIFIER
                                   ::={1 0 10180 5 0}
```

——对应于 Object Name DPI/Medium

```
Medium-Spec                    ::=SEQUENCE{
  medium-id                        [0]IMPLICIT Name,
  medium-desc                      [1] IMPLICIT Medium-Description
                                       }
Medium-Description             ::=Property-List
```

```
——specifying medium properties
——Medium Properties
Medium-Name                    ::=SEQUENCE{
  prop-name                        [0] IMPLICIT OBJECT IDENTIFIER
                                       (dpi-medium-name),
  prop-value                       [1] Reference-Name
                                       }
  dpi-medium-name                      OBJECT IDENTIFIER
                                           ::={1 0 10180 5 0 1}
——对应于                       Object Name DPI/Medium/Name
  dpi-medium-name-default OBJECT IDENTIFIER
                                           ::={10 10180 5010}
——对应于                       Object Name DPI/Medium/Name/default
Medium-Size                    ::=SEQUENCE{
  Prop-name                        [0] IMPLICIT OBJECT IDENTIFIER
                                       (dpi-medium-size)
prop-value                         [1] SEQUENCE{
  x-size                               Number,
  y-size                               Number
                                           }
                                       }
  dpi-medium-size              OBJECT IDENTIFIER
                                   ::={1 0 10180 5 0 3}
——对应于                       Object Name DPI/Medium/Size
Medium-Color                   ::=SEQUENCE{
  prop-name                        [0] IMPLICIT OBJECT IDENTIFIER
                                       (dpi-medium-color)
  prop-value                       [1] OBJECT IDENTIFIER DEFAULT
                                       dpi-medium-color-white
                                       }
dpi-medium-color               OBJECT IDENTIFIER
                                           ::={1 0 10180 5 0 4}
——对应于                       Object Name DPI/Medium/Color
  dpi-medium-color-white OBJECT IDENTIFIER
                                           ::={1 0 10180 5 0 4 0}
——对应于                       Object Name DPI/Medium/Color/white
  dpi-medium-color-pink OBJECT IDENTIFIER
                                           ::={1 0 10180 5 0 4 1}
——对应于                       Object Name DPI/Medium/Color/pink
  dpi-medium-color-yellow OBJECT IDENTIFIER
                                           ::={1 0 10180 5 0 4 2}
——对应于                       Object Name DPI/Medium/Color/yellow
  dpi-medium-color-buff OBJECT IDENTIFIER
```

::={1 0 10180 5 0 4 3}

——对应于 Object Name DPI/Medium/Color/buff

dpi-medium-color-goldenrod OBJECT IDENTIFIER

::={1 0 10180 5 0 4 4}

——对应于 Object Name DPI/Medium/Color/goldenrod

dpi-medium-color-blue OBJECT IDENTIFIER

::={1 0 10180 5 0 4 5}

——对应于 Object Name DPI/Medium/Color/blue

dpi-medium-color-green OBJECT IDENTIFIER

::={1 0 10180 5 0 4 6}

——对应于 Object Name DPI/Medium/Color/green

dpi-medium-color-nocolor OBJECT IDENTIFIER

::={1 0 10180 5 0 4 7}

——对应于 Object Name DPI/Medium/Color/nocolor

Medium-Weight ::=SEQUENCE{

prop-name [0] IMPLICIT OBJECT IDENTIFIER

(dpi-medium-weight)

prop-value [1] Number

}

dpi-medium-weight OBJECT IDENTIFIER

::={1 0 10180 5 0 5}

——对应于 DPI/Medium/Weight

Medium-Opacity ::=SEQUENCE{

prop-name [0] IMPLICIT OBJECT IDENTIFIER

(dpi-medium-opacity)

prop-value [1] ENUMERATED{

Transparent(0)

Opaque(1)

}

dpi-medium-opacity OBJECT IDENTIFIER

::={1 0 10180 5 0 6}

——对应于 DPI/Medium/Opacity

Medium-Pre-Finish ::=SEQUENCE{

Prop-name [0]IMPLICIT OBJECT IDENTIFIER

(dpi-medium-prefinish)

prop-value [1] Reference Name

}

dpi-medium-prefinish OBJECT IDENTIFIER

::={1 0 10180 5 0 7}

——对应于 DPI/Medium/Prefinish

dpi-medium-prefinish-plain-OBJECT IDENTIFIER

::={10 10180 5 0 7 0}

——对应于 DPI/Medium/PreFinish/plain

```
  dpi-medium-prefinish-precuttab OBJECT IDENTIFIER
                                        ::={1 0 10180 5 0 7 1}
——对应于                        OPI/Medium/Prefinish/pre CutTab
  dpi-medium-prefinish-continuous OBJECT IDENTIFIER
                                        ::={1 0 10180 5 0 7 2}
——对应于                        DPI/Medium/Prefinish/continuous
  dpi-medium-prefinish-adhesivelabels OBJECT IDENTIFIER
                                        ::={1 0 10180 5 0 7 3}
——对应于                        DPI/Medium/Prefinish/adhesiveLabels
  dpi-medium-prefinish-envelopeplain OBJECT IDENTIFIER
                                              ::={1 0 10180 5 0 7 4}
——对应于                        DPI/Medium/Prefinish/envelopePlain
  dpi-medium-prefinish-envelopewindow OBJECT IDENTIFIER
                                              ::={1 0 10180 5 0 7 5}
——对应于                        DPI/Medium/Prefinish/envelopeWindow
Medium-hole-count               ::=SEQUENCE{
  prop-name                         [0] IMPLICIT OBJECT IDENTIFIER
                                        (dpi-medium-hole-count)
  prop-value                        [1] INTEGER
                                        }
  dpi-medium-hole-count OBJECT IDENTIFIER
                                        ::={1 0 10180 5 0 8}
——对应于                        DPI/Medium/Hole Count
Medium-ordered-count            ::=SEQUENCE{
  prop-name                         [0] IMPLICIT OBJECT IDENTIFIER
                                        (dpi-medium-ordered count)
  prop-value                        [1] SEQUENCE{
  sequence-number                           INTEGER,
  sequence-length                           INTEGER
                                            }
                                        }
  dpi-medium-ordered-count OBJECT IDENTIFIER
                                        ::={1 0 10180 5 0 9}
——对应于                        DPI/Medium/OrderedCount
Medium-finish-edge              ::=SEQUENCE{
  prop-name                         [0] IMPLICIT OBJECT IDENTIFIER
                                        (dpi-medium-finish-edge)
  prop-value                        [1] ENUMERATED{
                                            bottom(0),
                                            right(1),
                                            top(2),
                                            left(3)
                                            }
```

```
                                    }
  dpi-medium-finish-edge OBJECT IDENTIFIER
                                    ::={1 0 10180 5 0 10}
——对应于                            DPI/Medium/FinishEdge
Medium-labels                       ::=SEQUENCE{
  Prop-name                             [0] IMPLICIT OBJECT IDENTIFIER
                                        (dpi-medium-labels)
  prop-value                            [1] SEQUENCE{
  per-column                                INTEGER,
  per-row                                   INTEGER
                                    }
  dpi-medium-labels OBJECT IDENTIFIER
                                    ::={10 10180 5 0 11}
——对应于                            DPI/Medium/Labels
Medium-message                      ::=SEQUENCE{
  prop-name                             [0] IMPLICIT OBJECT IDENTIFIER
                                        (dpi-medium-message)
  prop-value                            [1] Printable String
                                    }
  dpi-medium-message OBJECT IDENTIFIER
                                    ::={1 0 10180 5 0 12}
——对应于                            DPI/Medium/Message
——Colorants Document Production Instruction
Colorants-DPI                       ::=SEQUENCE{
  dpi-name                              [0] IMPLICIT OBJECT IDENTIFIER
                                        (dpi-colorants),
  dpi-value                             [1] SEQUENCE{
    colorants-set-id                        [0] Name
    colorants-set                           [1] SET OF Colorant-id
                                        }
  dpi-colorants OBJECT IDENTIFIER
                                    ::={1 0 10180 5 17}
——对应于                            DPI/Colorants
Colorant-id                         ::=Reference-name
——Copies Document Production Instruction
Copies-DPI                          ::=SEQUENCE{
  dpi-name                              [0] IMPLICIT OBJECT IDENTIFIER
                                        (dpi-copies),
  dpi-value                             [1] INTEGER
                                    }
  dpi-copies                        OBJECT IDENTIFIER
                                    ::={1 0 10180 5 1}
——对应于                            Object Name DPI/Copies
```

```
——Page Select Document Production Instruction
Page-Select-DPI                ::=SEQUENCE{
  dpi-name                         [0] IMPLICIT OBJECT IDENTIFIER
                                       (dpi-page-select),
  dpi-value                        [1] Run-Vector(WITH COMPONENTS
                                       {
                                       run-length,
                                       run-value(INTEGER)
                                           }
                                       }
  dpi-page-select              OBJECT IDENTIFIER
                                       ::={1 0 10180 5 2}
——对应于                       Object Name DPI/PageSelect
——Medium Select Document Production Instruction
Medium-Select-DPI              ::=SEQUENCE{
  dpi-name                         [0] IMPLICIT OBJECT IDENTIFIER
                                       (dpi-medium-select),
  dpi-value                        [1] Run-Vector(WITH COMPONENTS
                                       {
                                       run-length,
                                       run-value(Name)
                                           }
                                       }
  dpi-medium-selet OBJECT IDENTIFIER
                                       ::={10 10180 53}
——对应于                       Object Name DPI/MediaSelect
——Current Medium Document Production Instruction
    Current-Medium-DPI         ::=SEQUENCE{
  dpi-name                         [0] IMPLICIT OBJECT IDENTIFIER
                                       (dpi-current-medium),
  dpi-value                        [1] Name
                                       }
dpi-current-medium OBJECT IDENTIFIER
                                       ::={1 0 10180 5 18}
——对应于                       Object Name DPI/CurrentMedium
——Colorants Select Document Production Instruction
    Colorants-Select-DPI       ::=SEQUENCE{
  dpi-name                         [0] IMPLICIT OBJECT IDENTIFIER
                                       (dpi-colorants-select),
  dpi-value                        [1] Run-Vector(WITH COMPONENTS
                                       {
                                       run-length,
                                       run-value(Name)
```

```
                                    }
                                }
  dpi-colorants-select OBJECT IDENTIFIER
                                    ::={1 0 10180 5 19}
——对应于                    Object Name DPI/ColorantsSelect
——Current Colorants Document Production Instruction
Current-Colorants-DPI       ::=SEQUENCE{
  dpi-name                      [0] IMPLICIT OBJECT IDENTIFIER
                                    (current-colorants),
  dpi-value                     [1] Run-Vector
                                    }
  current-colorants         OBJECT IDENTIFIER
                                    ::={1 0 10180 5 20}
——对应于                    Object Name DPI/Current Colorants
——Plex Document Production Instruction
Plex-DPI                    ::=SEQUENCE{
  dpi-name                      [0]IMPLICIT OBJECT IDENTIFIER
                                    (dpi-plex),
  dpi-value                     [1] OBJECT IDENTIFIER DEFAULT
                                    dpi-simplex
                                    }
  dpi-plex                  OBJECT IDENTIFIER
                                    ::={1 0 10180 5 4}
——对应于                    Object Name DPI/Plex
  dpi-simplex               OBJECT IDENTIFIER
                                    ::={1 0 10180 5 4 0}
——对应于                    Object Name DPI/simplex
  dpi-duplex                OBJECT IDENTIFIER
                                    ::={1 0 10180 5 4 1}
——对应于                    Object Name DPI/duplex
  dpi-tumble-duplex           OBJECT IDENTIFIER
                                    ::={1 0 10180 5 4 2}
——对应于                    Object Name DPI/tumbleDuplex
——x-Image-Shift Document Production Instruction
X-Image-Shift-DPI           ::=SEQUENCE{
  dpi-name                      [0] IMPLICIT OBJECT IDENTIFIER
                                    (dpi-x-image-shift),
  dpi-value                     [1] Number
                                    }
  dpi-x-image-shift         OBJECT IDENTIFIER
                                  ::={1 0 10180 5 5}
——对应于                    Object Name DPI/XImageShift
——Y-Image-Shift Document Production Instruction
```

```
    Y-Image-Shift-DPI          ::=SEQUENCE{
      dpi-name                     [0] IMPLICIT OBJECT IDENTIFIER
                                       (dpi-y-image-shift),
      dpi-value                    [1] Number
                                       }
dpi-y-image-shift              OBJECT IDENTIFIER
                                       ::={1 0 10180 5 6}
——对应于                       Object Name DPI/YlmageShift
——Only On Duplex Document Production Instruction
Only-On-Duplex-DPI             ::=SEQUENCE{
  dpi-name                         [0] IMPLICIT OBJECT IDENTIFIER
                                       (dpi-only-on-duplex),
  dpi-value                        [1] BOOLEAN
                                       }
dpi-only-on-duplex  OBJECT IDENTIFIER
                                       ::={1 0 10180 5 7}
——对应于                       Obiect Name DPI/OnlyOnDuplex
——Break Page Type Document Production Instruction
    Break-Page-Type-DPI        ::=SEQUENCE{
  dpi-name                         [0]IMPLICIT OBJECT IDENTIFIER
                                       (dpi-break-page-type),
  dpi-value                        [1] OBJECT IDENTIFIER
                                       }
  dpi-break-page-type OBJECT IDENTIFIER
                                       ::={1 0 10180 5 8}
——对应于                       Object Name DPI/Breakpage Type
  dpi-break-page-type-none  OBJECT IDENTIFIER
                                       ::={1 0 10180 5 8 0}
——对应于                       Object Name DPI/BreakPageType/none
  dpi-break-page-type-terse OBJECT IDENTIFIER
                                       ::={1 0 10180 5 8 1}
——对应于                       Object Name DPI/BreakPageType/terse
  dpi-break-page-type-verbose  OBJECT IDENTIFIER
                                       ::={1 0 10180 5 8 2}
——对应于                       Object Name DPI/BreakPageType/verbose
——Output Position Document Production Instruction
Output-Position-DPI            ::=SEQUENCE{
  dpi-name                         [0] IMPLICIT OBJECT IDENTIFIER
                                       (dpi-output-position),
  dpi-value                        [1] Run-Vector(WITH COMPONENTS
                                       {
                                       run-length,
                                       run-value(INTEGER)
```

```
                                        }
                                    }
dpi-output-position OBJECT IDENTIFIER
                                    ::={1 0 10180 5 9}
——对应于                        Object Name DPI/OutputPosition
——Stacking Document Production Instruction
Stacking-DPI                    ::=SEQUENCE{
  dpi-name                          [0] IMPLICIT OBJECT IDENTIFIER
                                        (dpi-stacking),
  dpi-value                         [1] Property-List
                                        }
  dpi-stacking                  OBJECT IDENTIFIER
                                    ::={1 0 10180 5 1 0}
——对应于                        Object Name DPI/Stacking
——Stacking Properties
Stacking-Collated               ::=SEQUENCE{
  prop-name                         [0] IMPLICIT OBJECT IDENTIFIER
                                        (dpi-stacking-collated),
  prop-value                        [1] BOOLEAN
                                        }
  dpi-stacking-collated  OBJECT IDENTIFIER
                                    ::={1 0 10180 5 10 0}
——对应于                        Object Name DPI/Stacking/Collated
Stacking-Offset                 ::=SEQUENCE{
  prop-name                         [0] IMPLICIT OBJECT IDENTIFIER
                                        (dpi-stacking-offset),
  prop-value                        [1] BOOLEAN
                                        }
dpi-stacking-offset OBJECT IDENTIFIER
                                    ::={10 1 0180 5 10 1}
——对应于                        Object Name DPI/Stacking/Offset
Stacking-Slipsheet              ::=SEQUENCE{
  prop-name                         [0] IMPLICIT OBJECT IDENTIFIER
                                        (dpi-stacking-slipsheet),
  prop-value                        [1] BOOLEAN
                                        }
dpi-stacking-slipsheet OBJECT IDENTIFIER
                                    ::={1 0 10180 5 10 2}
——对应于                        Object Name DPI/Stacking/Slipsheet
——Finishing Document Production Instruction
Finishing-DPI                   ::=SEQUENCE{
  dpi-name                          [0] IMPLICIT OBJECT IDENTIFIER
                                        (dpi-finishing),
```

```
  dpi-value                    [1] SEQUENCE OF Finishing-
                                   Specification
                               }
dpi-finishing                  OBJECT IDENTIFIER
                                   ::={1 0 10180 5 11}
——对应于                       Object Name DPI/Finishing
Finishing-Specification        ::=Property-List
——Finishing Properties
Finishing-Reference-Size           ::=SEQUENCE{
  Prop-name                    [0] IMPLICIT OBJECT IDENTIFIER
                                   (dpi-finishing-reference-
                                   size)
  prop-Value                   [1] SEQUENCE{
  x-size                           Number,
  y-size                           Number
                                        }
                                   }
dpi-finishing-reference-size   OBJECT IDENTIFIER
                                   ::={1 0 10180 5 11 0}
——对应于                       Object Name DPI/Finishing/ReferenceSize
Finishing-Reference-Edge       ::=SEQUENCE{
  prop-name                        [0] IMPLICIT OBJECT IDENTIFIER
                                   (dpi-finishing-reference-
                                   edge)
  prop-value                   [1] ENUMERATED{
                                        bottom(0),
                                        right(1),
                                        top(2),
                                        left(3)
                                        }
                                   }
dpi-finishing-reference-edge   OBJECT IDENTIFIER
                                   ::={1 0 10180 5 11 1}
——对应于                       Object Name DPI/Finishing/ReferenceEdge
Finishing-locations            ::=SEQUENCE{
  prop-name                        [0] IMPLICIT OBJECT IDENTIFIER
                                   (dpi-finishing-locations)
  prop-value                   [1] Property-List
                                   }
dpi-finishing-locations OBJECT IDENTIFIER
                                   ::={1 0 10180 5 11 2}
——对应于                       Object Name DPI/Finishing/Locations
Finishing-Locations-Head-Count ::=SEQUENCE{
```

```
  prop-name                          [0] IMPLICIT OBJECT IDENTIFIER
                                         (dpi-finishing-locations-
                                         head-count)
  prop-value                         [1] INTEGER
                                         }
dpi-finishing-locations-head-count OBJECT IDENTIFIER
                                         ::={1 0 10180 5 11 20}
——对应于                          Object Name
——DPI/Finishing/Locations/HeadCount
Finishing-Locations-Process-Offset  ::=SEQUENCE{
  prop-name                          [0] IMPLICIT OBJECT IDENTIFIER
                                         (dpi-finishing-locations-
                                         process-offset)
  prop-value                         [1] Number
                                         }
dpi-finishing-locations-process-offset OBJECT IDENTIFIER
                                         ::={1 0 10180 5 11 2 1}
——对应于                          Object Name
——DPI/Finishing/Locations/ProcessOffset
Finishing-Locations-Head-Locations  ::=SEQUENCE{
  Prop-name                          [0] IMPLICIT OBJECT IDENTIFIER
                                         (dpi-finishing-locations-
                                         head-locations)
    prop-value                       [1] SEQUENCE OF Number
                                         }
dpi-finishing-locations-head-locations OBJECT IDENTIFIER
                                         ::={1 0 10180 5 11 2 2}
——对应于                          Object Name
——DPI/Finishing/Locations/HeadLocations
Finishing-Inserts                  ::=SEQUENCE{
  prop-name                          [0] IMPLICIT OBJECT IDENTIFIER
                                         (dpi-finishing-inserts)
  prop-value                         [1] SEQUENCE OF Property-List
                                         }
dpi-finishing-inserts  OBJECT IDENTIFIER
                                         ::={1 0 10180 5 11 3}
——对应于                          Object Name DPI/Finishing/Inserts
Finishing-Inserts-Insert-Name  ::=SEQUENCE{
  prop-name                          [0] IMPLICIT OBJECT IDENTIFIER
                                         (dpi-finishing-inserts-
                                         insert-name)
  prop-value                         [1] Reference-Name
                                         }
```

```
dpi-finishing-inserts-insert-name   OBJECT IDENTIFIER
                                        ::={1 0 10180 5 11 3 0}
——对应于                            Object Name
DPI/Finishing/Inserts/InsertName
Finishing-Inserts-Insert-Size   ::=SEQUENCE{
  Prop-name                         [0] IMPLICIT OBJECT IDENTIFIER
                                        (dpi-finishing-inserts-
                                        insert-size)
  prop-value                        [1] SEQUENCE{
  x-size                                Number,
  y-size                                Number
                                            }
                                        }
dpi-finishing-inserts-insert-size   OBJECT IDENTIFIER
                                        ::={1 0 10180 5 11 3 1}
——对应于                            Object Name
——DPI/Finishing/Inserts/InsertSize
Finishing-Inserts-Insert-Edge   ::=SEQUENCE{
  prop-name                         [0] IMPLICIT OBJECT IDENTIFIER
                                        (dpi-finishing-inserts-
                                        insert-edge)
  prop-value                        [1] ENUMERATED{
                                            bottom(0),
                                            right(1),
                                            top(2),
                                            left(3)
                                            }
                                        }
dpi-finishing-inserts-insert-edge   OBJECT IDENTIFIER
                                        ::={1 0 10180 5 11 3 2}
——对应于                            Object Name
——DPI/Finishing/Inserts/InsertEdge
Finishing-Inserts-Insert-Top-Surface   ::=SEQUENCE{
  prop-name                         [0] IMPLICIT OBJECT IDENTIFIER
                                        (dpi-finishing-inserts-
                                        insert-top-surface)
  prop-value                        [1] ENUMERATED{
                                            top(0),
                                            bottom(1)
                                            }
                                        }
dpi-finishing-inserts-insert-top-surface
                        OBJECT IDENTIFIER
```

```
                    ::={10 10180 5 11 33}
——对应于                Object
——DPI/Finishing/Inserts/InsertTopSurface
Finishing-Inserts-Insert-Bin  ::=SEQUENCE{
  prop-name                 [0] IMPLICIT OBJECT IDENTIFIER
                                (dpi-finishing-inserts-
                                insert-bin)
  prop-value                [1] INTEGER
                                }
dpi-finishing-inserts-insert-bin OBJECT IDENTIFIER
                                ::={1 0 10180 5 11 3 4}
——对应于              Object DPI/Finishing/Inserts/InsertBin
Finishing-Inserts-Insert-After  ::=SEQUENCE{
  prop-name                 [0] IMPLICIT OBJECT IDENTIFIER
                                (dpi-finishing-inserts-
                                insert-after)
  prop-value                [1] INTEGER
                                }
dpi-finishing-inserts-insert-after  OBJECT IDENTIFIER
                                ::={1 0 10180 5 11 3 5}
——对应于              Object DPI/Finishing/Inserts/InsertAfter
Finishing-Trimmed-Size        ::=SEQUENCE{
  prop-name                 [0] IMPLICIT OBJECT IDENTIFIER
                                (dpi-finishing-trimmed-
                                size)
  prop-value                [1] SEQUENCE{
  x-size                        Number,
  y-size                        Number,
  edge-offset                   Number,
                                }
                                }
dpi-finishing-trimmed-size  OBJECT IDENTIFIER
                                ::={1 0 10180 5 11 4}
——对应于              Object DPI/Finishing/TrimmedSize
Finishing-Die-Cut-Name        ::=SEQUENCE{
  prop-name                 [0] IMPLICIT OBJECT IDENTIFIER
                                (dpi-finishing-die-cut-
                                name)
  prop-value                [1] Reference-Name
                                }
dpi-finishing-die-cut-name  OBJECT IDENTIFIER
                                ::={1 0 10180 5 11 5}
——对应于              Object DPI/Finishing/DieCutName
```

```
Finishing-Die-Cut-Position  ::=SEQUENCE{
  prop-name                      [0] IMPLICIT OBJECT IDENTIFIER
                                     (dpi-finishing-die-cut-
                                     position)
  prop-value                     [1] SEQUENCE{
    x-size                             Number,
    y-size                             Number,
    x-offset                           Number,
    y-offset                           Number
                                       }
                                   }
dpi-finishing-die-cut-position  OBJECT IDENTIFIER
                                   ::={1 0 10180 5 11 6}
——对应于                        Object DPI/Finishing/DieCutPosition
Finishing-Binding-Type          ::=SEQUENCE{
  prop-name                    [0] IMPLICIT OBJECT IDENTIFIER
                                     (dpi-finishing-binding-
                                     type)
  prop-value                     [1] Reference-Name
                                   }
dpi-finishing-binding-type  OBJECT IDENTIFIER
                                   ::={1 0 10180 5 11 7}
——对应于                        Object DPI/Finishing/BindingType
dpi-finishing-binding-type-tape  OBJECT IDENTIFIER
                                   ::={1 0 10180 5 11 7 0}
——对应于                        Object DPI/Finishing/BindingType/tape
dpi-finishing-binding-type-plastic  OBJECT IDENTIFIER
                                   ::={1 0 10180 5 11 7 1}
——  对应于                      Object DPI/Finishing/BindingType/plastic
dpi-finishing-binding-type-velo  OBJECT IDENTIFIER
                                   ::={10 10180 5 11 7 2}
——对应于                        Object DPI/Finishing/BindingType/velo
dpi-finishing-binding-type-perfect  OBJECT IDENTIFIER
                                   ::={1 0 10180 5 11 7 3}
——对应于                        Object DPI/Finishing/BindingType/perfect
dpi-finishing-binding-type-spiral  OBJECT IDENTIFIER
                                   ::={1 0 10180 5 11 7 4}
——对应于                        Object DPI/Finishing/BindingType/spiral
dpi-finishing-binding-type-default OBJECT IDENTIFIER
                                   ::={1 0 10180 5 11 7 5}
——对应于                        Object DPI/Finishing/BindingType/default
Finishing-Binding-Color         ::=SEQUENCE{
  prop-name                    [0] IMPLICIT OBJECT IDENTIFIER
```

(dpi-finishing-binding-
color)
prop-value [1] Reference-Name
}

dpi-finishing-binding-color OBJECT IDENTIFIER、
::={1 0 10180 5 11 8}
——对应于 Object DPI/Finishing/BindingColor

dpi-finishing-binding-color-black OBJECT IDENTIFIER
::={1 0 10180 5 11 8 0}
——对应于 Object DPI/Finishing/BindingColor/black

dpi-finishing-binding-color-blue OBJECT IDENTIFIER
::={1 0 10180 5 11 8 1}
——对应于 Object DPI/Finishing/BindingColor/blue

dpi-finishing-binding-color-gray OBJECT IDENTIFIER
::={1 0 10180 5 11 8 2}
——对应于 Object DPI/Finishing/BindingColor/gray

dpi-finishing-binding-color-brown OBJECT IDENTIFIER
::={1 0 10180 5 11 8 3}
——对应于 Object DPI/Finishing/BindingColor/brown

dpi-finishing-binding-color-default OBJECT IDENTIFIER
::={1 0 10180 5 11 8 4}
——对应于 Object DPI/Finishing/BindingColor/default

Finishing-Operation ::=SEQUENCE{
prop-name [0] IMPLICIT OBJECT IDENTIFIER
(dpi-finishing-operation)
prop-value [1] Reference-Name
}

dpi-finishing-operation OBJECT IDENTIFIER
::={1 0 10180 5 11 9}
——对应于 Object DPI/Finishing/FinishingOperation

dpi-finishing-convenience-staple OBJECT IDENTIFIER
::={1 0 10180 5 11 9 0}
——对应于 Object DPI/Finishing/convenienceStaple

dpi-finishing-edge-stitching OBJECT IDENTIFIER
::={1 0 10180 5 11 9 1}
——对应于 Object DPI/Finishing/edgeStitching

dpi-finishing-binding OBJECT IDENTIFIER
::={10 10180 5 11 92}
——对应于 Object DPI/Finishing/binding

dpi-finishing-saddle-stitching OBJECT IDENTIFIER
::={1 0 10180 5 11 9 3}
——对应于 Object DPI/Finishing/saddleStitching

dpi-finishing-punching OBJECT IDENTIFIER

```
                                        ::={1 0 10180 5 11 9 4}
——对应于                                Object DPI/Finishing/punching
dpi-finishing-perforating     OBJECT IDENTIFIER
                                        ::={1 0 10180 5 11 9 5}
——对应于                                Object DPI/Finishing/perforating
dpi-finishing-slitting        OBJECT IDENTIFIER
                                        ::={1 0 10180 5 11 9 6}
——对应于                                Object DPI/Finishing/slitting
dpi-finishing-trimming        OBJECT IDENTIFIER
                                        ::={1 0 10180 5 11 9 7}
——对应于                                Object DPI/Finishing/trimming
dpi-finishing-folding         OBJECT IDENTIFIER
                                        ::={1 0 10180 5 11 9 8}
——对应于                                Object DPI/Finishing/folding
dpi-finishing-inserting OBJECT IDENTIFIER
                                        ::={ 1 0 10180 5 11 9 9 }
——对应于                                Object DPI/Finishing/inserting
dpi-finishing-die-cutting OBJECT IDENTIFIER
                                        ::={ 1 0 10180 5 11 9 10 }
——对应于                                Object DPI/Finishing/dieCutting
Finishing-message             ::=SEQUENCE {
    prop-name                               [0]  IMPLICIT OBJECT IDENTIFIER
                                                 (dpi-finishing-message)
    prop-value                              [1]  prinatblestring
                                                 }
dpi-finishing-message         OBJECT IDENTIFIER
                                        ::={ 1 0 10180 5 11 10 }
——对应于                                Object DPI/Finishing/Message
——Document Comment Document Production Instruction
Document-comment-DPI          ::=SEQUENCE {
    dpi-name                                [0]  IMPLICIT OBJECT IDENTIFIER
                                                 (dpi-document-comment),
    dpi-value                               [1]  printablestring
                                                 }
dpi-document-comment          OBJECT IDENTIFIER
                                        ::={ 1 0 10180 5 12 }
——对应于                                Object Name DPI/DocumentComment
——Document Start Message Document Production Instruction
Document-start-message-DPI    ::=SEQUENCE {
    dpi-name                                [0]  IMPLICIT OBJECT IDENTIFIER
                                                 (dpi-document-start-message),
    dpi-value                               [1]  printablestring
                                                 }
```

```
dpi-document-start-message      OBJECT IDENTIFIER
                                    ::={ 1 0 10180 5 13 }
——对应于                          Object Name DPI/DocumentStartMessage
——Document End Message Document Production Instruction
Document-end-message-DPI            ::=SEQUENCE {
    dpi-name                            [0]  IMPLICIT OBJECT IDENTIFIER
                                             (dpi-document-end-message),
    dpi-value                           [1]  printablestring
                                             }
dpi-document-end-message        OBJECT IDENTIFIER
                                    ::={ 1 0 10180 5 14 }
——对应于                          Object Name DPI/DocumentEndMessage
——Timeout Document Production Instruction
Timeout-DPI             ::=SEQUENCE {
    dpi-name                            [0]  IMPLICIT OBJECT IDENTIFIER
                                             (dpi-timeout),
    dpi-value                           [1]  INTEGER
                                             }
dpi-timeout     OBJECT IDENTIFIER
                                    ::={ 1 0 10180 5 15 }
——对应于                          Object Name DPI/Timeout
——Abort policy Document Production Instruction
Abort-policy-DPI            ::=SEQUENCE {
    dpi-name                            [0]  IMPLICIT OBJECT IDENTIFIER
                                             (dpi-abort-policy),
    dpi-value                           [1]  OBJECT IDENTIFIER
                                             }
dpi-abort-policy      OBJECT IDENTIFIER
                                    ::={ 1 0 10180 5 16 }
——对应于                          Object Name DPI/AbortPolicy
dpi-abort-policy-on-warning     OBJECT IDENTIFIER
                                    ::={ 1 0 10180 5 16 0 }
——对应于                          Object Name DPI/AbortPolicy/onWarning
dpi-abort-policy-on-error       OBJECT IDENTIFIER
                                    ::={ 1 0 10180 5 16 1 }
——对应于                          Object Name DPI/AbortPolicy/onError
dpi-abort-policy-struggle-on    OBJECT IDENTIFIER
                                    ::={ 1 0 10180 5 16 2 }
——对应于                          Object Name DPI/AbortPolicy/struggleOn
END
```

25.2 二进制内容交换格式

SPDL 内容由一系列标记(tokens)序列组成。每一标记代表一个由内容处理器处理的信息单位。以下说明 SPDL 内容中标记的二进制编码。

25.2.1 标记结构和注释

每一标记由一个或多个八位字节序列组成。由以下三种方式说明的八个二进制位组成一个八位字节：

——通过各位值的说明，这里第八位为高位，第一位为低位；

——作为位值串，其中第一或最左位的值是八位字节中最高位或最有效位(Bit8)的值；

——将上述位值序列作为二进制数获得的数值(0～255)。

数值是八位字节的二进制值

位	8	7	6	5	4	3	2	1
八位字节	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1

二进制值$=b_8\times2^7+b_7\times2^6+b_6\times2^5+b_5\times2^4+b_4\times2^3+b_3\times2^2+b_2\times2^1+b_1\times2^0$

标记由说明标记类型、长度或值的一个或多个位序列构成。每个位序列称作域(field)。一个域可包含一个字节或可扩展成几个八位字节。说明标记类型、长度和值的域分别称作类型域、长度域和值域。

25.2.2 标记类型

二进制内容编码使用4个基本标记类型：

——短操作码

——类型/值

——类型/长度/值

——短整型

每个基本标记类型的格式，连同区分它们的位值，在表25-1中列出。

类型/值和类型/长度/值这两个类型都可各自再分成32个子类型。这些标记的类型域由基本类型说明和子类型说明构成。

每个标记的第一个八位字节指出标记类型，它被称作类型字节。类型字节的值决定了基本标记类型，说明如下：

类型字节	标记类型
0～63	短操作码标记
短操作码	0 0 × × × × × ×
类型/值	0 1 0 s s s s s | 值 | 值
类型/长度/值	0 1 0 s s s s s | 长度 | 长度
短整型	1 × × × × × × × | 值 … 值 | × × × × × × × ×

其中：s s s s s ＝子类型

× × × × × × ＝值

64～95	类型/值标记
96～127	类型/长度/值标记
128～255	短整型标记

类型字节还包含依赖基本标记类型的附加信息。

——对短操作码标记，类型字节包含类型域和值域。

——对类型/值和类型/长度/值标记，子类型说明是类型域的一部分，且包含在类型字节中。

——对短整型标记，类型字节包含类型域以及值域的高 7 位。

标记类型隐含了短操作码、类型/值和短整型标记的长度。当长度域中包含有组成值域的字节数时，标记类型隐含了类型域的长度和类型/长度/值标记的长度域。

对不同的标记，二进制编码提供了一种替换方法。任意八位字节串或数据块类型/长度/值标记与长八位字节串或长数据块类型/长度/值标记有相同的值。任意短整型标记与类型/值整型标记有相同的值。任意短操作码标记和类型/长度/值可执行名标记(它的值是赋给短操作码值的名)相等。

25.2.3 值的表示

二进制值用来区分标记类型和子类型，并指出类型/长度/值标记的值域的长度。域表示的二进制值是二进制数的值，此二进制数的位是域中包含在第一字节中的位序列，其后又紧跟着包含在域内第二和后继字节中的位序列。

25.2.3.1 无符号的整型值域

无符号的整型值域由两个或四个字节组成。整型表示的值是由域表示的二进制值。

25.2.3.2 有符号的整型值域

有符号的整型值域由两个或四个字节组成。整型表示的值依赖于域的第一字节的高位值。

——若第一字节的高位为 0，则整型的值是由域表示的二进制值。

——若第一字节的高位为 1，域长度是 n 个字节，则整型的值是 256n 减去由域表示的二进制值。

注：这是带符号整型值的 2 的补码表示。

25.2.3.3 浮点实数的值域

浮点实数的值域由四个字节组成。IEEE 标准中浮点数部分指出了表示的实数值。

25.2.3.4 定点实数的值域

定点实数的值域由三个或五个字节组成。若值域中第一字节的二进制值为 r，且由值域中剩余的二个或四个字节构成的域的二进制值为 n，则实数表示的值为 $n \div 2^r$。

25.2.3.5 八位字节串的值域

一个八位字节串的值域由可变数目的八位字节组成。值是由组成值域的八位字节序列来表示。

25.2.3.6 数据块值域

数据块值域由可变数目的八位字节组成。值是由组成值域的八位字节序列来表示。

25.2.3.7 过程的值域

过程值域由可变数目的八位字节组成。值是由组成值域的八位字节序列表示的标记序列(以二进制编码给出)来表示。

25.2.3.8 齐次数值域的向量

齐次(homogeneous)数值域的向量由可变数目的八位字节组成。

——值域第一个字节的值表示每一个向量元素的字节数。

——值域第二个字节的值表示数的格式。

——值域的余下部分由表示数的子域的整型数组成。

值域表示的向量是由这些数形成的向量。

向量元素的表示依赖于第二个字节的值。说明如下：

第二个字节	数的表示
0	无符号整型值域表示的无符号整型数
1	有符号整型值域表示的有符号整型数
2	浮点实数值域表示的浮点实数
>8	定点实数，由定点实数值域的第二个和后继字节来表示，而该

定点实数值域的初始字节等于第二个字节的值减 8

25.2.4 短操作码标记

一个短操作码标记表示单个八位字节中 64 个可执行名中的一个。短操作码标记构造如下：

——类型八位字节的位 8 和位 7 为 0；

——类型八位字节中从位 1 到位 6 构成值域。

由短操作码标记表示的可执行名是表 25-1 中值域的二进制值所对应的名。

25.2.5 类型/值标记

类型/值标记类型有 32 个子类型。类型八位字节的高 3 位标识类型/值的类型，低 5 位说明子类型。标记的长度和表示值的类型隐含在标记类型和子类型中。类型/值标记构造如下：

——类型八位字节的二进制值是表 25-1 中表示的值的一种。

——类型八位字节后面跟着一个或多个构造值域的八位字节。值域中八位字节的数目和值域说明的值也在表 25-1 中给出。

表 25-1 类型/值标记格式

类型字节	值长度	值域
64	1	可执行名对应于值域表示的二进制值，如同附录 A 中的说明
65	1	可执行名对应于值域表示的二进制值加上 256，如同附录 A 中的说明
66～67	—	保留
68	2	有符号整型，它的值是由紧跟类型八位字节＃＜X3 的二个字节来指定
69	4	有符号整型，它的值是由紧跟类型字节的四个八位字节来指定
70	4	浮点实数，它的值由紧跟类型字节的四个八位字节来表示
71	3	紧跟类型字节的三个八位字节表示的实数。若紧跟类型字节的八位字节的值为 r，且余下的两个八位字节为 n，则表示的值为$(n/2^r)$
72	5	紧跟类型字节的五个八位字节表示的实数。若紧跟类型字节的八位字节的值为 r，且余下的四个八位字节为 n，则表示的值为$(n/(2\times T^r))$
73～95	—	未分配

注

1 类型字节值为 64 或 65 的标记的值域所表示的值与由标记的低 13 位所表示的二进制值相等。

2 类型字节值为 66 和 67 所对应的子类型为增加可执行名值而保留。

25.2.6 类型/长度/值标记

类型/长度/值标记是可变长度的标记。跟在类型字节后面的一个、二个或八位字节说明了值域中字节的数目。所有这些字节被称为长度字节。

类型字节值、对应于长度域的长度、值的类型和编码在表 25-2 中给出。

表 25-2 类型/长度/值 标记

类型字节值	长度域	值域
96	1	在纯文本编码中表示的一个可执行名
97	1	在纯文本编码中表示的一个字面名
98	1	一个八位字节串
99	2	一个八位字节串
100	2	一个数据块

101	4	一个数据块
102	2	一个不完整数据块。以下标记或是一个数据块标记(类型八位字节=100,或 101),或是另一个不完整数据块标记(类型字节=102)
103	2	一个过程
104	2	一个齐次数值向量
105～125		未分配
126	1	由纯文本编码表示的一个结构名
127	2	一个八位字节序列,它表示一个加密标记的序列。值域的前两个字节包含一个加密标识符。其余字节包含一个加密表示的标记序列

25.2.7 短整型标记

短整型标记表示一个范围在－4096 到 28671 间的整数。整数的值是类型字节的值减去 36864。

25.2.8 操作符的编码值

由 SPDL 语法定义的操作符可以由值为操作符名的类型/长度/值标记来表示,或者由编码值或操作码标识的操作符标记来表示。表示 SPDL 操作符的操作码在附录 A 中给出。

25.3 纯正文结构交换格式

下文叙述 SPDL 文本结构的 SGML 编码。

```
〈! --This DTD makes reference to ELEMENTs"fontset"and"strucnm"
as described in"ISO/IEC 9541-2: DTD Font Attribute Set//EN"--〉
〈! DOCTYPE SPDL [
〈! ELEMENT SPDL--(document | resdef)〉
〈! --List of widely-used ELEMENTs--〉
〈! ELEMENT boolean-o EMPTY〉
〈! ATTLIST boolean
value (true | false)"true"〉
〈! ELEMENT comment--(#CDATA)〉
〈! ELEMENT int o o(#PCDATA)--integer〉
〈! ELEMENT name o o(#PCDATA)〉
〈! ELEMENT number o o(#PCDATA)--integer or real〉
〈! ELEMENT objid o o(strucnm)〉
〈! ELEMENT octetstr o o(#PCDATA)〉
〈! ELEMENT printabl o o (#PCDATA)--Printable String〉
〈! ELEMENT refname o o(#PCDATA)〉
〈! ELEMENT tokenseq o o(#CDATA)〉
〈! --Description of document--〉
〈! ELEMENT document-o(pageset | picture)*〉
〈! ELEMENT pageset-o(prologue?,psbody?)〉
〈! ATTLIST pageset
   spdlid CDATA #REQUIRED〉
〈! ELEMENT picture-o(prologue?,picbody?)〉
〈! ATTLIST pageset
   spdlid CDATA #REQUIRED
   cntnttyp CDATA #REQUIRED〉
〈! ELEMENT prologue o o(extdecls?,infdecl?,resdefs?,
```

```
        resdecls?,dpidecls?,ctxtdecl?,dictgens?,
        setups?)〉
    〈! ELEMENT infdecl-o(hint)*〉
    〈! ELEMENT hint o o(refname,hintvalu)〉
    〈! ELEMENT hintvalu o o(#PCDATA)〉
    〈! ELEMENT ctxtdecl -o(name)+〉
    〈! ELEMENT dictgens-o(dictgen)*〉
    〈! ELEMENT dictgen o o(tokenseq)*〉
    〈! ATTLIST dictgen
    dictid CDATA #REQUIRED
    size CDATA #REQUIRED〉
    〈! ELEMENT setups-o(setup)*〉
    〈! ELEMENT setup o o(tokenseq)*〉
    〈! ELEMENT extid o o(name,objid)〉
    〈! ELEMENT extdecls-o(extdecl)*〉
    〈! ELEMENT extdecl o o(extid,structid)〉
    〈! ATTLIST extdecl
    structyp CDATA #REQUIRED〉
    〈! ELEMENT structid o o(objid | octetstr)〉
    〈! ELEMENT resdecls-o(resdecl)*〉
    〈! ELEMENT resdecl o o(intid,restype,resid)〉
    〈! ELEMENT intid o o(name)〉
    〈! ELEMENT restype o o(#PCDATA)〉
    〈! ELEMENT resid o o(refname)〉
    〈! ELEMENT resdefs o o(resdef)*〉
    〈! ELEMENT resdef-o(resid,restype,resspec?)〉
    〈! ATTLIST resdef
        spdlid CDATA #REQUIRED
        funcid(define | undefine)"define"〉
    〈! ELEMENT resspec o o(fobjspec | gimspec | fimspec | colrspec |
pattspec | formspec)〉
    〈! ELEMENT fobjspec-o(frefspec | bfntspec | namfspec | cfntspec |
cnsfspec)〉
    〈! ELEMENT frefspec-o(fontref,gimid)〉
    〈! ELEMENT gimid o o(refname)〉
    〈! ELEMENT bfntspec-o(octetstr)〉
    〈! ELEMENT namfspec-o(fobjid,gimid?)〉
    〈! ELEMENT fobjid o o(refname)〉
    〈! ELEMENT cfntspec-o(fmapprms?,fimid,fobjlist)〉
    〈! ATTLIST cfntspec
        fmaptype CDATA #REQUIRED〉
    〈! ELEMENT fimid o o(refname)〉
    〈! ELEMENT fobjlist o o(fobjspec)+〉
```

```
<! ELEMENT fmapprms-o(subsvect)?>
<! ATTLIST fmapprms
   escchar CDATA 255
   shiftin CDATA 14
   shiftout CDATA 15>
<! ELEMENT subsvect o o(octetstr)>
<! ELEMENT cnsfspec-o(tokenseq)+>
<! ELEMENT gimspec-o(gidlist?)>
<! ATTLIST gimspec
   mapsize CDATA #REQUIRED>
<! ELEMENT gidlist o o(gid)+>
<! ELEMENT gid o o(name | strucnm | iso10036)>
<! ELEMENT iso10036 o o(name)>
<! ELEMENT fimspec-o(int)+>
<! ATTLIST fimspec
   mapsize CDATA #REQUIRED>
<! ELEMENT colrspec-o(tokenseq)>
<! ATTLIST colrspec
   type CDATA #REQUIRED
   primary CDATA #OPTIONAL>
<! ELEMENT pattspec-o(tokenseq)>
<! ELEMENT formspec-o(tokenseq)>
<! ELEMENT dpidecls-o(dpidecl)*>
<! ELEMENT dpidecl o o(dpiname,(mediadpi | clrsdpi |
   copydpi | pgseldpi | mdseldpi | curmddpi | clrsldpi |
   curcldpi | plexdpi | xshftdpi | yshftdpi | odplxdpi |
   bptypdpi | opostdpi | stackdpi | finshdpi | comntdpi |
   stmsgdpi | ndmsgdpi | tmoutdpi | abrtpdpi))>
<! ELEMENT dpiname o o(objid)>
<! ELEMENT runvect o o(int,(name | int))*>
<! ELEMENT proplist o o(propname,propvalu)*>
<! ELEMENT propname o o(objid)>
<! ELEMENT propvalu o o(#PCDATA)--as appropriate for particular
   property>
<! ELEMENT mediadpi-o(name,proplist)>
<! ELEMENT clrsdpi-o(name,refname+)>
<! ELEMENT copydpi-o(int)>
<! ELEMENT pgseldpi-o(runvect)>
<! ELEMENT mdseldpi-o(runvect)>
<! ELEMENT curmddpi-o(name)>
<! ELEMENT clrsldpi-o(runvect)>
<! ELEMENT curcldpi-o(name)>
<! ELEMENT plexdpi-o(objid)>
```

〈! ELEMENT xshftdpi-o(number)〉
〈! ELEMENT yshftdpi-o(number)〉
〈! ELEMENT odplxdpi-o(boolean)〉
〈! ELEMENT bptypdpi-o(objid)〉
〈! ELEMENT opostdpi-o(runvect)〉
〈! ELEMENT stackdpi-o(proplist)〉
〈! ELEMENT finshdpi-o(proplist)+〉
〈! ELEMENT comntdpi-o(printabl)〉
〈! ELEMENT stmsgdpi-o(printabl)〉
〈! ELEMENT ndmsgdpi-o(printabl)〉
〈! ELEMENT tmoutdpi-o(int)〉
〈! ELEMENT abrtpdpi-o(objid)〉
]〉

25.4 纯正文内容交换格式

标记序列组成SPDL内容。每一标记代表内容处理器处理的信息单位。下文说明SPDL内容中标记的纯正文编码。

25.4.1 标记结构和定界符

每一个标记通过一个或多个GB 1988编码的八位字节(以后称作字符)的序列来编码。标记结束由一个或多个空白字符Null、Tab、Line Feed、Form Feed、Carriage Return、Space来标识。字符(,),〈,〉,[,],{,},/,%作为句法的定界符,这些字符都能中止标记,但其本身并不是标记的一部分。

任何出现在字符%之后的八位字节串、数据块或数据块续元(DataBlockContinuation)将是注释。注释包括介于%和下一个回车、换行、换页字符之间的所有字符。字符%和由它定界的注释被作为一个空格字符来处理。

25.4.2 标记类型 Token types

在纯正文内容编码中有两种标记类型:复合标记和基本标记。一个复合标记可以包含别的复合标记和/或基本标记。在纯正文内容编码中仅仅复合标记是可执行向量。

基本标记中不包含别的标记。基本标记的例子包括八位字节串、整数和名字。当在内容中遇到复合标记时,内容处理器先处理每一个遇到的标记,然后再处理复合标记。

25.4.3 对象类型的编码

以下详述每一个SPDL对象类型的纯正文编码。对操作符类型对象的编码在附录A中说明。

25.4.3.1 Boolean

在内容交换格式中没有类型Boolean的表示。通过操作符True和False的执行,在内容中可创建类型Boolean的对象。

25.4.3.2 词典 Dictionary

在内容交换格式中没有类型Dictionary的表示。通过操作符MakeDictionary或MakeandStore-Dictionary的执行,在内容中可创建类型Dictionary的对象。

自定界操作符标记〈〈和〉〉(各自表示Mark和MakeandStoreDictionary;见附录A)可以用来清楚地在定界的内容中创建的Dictionary:

〈〈token token …token〉〉

25.4.3.3 整型 Integer

在交换格式中有两种整型表示。一种表示包括一个可选符号,后面紧跟着一个或多个十进制数。该数被看作为一个有符号的十进制整数,并被转换为一个SPDL整数对象。

另一表示形式为base#number,这里base是2到36之间的十进制数,number根据base进行转换,

它由 0 到 base-1 间的数组成。超过 9 的位用字形 A～Z 表示(或 a～z)。number 作为一个无符号整数,并被转换为一个 SPDL 整数对象。

若任一种表示的数超过了依赖于实现的整数范围,此数在压入操作数栈前先被转换为一个 SPDL 实数对象。

25.4.3.4 Mark

在内容交换格式中没有类型 Mark 的表示。通过执行操作符 Mark,可在内容中创建类型 Mark 的对象。

25.4.3.5 Name

类型 Name 的对象有两种表示:一种是可执行 Names,另一是字面(Literal)Names。

25.4.3.5.1 可执行 Name

可执行 Name 的表示由拉丁字形中的字形组成:a,b,c,…,z;A,B,C,…,Z;或一个全停符(fullstop)。序列中每一后继字符可从拉丁字形表中的字形、十进制数 0,1,…,9、下划线或全停符中选择。

25.4.3.5.2 字面 Name

字面 Name 的表示由字符/、后面跟着拉丁字形:a,b,a,…,z;A,B,C,…,Z;或一个全停符组成。序列中每一后继字符可从拉丁字形表中的字形、十进制数 0,1,…,9,下划线、冒号:或全停符中选择。仅当 Name 用作为 GlyphID(见 16)时,在构造 SPDL 文件时在 Name 后面使用字符。

25.4.3.6 Null

在内容交换格式中没有类型 Null 的表示。通过执行操作符 Null,可在内容中创建类型 Null 的对象。

25.4.3.7 八位字节串

一个八位字节串的表示形式为:

(xyz)

这里 xyz 表示零个或多个八位字节的序列。八位字节串由匹配的圆括号来定界。在八位字节串中空格字符是作为普通字符,而不是定界符。在八位字节串标记中可能出现一对括号( )连同其中零个或多个八位字节,它们将作为正常字符而不是八位字节串标记。

字符\是对引用不匹配的括号、非打印字符、和字符\本身的转意。紧跟其后的字符决定了正确的含义。

\r	回车(Carriage Return)
\t	水平制表符(Horizontal Tab)
\b	回空格(BackSpace)
\f	换页(Form Feed)
\\	\
\(	(
\)	)
\ddd	任意八字节(Arbitrary Octet)

若\后面的字符不在上表中,则\被略过,字符按常意解释。在八位字节串中格式\ddd 可用于包括任意个八位,\后面的三个八数字用于说明八位字节串中可能有的八位字节值。

25.4.3.8 路径 Path

在内容交换格式中没有路径表示。通过执行路径构造操作符(在第 18 章中定义),可在内容中创建路径。

25.4.3.9 实型 Real

内容交换格式中实型数的表示形式是:一个可选符号,后面是一个或多个十进制数的序列,序列的头或中间有一个全停位(小数点),后面跟着指数。指数若存在,其构成是字符 E 或 e 后面跟着一个可选

符号，再后面是一个或多个十进制数。

25.4.3.10 SaveState

在内容交换格式 中没有 SaveState 的表示。通过执行操作符 SaveState，可在内容中创建 SaveState。

25.4.3.11 流对象 StreamObject

在内容交换格式中没有类型 StreamObject 的表示。通过执行操作符 Filter，可在内容中创建类型 StreamObject 的对象。

25.4.3.12 向量 Vector

在 SPDL 中有两种向量对象：字面向量和可执行向量(过程)。

25.4.3.12.1 字面向量

在内容交换格式中没有字面向量的表示。通过执行操作符 MakeVector 或 MakeandStoreVector，可在内容中创建字面向量。

自定界操作符标记[和](各自表示 Mark 和 MakeandStoreVector；见附录 A)可用于在内容中区别字面向量的创建：

[token token …token]

25.4.3.12.2 可执行向量

在内容交换格式中一个可执行向量，或长度为 N 的一个过程的表示形式为：

(token1 token2 …tokenN)

这里每一个标记(token)表示向量的一个元素。可执行标记导致可执行 SPDL 对象被存储在向量中而不是被执行。

25.4.3.13 操作符 Operators

除了自定界标记[，]，〈〈，和〉〉，在纯文本内容交换格式中没有类型 Operator 的对象表示。在系统词典(SystemDict)中 Operator 对象与其名字(Names)相结合，在内容中(和在纯文本内容交换格式中)通过它们的名字来存取。附录 A 列出了每个 SPDL 操作符和它的名字(若合适，和自定界标记)。通过名字，可在纯文本内容交换格式中存取操作符。

25.4.4 内含(In-line)光栅图形图像数据标记

在纯文本文件中有两种标记类型用来对光栅图形图像数据进行编码。.这两种标记类型的讨论见第 17 章。

25.4.4.1 数据块标记 DataBlock Token

数据块标记对 ASCII85-编码的数据进行编码。数据块标记的表示形式为：

〈～xyz～〉

这里 xyz 表示零个或多个 ASCII85-代码的八位字节序列(见 22.2.2)。

25.4.4.2 数据块续元(DataBlockContinuation)标记

数据块续元标记对 ASCII85-代码的数据进行编码。数据块续元标记的表示形式为：

〈#xyz#〉

这里 xyz 表示零个或多个 ASCII85-代码的八位字节序列(见 22.2.2)。

26 适应性

26.1 交换格式的 SPDL 文件

一个合格的文件其句法应符合第 6～16 章中的规定，且其描述应该符合 17.1～17.2 中的规定。

每一个合格的文件至少应该有一个 PAGESET，PAGE 或 RESOURCE SPECIFICATION 结构元素。

26.2 SPDL 操作过程

一个合格的操作过程应该生成合格的 SPDL 文件。

26.3 SPDL 表示过程的实现

任何实现过程将对所实现的文件的复杂性和大小有一定的限制。

26.3.1 结构

26.3.1.1 资源定义

每一个合格的表示过程应该能够处理隶属于一个 DOCUMENT 的 RESOURCE DEFINITION 结构元素，并且应该使它们定义的资源适用于内容处理。一个合格的表示过程并不需要在最高结构层增加由 RESOURCE DEFINITION 定义的资源，但是，如果上述要求并未完成的话，必须报告一个异常。

26.3.1.2 所要求的资源

一个合格表示过程可用的一组资源至少应该包括对应于下面一些对象名的由公用对象标识符值定义的资源：

资源类型	对象名
Data Source	DataSource/Document
Filter	Filter/BitPerPel
	Filter/OctetPerPelGray
FontObjects	ISOSerif
	ISOSerif/BoId
	ISOSerif/Italic
	ISOSerif/Bolditalic
	ISOSanSerif
	ISOSanSerif/Boid
	ISOSanSerif/Italic
	ISOSanSerifBoIdItalic
	ISOMonospace
	ISOMonospace/Bold
	ISOMonospace/ItaIic
	ISOMonospace/BoldItalic
Glyph index Map	GlyphIndexMap/IR/nn
	GlyphIndexMap/LatinPubishing
Color Space	ColorSpaceType/CIELAB
	ColorSpaceType/CIEBasedABC
	ColorSpaceType/DeviceGray
	ColorSpaceType/DeviceRGB
	ColorSpaceType/DeviceCMYK
Filter	Filter/RunLengthDecode
	Filter/CCITTFAXDecode
	Filter/NullDecode
	Filter/ASCIIHexDecode
	Filter/ASCII85Decode

索引和命名的彩色之间的一组接口可通过 Color SpaceTypes 的彩色空间来表示

——ColorSpaceType/Indexed

——ColorSpaceType/NamedColor

它将由每一个合格的表示过程来支持。

26.3.1.3 外部结构元素

每一个表示过程有一个与它相对应的有效值范围(可能是 Null),从该范围可以求得外部结构元素。在这个范围内部不需要任何外部结构元素。

26.3.1.4 文件生成指令

每一个合格的表示过程将可支持由第 8 章定义的所有文件生成指令,这一功能是通过或者支持所定义的语义,或者指定撤退动作来实现的。

26.3.1.5 结构语法

一个合格的表示过程必须能够处理任何合格的 SPDL 文件结构。

26.3.2 内容

任何一个实现过程对所要处理的文件的复杂性和大小有一个限制,在该限制范围内,一个合格的表示过程必须能够处理任何合格 SPDL 文件的所有内容元素。

26.3.2.1 构造和保存对象的能力

所有合格的实现过程对内容处理最少将有如下功能:

——在操作数栈存储 500 个整数;

——构造一个有 10000 个整数的向量;

——存储由 250 个字形表示的字符串;

——构造一个包含 1000 个名字—值对(其值是整数)的词典;

——构造一个包含 1500 个路径元素的路径,而这些路径元素由一系列标记(tokens)来建立,其格式如下:

{BeginPathSegment LineTo LineTo…LineTo};

——在上下文栈中存储 20 个 DictionaryReference;

——存储 15 个状态保存级;

——存储 30 个图形状态保存级(包括保存全局 SaveState 的操作结果)。

这些要求是独立的:对一个合格的具体实现并不一定要求能同时存储一个以上的上述数据对象。

26.3.2.2 表示值的能力

所有合格的实现过程在进行内容处理时应至少能表示以下一些值:

——范围在$(-2^{31}-1)$到2^{31}之间的整数;

——由 IEEE854—1987 所指定范围和精度的实数。

附 录 A
（标准的附录）
操作符编码

所有 SPDL 的操作符都和系统词典(SystemDict)中的名字相关。在内容交换格式的纯正文编码中，类型为操作符的对象没有明确编码，而是给出在 SystemDict 中操作符名字的编码，这样就可以执行与之相关的操作符，它们在文件内容中可以是隐含的，也可以是显式的。

在交换格式的二进制编码中，SystemDict 中的操作符的名字可以表示的形式为：类型/长度/值，这里值为名字；也可以用编码值或操作码来表示。

表 A1 中列出标准的页面描述语言 SPDL 的操作符，这些操作符在系统词典 SystemDict 中的名字可以使用纯文本或二进制的内容交换格式，操作码使用二进制内容交换格式。

注：与操作符 Mark 有关的附加的名字(MakeandStoreVector 和 MakeandStoreDictionary 操作符)是为了在纯文本内容交换格式中使用方便而引进的表示法。

表 A1

操 作 符	功 能	名 字	操 作 码
AbsoluteValue	绝对值	abs	60
Add	加	add	49
AnchorSearch	锚定搜索	anchorsearch	61
And	与	and	62
AppendPath	附加路径	uappend	63
ArcTangent	反正切	atan	64
ArcToClockwise	顺时针画弧	arcn	32
ArcToCounterClockwise	逆时针画弧	arc	31
BeginPathSegment	开始路径段	moveto	18
BeginPathSegmentRelative	相对开始路径段	rmoveto	19
Capacity	容量	capacity	65
Ceiling	取上限	top	66
CheckifExecutable	检查是否可执行	echeck	67
CheckifReadable	检查是否可读	rcheck	68
Checkifwriteable	检查是否可写	wcheck	69
ClearStack	清除栈	clear	70
ClearToMark	清除栈到标记	cleartomark	71
ClipPath	裁剪路径	clip	27
ClipPathEvenOdd	奇偶裁剪路径	eoclip	28
ClosePathSegment	闭合路径段	closepath	35
Concat	连接(矩阵乘)	concat	72
ConcatT	连接矩阵 T	concatmatrices	13
ContextStack	上下文栈	dictstack	73
ConvertToExecutable	变为可执行	cve	74
ConvertToIdentifier	变为标识符	cvid	75
ConvertToInteger	变为整数	cvi	76
ConvertToReal	变为实数	cvr	77
ConvertToString	为字符串	cvs	78
Copy	拷贝	copy	79
Cosine	余弦	cos	80
Count	计数	count	81

表 A1(续)

操 作 符	功 能	名 字	操 作 码
CountToMark	计数到标记	counttomark	82
CurveTo	画曲线	curveto	33
CurveToRelative	按相对位置画曲线	rcurveto	34
Define	定义	def	7
DefineFont	定义字型	deffontobj	83
Divide	除	div	84
Dup	复制	dup	2
EntriesUsed	已用(词典)项数	entriesused	85
Equal	等于	eq	46
Exchange	互换	exch	86
Execute	执行	exec	87
ExecuteForm	执行模板	execform	88
ExecuteTrapped	执行陷阱	stopped	89
Exit	退出	exit	53
Exponentiate	指数	exp	90
False	逻辑假	false	45
FillPath	填充路径	fill	42
FillPathEvenOdd	奇偶填充路径	eofill	43
Filter	过滤器	filter	91
FindFont	查找字型	findfontobj	92
FindResource	查找资源	findres	93
Floor	取下限	bottom	94
For	For 循环	for	54
ForAll	ForAll 循环	forall	55
Get	取	get	3
GetBlackGeneration	取黑色生成	currentblackgeneration	95
GetColour	取当前色	currentcolor	15
GetColourRendering	取当前色还原	currentcolorrendering	96
GetColourSpace	取彩色空间	getcolorspace	36
GetCurrentDictionary	取当前词典	currentdict	97
GetDashPattern	取线型	currentdash	98
GetDeviceDescription	取设备描述	getdevinfo	99
GetHalftone	取半色调值	currenthalftone	100
GetInterval	取区间(数据)	getinterval	101
GetMitrelimit	取斜接限定值	currentmiterlimit	102
GetOverPrint	取叠印	currentoverprint	103
Getpath	取路径	getpath	26
GetPoisition	取当前点位置	currentpoint	20
GetRootFont	取根字型	rootfont	104
GetSelectedFont	取当前字型	currentfont	17
GetStrokeAdjust	取画线调整	currentstrokeadjust	105
GetStrokeEnd	取当前线端	currrentlinecap	106
GetStrokeJoin	取当前线交方式	currentlinejoin	107
GetStrokeWidth	取当前线宽	currentlinewidth	108
GetTest	取测试(结果)	known	4
GetTrans	取当前变换	gettrans	109
GetTransfer	取当前色传递函数	currentcolortransfer	110

表 A1(续)

操 作 符	功 能	名 字	操 作 码
GetUnderColourRemoval	取底色去除	currentundercolorremoval	111
GetValue	取值	load	5
GetValueTest	取值测试	where	6
GlyphToPath	沿路径输出字形	charpath	112
GreaterOrEqual	大于或等于	GE	113
GreaterThan	大于	GT	114
If	如果	if	47
IfElse	否则	ifelse	48
ImageRasterElement	光栅图像元素	image	115
Index	索引	index	116
IntegerDivide	整数除法	idiv	117
LessOrEqual	小于或等于	LE	118
LessThan	小于	LT	119
LineTo	画线	lineto	29
LineToRelative	按相对位置画线	rlineto	30
Logarithm	对数	log	120
LogicalShift	逻辑移位	bitshift	121
Loop	循环	loop	56
MakeDictionary	生成词典	dict	122
MakeExecuteOnly	置只执行	executeonly	123
Makepattern	生成图案	makepattern	124
MakeReadOnly	置只读	readonly	125
MakeString	生成串	string	126
MakeVector	创建向量	array	127
MakeandStoreDictionary	生成并存储词典	››	128
MakeandStoreVector	创建并存储向量	]	129
Mark	标记	mark	8
Mark	标记	[	8
Mark	标记	‹‹	8
MaskBitMap	掩膜位图	imagemask	130
Multipply	乘	mul	131
NaturalLogarithm	自然对数	In	132
Negate	取负	neg	133
NewPath	新路径(清除旧路径)	newpath	24
Noop	空操作	noop	1
Not	取反	not	134
NotEqual	不等	ne	51
Null	空	null	52
OpenFont	打开字型	openfont	135
Or	或	or	136
Pop	弹出	pop	137
PopcontextStack	弹出上下文栈	end	138
Print	打印	print	139
PushContextStack	压入上下文栈	begin	140
Put	替换	put	9
putInterval	区间替换	putinterval	142
PutValue	存值	store	10

表 A1(续)

操 作 符	功 能	名 字	操 作 码
PutWMode	存储写模式	prtWMode	143
RaiseError	引发错误	raiseerror	144
RaiseException	引发异常	stop	145
RaiseWarning	引发警告	raisewarning	146
Rand	随机数	rand	147
RandSetState	置随机数状态	srand	148
Remainder	余数取模	mod	149
Repeate	重复	repeat	57
RestoreGraphicsState	恢复图形状态	grestore	59
RestoreGraphicsStateXCP	恢复除当前位置外的图形状态	grestoreXCP	150
RestoreSavedGraphicsState	恢复保存的图形状态	grestoreall	151
RestoreState	恢复状态	restore	152
Roll	滚动	roll	153
Rotate	旋转变换	rotate	141
RotateT	旋转变换矩阵 T	rotateT	154
Round	取整	rnd	155
SaveGraphicsState	保存图形状态	gsave	58
SaveState	保存状态	save	156
Scale	比例变换	scale	11
ScaleT	比例变换矩阵 T	scaleT	187
ScaleFont	字模缩放	scalefont	157
Search	查找	search	158
SetBlackGeneration	置黑色生成	setblackgeneration	159
SetColour	置彩色	setsolidcolor	14
SetColourRendering	置彩色描绘	setcolorrendering	160
SetColourScreenAngle	置彩色屏幕角度	setscreenangle	161
SetColourScreenFrequency	置彩色屏幕频率	setscreenfreq	162
SetColourSpace	置彩色空间	setcolorspace	163
SetColourSpotFunction	置彩色网点函数	setspotfunction	164
SetColourTransfer	置彩色传送函数	setcolortransfer	165
SetDashPattern	置线型	setdash	40
SetFont	置字型	setfont	16
SetHalftone	置半色调	sethalftone	166
SetMitreLimit	置斜接限定值	setmiterlimit	167
SetOverPrint	置叠印	setoverprint	168
SetPath	置路径	setpath	25
SetpatternColour	置图像颜色	setpatterncolor	169
SetPosition	置当前点位置	moveto	18
SetpositionRelative	置当前点相对位置	rmoveto	19
SetSereenAngle	置屏幕角度	setscreenangle	170
SetScreenFrequency	置屏幕频率	setscreenfreq	171
SetSpotFunction	置光点函数	setspotfunction	172
SetStrokeAdjust	置画线调整	setstrokeadjust	173
SetStrokeEnd	置线端	setlinecap	39
SetStrokeJoin	置线交方式	setlinejoin	38
SetStrokeWidth	置线宽	setlinewidth	37

表 A1(完)

操 作 符	功 能	名 字	操 作 码
SetTrans	置当前变换	settrans	174
SetTransfer	置传送函数	settransfer	175
SetUnderColourRemoval	置底色去除	setundercolorremoval	176
ShowGlyph	输出字形	showglyph	21
ShowString	输出字符串	show	22
ShowStringEscapedx		xshow	23
ShowStringEscapedxy		xyshow	177
ShowStringEscapedy		yshow	178
Sine	正弦	sin	179
SquareRoot	平方根	sqrt	180
StoreVector	存储向量	astore	181
StrokePath	画路径	storke	41
Subtract	减	sub	50
TransformFont	变换字型	makefont	182
Translate	平移变换	translate	12
TranslateT	平移变换矩阵 T	translateT	188
True	逻辑真	true	44
Truncate	截尾	trunc	183
Type	对象类型	objtype	184
VectorLoad	取向量	alode	185
Xor	异或	xor	186

附 录 B

(标准的附录)

ASN.1 由本标准定义的对象标识符和 SGML 格式的公用标识符

本附录说明本标准为标识信息对象所定义的公用对象标识符值。

每个信息对象有一个公用对象标识值,并在定义文本中指定一个对象名。对于每一个这样的对象名,本附录说明如下:

——公用标识符按 ISO 9070 中的规则定义

——对象标识符值按 GB/T 16262 中的规则定义

在表 B 1 中,对象名"GlyphIndexMap/AFII/nn"和 ASN.1 中的数字串"60n"相对应,是指形式为"GlyphIndexMap/AFII/nn"的一个对象名和 ASN.1 数字串"60n"相对应,这里的"nn"是一个以非"0"数字开头的十六进制数字序列;"n"是由 16 进制数字序列"nn"表示的基数字。

B1 公用标识符

本标准为每一个信息对象指定一个对象名 object-name,这些信息对象是由公用标识符来标识的,公用标识符的规范字符串格式为:已注册的所有者前缀 GB/T ××××后接"//"和 object-name 字符序列。

B2 对象标识符值

由本标准指出的对象名为 object-name 的 ASN.1 对象标识符值的格式为:前缀 iso(1)standard(0)10180(10180),后接由表 B1 中说明的与对象名相关的数字序列。

B3 对象名和对象标识值的关系

下表列出由本标准定义的对象名和标识对象的 ASN.1 对象标识符的数字序列。

表 B1 对象名和 ASN.1 对象标识符值

对 象 名	ASN.1 数字序列	对 象 名	ASN.1 数字序列
SPDL	0	DPI/Medium/Color/nocolor	5 0 4 7
ContentType/SPDLBinary	1 0	DPI/Medium/Weight	5 0 5
ContentType/SPDLClearText	1 1	DPI/Medium/Opacity	5 0 6
ColorSpaceName/CIELAB	2 0	DPI/Medium/Prefinish	5 0 7
ColorSpaceName/CIELUV	2 1	DPI/Medium/Prefinish/plain	5 0 7 0
ColorSpaceName/CIEBasedABC	2 2	DPI/Medium/Prefinish/preCutTab	5 0 7 1
ColorSpaceName/CIEBasedA	2 3	DPI/Medium/Prefinish/continuous	5 0 7 2
ColorSpaceName/DeviceRGB	2 4	DPI/Medium/Prefinish/adhesiveLabels	5 0 7 3
ColorSpaceName/DeviceCMYK	2 5	DPI/Medium/Prefinish/envelopePlain	5 0 7 4
ColorSpaceName/DeviceGray	2 6	DPI/Medium/Prefinish/envelopeWindow	5 0 7 5
ColorSpaceName/Indexed	2 7	DPI/Medium/HoleCount	5 0 8
ColorSpaceName/NamedColour	2 8	DPI/Medium/OrderedCount	5 0 9
ColorSpaceName/Highlight	2 9	DPI/Medium/FinishEdge	5 0 10
DataSource/Document	3 0	DPI/Medium/Labels	5 0 11
Filter/RunLengthDecode	4 0	DPI/Medium/Message	5 0 12
Filter/CCITTFAXDecode	4 1	DPI/Copies	5 1
Filter/NullDecode	4 2	DPI/PageSelect	5 2
Filter/ASCIIHexDecode	4 3	DPI/MediumSelect	5 3
Filter/ASC1185Decode	4 4	DPI/Plex	5 4
DPI	5	DPI/Plex/simplex	5 4 0
DPI/Medium	5 0	DPI/Plex/duplex	5 4 1
DPI/Medium/Name	5 0 1	DPI/Plex/tumbleDuplex	5 4 2
DPI/Medium/Name/default	5 0 1 0	DPI/XImageShift	5 5
DPI/Medium/Size	5 0 3	DPI/YImageShift	5 6
DPI/Medium/Color	5 0 4	DPI/OnlyOnDuplex	5 7
DPI/Medium/Color/white	5 0 4 0	DPI/BreakPageType	5 8
DPI/Medium/Color/pink	5 0 4 1	DPI/BreakPageType/none	5 8 0
DPI/Medium/Color/yellow	5 0 4 2	DPI/BreakPageType/terse	5 8 1
DPI/Medium/Color/buff	5 0 4 3	DPI/BreakPageType/verbose	5 8 2
DPI/Medium/Color/goldenrod	5 0 4 4	DPI/OutputPosition	5 9
DPI/Medium/Color/blue	5 0 4 5		
DPI/Medium/Color/green	5 0 4 6		

表 B1(完)

对 象 名	ASN.1 数字序列	对 象 名	ASN.1 数字序列
DPI/Stacking	5 10	DPI/Finishing/BindingType/perfect	5 11 7 3
DPI/Stacking/Collated	5 10 0	DPI/Finishing/BindingType/spiral	5 11 7 4
DPI/Stacking/Offset	5 10 1	DPI/Finishing/BindingType/default	5 11 7 5
DPI/Stacking/Slipsheet	5 10 2	DPI/Finishing/BindingColor	5 11 8
DPI/Finishing	5 11	DPI/Finishing/BindingColor/black	5 11 8 0
DPI/Finishing/ReferenceSize	5 11 0	DPI/Finishing/BindingColor/blue	5 11 8 1
DPI/Finishing/ReferenceEdge	5 11 1	DPI/Finishing/BindingColor/gray	5 11 8 2
DPI/Finishing/Locations	5 11 2	DPI/Finishing/BindingColor/brown	5 11 8 3
DPI/Finishing/Locations/HeadCount	5 11 2 0	DPI/Finishing/BindingColor/default	5 11 8 4
DPI/Finishing/Locations/ProcessOffset	5 11 2 1	DPI/Finishing/FinishingOperation	5 11 9
DPI/Finishing/Locations/HeadLocations	5 11 2 2	DPI/Finishing/convenienceStaple	5 11 9 0
DPI/Finishing/Inserts	5 11 3	DPI/Finishing/edgeStitching	5 11 9 1
DPI/Finishing/Inserts/InsertName	5 11 3 0	DPI/Finishing/binding	5 11 9 2
DPI/Finishing/Inserts/InsertSize	5 11 3 1	DPI/Finishing/saddleStitching	5 11 9 3
DPI/Finishing/Inserts/InsertEdge	5 11 3 2	DPI/Finishing/punching	5 11 9 4
DPI/Finishing/Inserts/InsertTopSurface	5 11 3 3	DPI/Finishing/perforating	5 11 9 5
DPI/Finishing/Inserts/InsertBin	5 11 3 4	DPI/Finishing/slitting	5 11 9 6
DPI/Finishing/Inserts/InsertAfter	5 11 3 5	DPI/Finishing/trimming	5 11 9 7
DPI/Finishing/TrimmedSize	5 11 4	DPI/Finishing/folding	5 11 9 8
DPI/Finishing/DieCutName	5 11 5	DPI/Finishing/inserting	5 11 9 10
DPI/Finishing/DieCutPosition	5 11 6	DPI/Finishing/dieCutting	5 11 9 11
DPI/Finishing/BindingType	5 11 7	DPI/Finishing/Message	5 11 10
DPI/Finishing/BindingType/tape	5 11 7 0	DPI/DocumentComment	5 12
DPI/Finishing/BindingType/plastic	5 11 7 1	DPI/DocumentStartMessage	5 13
DPI/Finishing/BindingType/velo	5 11 7 2		

附 录 C
（标准的附录）
SPDL 字型对象描述

C0 引言

本附录提供了 SPDL 文件中 SPDL 字型对象的描述。

C1 约定

本附录中描述的所有编码元素(除了另作规定外)都用带括号的术语 OPTIONAL 来标志它。

本文中的短语"用于公用交换"(*for public interchange*)和"用于专用交换"(*for private interchange*)是指在 SPDL 文件中 SPDL 字型对象信息的交换。请勿和 ISO/IEC 9541 中指定的字型资源的交换相混淆。

本附录中所用的印刷上的重要约定如下:

——粗体字的项是八位字节串,当出现在本附录中时必须精确地按 FontObject 编码出现;

——斜体字的项是八位字节串位置标识符(placeholder),其内容用来指定一个所给的 FontObject。

——普通文本段包含了对编码元素的解释和对其内容的限制。

本附录中给出许多例子,为简短易读起见,特定义了几个位置标识符,以表示与之相对应的八位字节串:

在编码元素中斜体字 *rdef* 是位置标识符,它可以在实际编码中由下列符号之一表示:

——**def**

——**executeonly def**

——**readonly def**

同样,斜体字 *ndef* 可由下列符号之一表示:

——**def**

——**executeonly def**

——**readonly def**

——**noaccess def**

斜体字 *nput* 可由下列符号之一表示:

——**put**

——**executeonly put**

——**readonly put**

——**noaccess put**

C2 定义

在 ISO/IEC 9541 中定义的定义项(Definitions)通过引用收编在此。

本标准的第 3 章中定义的定义项(Definitions)通过引用也收编在此。

C3 SPDL 字型对象的描述

SPDL 字型对象 FontObject 编码由三部分组成,顺序为:头(Header),每个字型(PerFont)属性和字形形状信息,详见以下叙述。

C3.1 头 Header

一个 SPDL 字型对象(FontObject)的编码必须以下述形式开头:

%*FONTSTANDARD* – *STANDARDVERSION*: *TYPEFACE DATAVERSION* 这里 *FONTSTANDARD*(字型标准)在用于公用交换时应符合标准 ISO/IEC 9541-3,且在用于专用交换时可以是任何八位字节串。*STANDARDVERSION*、*TYPEFACE* 和 *DATAVERSION*(标准版本、字样和数据版本)在用于公用交换时由标准 ISO/IEC 9541 定义,用于专用交换时可以是任意八位字节串。

符合第一条要求的编码元素应为:

%% CreationData:*TIMESTAMP*

%% *PROPDATA*

其中，*TIMESTAMP* 是用于公用交换的 GB/T 16262 格林威治(Universal)时间串，当用于专用交换时可以是任意八位字节串，*PROPDATA* 见 ISO/IEC 9541 中定义。

在头段中最后的编码元素是如下形式的一行或多行。

% *DATACOPYRIGHT*

和/或

% *DSNCOPYRIGHT*

其中 *DATACOPYRIGHT* 和 *DSNCOPYRIGHT* 见 ISO/IEC 9541 中定义。

C3.2 每个字型的属性 PerFont Attributes

FontObject 编码的头后面紧接着是每个字型的属性(PerFont Attributes)，这些属性包括一些高层属性和一个 FontInfo 属性子集。本章中的高层属性 FontInfo 子集在编码的 PerFont 属性段中可以按任意次序出现。头的末尾和随之而来的 PerFont 属性段之间由下面一行来分界：

FONTDICTSIZE **dict begin**

这里，*FONTDICTSIZE* 是一个基数，其值比字型对象中实际存在的 PerFont 属性数大 1。为了计算 *FONTDICTSIZE*，FontInfo 属性子集可视为单个属性。

PerFont 属性段的尾部由下面一行来定界：

currentdict end

所有的 PerFont 属性(包括 FontInfo 属性子集)，如果在编码中存在，就必须出现在这两个定界行之间。以下一些条将给出各高层属性、FontInfo 属性子集以及 FontInfo 属性子集中的每一个属性的编码细节。

C3.2.1 FontInfo(OPTIONAL)字型信息

FontInfo 属性子集包含多种属性，它们都是可选择的，且可以按任意次序出现在 FontInfo 开始定界行之间：

开始定界行为：

/FontInfo NUMFIATTRIBUTES **dict dup begin**

其中 *NUMFIATTRIBUTES* 是 FontInfo 属性子集中实际存在的属性数。

结束定界行：

end *rdef*

在 FontInfo 属性子集中存在的属性其编码如下：

C3.2.1.1 Version(OPTIONAL)版本

/version(*DATAVERSION*) *rdef*

其中 *DATAVERSION* 用于公用交换时在 ISO/IEC 9541 中定义，用于专用交换时可以是任意八位字节串。

C3.2.1.2 Fullname(OPTIONAL)全称

/FullName(*TYPEFACENAME*) *rdef*

其中 *TYPEFACENAME* 用于公用交换时在 ISO/IEC 9541 中定义，用于专用交换时可以是任意八位字节串。

C3.2.1.3 FamilyName(OPTIONAL)字型族名

/FamilyName(*FONTFAMILY*)*rdef*

其中 *FONTFAMILY* 用于公用交换时在 ISO/IEC 9541 中定义，用于专用交换时可以是任意八位字节串。

C3.2.1.4 Weight(OPTIONAL)权

/Weight(*WEIGHT*) *rdef*

其中 *WEIGHT* 用于公用交换时在 ISO/IEC 9541 中定义，用于专用交换时可以是任意八位字节串。

C3.2.1.5 ItalicAngle(OPTIONAL)斜角

/ItalicAngle*POSTUREANGLE rdef*

其中 *POSTUREANGLE* 用于公用交换时在 ISO/IEC 9541 中定义，用于专用交换时可以是任意八位字节串。

C3.2.1.6 isFixedPitch(OPTIONAL)固定节距

/isFixedPitch*ESCCLASS rdef*

其中 *ESCCLASS* 用于公用交换时在 ISO/IEC 9541 中定义，用于专用交换时可以是任意八位字节串。

C3.2.1.7 UnderlinePosition 下划线位置

/UnderlinePosition*SCOREOFFSETX rdef*

其中 *SCOREOFFSETX* 用于公用交换时在 ISO/IEC 9541 中定义，用于专用交换时可以是任意八位字节串。

C3.2.1.8 UnderlineThickness(OPTIONAL)下划线宽度

/UnderlineThickness*SCORETHICK rdef*

其中 *SCORETHICK* 用于公用交换时在 ISO/IEC 9541 中定义，用于专用交换时可以任意八位字节串。

C3.2.2 FontName(OPTIONAL)字型名字

/FontName *FONTNAME rdef*

其中 *FONTNAME* 用于公用交换时在 ISO/IEC 9541 中定义，用于专用交换时可以是任意八位字节串。

C3.2.3 PaintType 涂色类型

/PaintType*STRUCTURE rdef*

其中 *STRUCTURE* 用于公用交换时在 ISO/IEC 9541 中定义，用于专用交换时可以是任意八位字节串。

C3.2.4 FontType 字型类型

/FontType*FONTTYPE rdef*

其中 *FONTTYPE* 是基数类型。如果字形形状技术(GlyphShape Technology)特性值(见 ISO/IEC 9541-3)是 **ISO/STANDARD**/9541-3/**TYPE**1，则 *FONTTYPE* 值为 1。当用于专用交换时，*FONTTYPE* 可以是 101 到 998 之间的任意基数类型值。

C3.2.5 FontMatrix 字型距阵

/FontMatrix[*a b c d e f*]*rdef*

或

/FontMatrix(*a b c d e*)*rdef*

根据公式 $x=aX+cY+e$

$y=bX+dY+f$

可将字形坐标系(ISO/IEC 9541 中定义)中的坐标(X、Y)转换为 SPDL 用户坐标系中的坐标(x，y)，典型的情况是：b、c、e 和 f 为 0，a 和 d 均为 1/*RELUNITS*，其中 *RELUNITS* 在 ISO/IEC 9541 中定义，且在 ISO/IEC 9541-3 中讨论。

C3.2.6 Encoding 编码

编码属性有两种形式，第一种形式为：

/Encoding*PRIVATEENCODING rdef*

其中 *PRIVATEENCODING* 可以是任意八位字节串，这种形式的编码属性仅用于专用交换。

第二种形式是一串索引到字形名字的映射，开始定界行是：

/Encoding 256 array

0 1 255{1 index exch/*DEFAULTGNAME* **put}for**

其中 *DEFAULTGNAME* 是在编码属性中索引没有明确映射到的缺省名字。

在开始定界行之后是形式如下的 0～256 个元素序列：

dup*INDEX*/*GNAME* **put**

其中 *INDEX* 是一个基数(值在 0～256 之间)表示是用来映射字形名字的索引;*GNAME* 是字形标识符,当用于公用交换时在 ISO/IEC 9541 中定义,用于专用交换时可以是任意八位字节串。

映射元素结束行为:

rdef

C3.2.7 FontBBox 字型界框

/**FontBBox**{*MINX*,*MINY*,*MAXX*,*MAXY*}*rdef*

或

/**FontBBox**[*MINX*,*MINY*,*MAXX*,*MAXY*] *rdef*

其中 *MINX*,*MINY*,*MAXX*,*MAXY* 在 ISO/IEC 9541 中 *MAXFONTTEXT* 特性(property)下定义。

C3.2.8 UniqueID(OPTIONAL)唯一标识符

/**UniqueID***UNIQUEID rdef*

其中 *UNIQUEID* 是整型,如果存在,其值应和字形形状信息段中的 UniqueID 专用属性一致。

C3.3 字形形状信息

在 PreFont 属性段之后是字形形状信息段。整个字形形状信息段通常用 **eexec** 加密算法加密(见本附录 3.3.3)。当且仅当此段数据用 **eexec** 加密算法加密后,加密数据放在定界符:

currentfile eexec

之后。

实际的形状信息(连同使用 **eexec** 加密时编码的 **eexec** 加密部分)本身由两部分组成:专用属性和字符串(CharString)。在 3.3.1 和 3.3.2 给出 **eexec** 加密前的这两部分的编码,3.3.3 给出可选择的 **eexec** 加密算法。

C3.3.1 专用属性(private)

专用属性段由一些必须的和可选的属性构成,这些属性的语义在 ISO/IEC 9541-3 中给出。在专用属性段中这些属性连同下述两种异常可以按任意次序出现。

两种异常为:

——RD、ND 和 NP 属性或其任意子集可以按任意次序出现在专用属性段之前或之中。在专用属性段之前放置这些属性是允许的,但并不推荐这样做。

——Subrs 属性必须是最后一个属性。

专用属性段开始定界行如下:

dup/**private***PRIVATESIZE* dict dup begin

其中 *PRIVATESIZE* 是基数,它指出专用属性段中存在的属性数,该数包含可出现在专用属性段之前的 RD、ND 或 NP 属性。专用属性段结束定界行为:

ND

其中 *ND* 是作为 ND 属性部分定义的八位字节串(见 3.3.1.2)。

所有的专用属性,如果出现在编码中就必须出现在这两个定界行之间(前述 RD、ND、NP 除外),以下一些条给出每个属性的编码细节。

C3.3.1.1 RD

/*RD*{**String currentfile exch readstring pop**}*rdef*

其中 *RD* 是任意八位字节串,它和 ND 属性、NP 属性所定义的八位字节串不相等,也和其他任何专用属性的名字不相等。

C3.3.1.2 ND

/*ND*{*ndef*} *rdef*

其中 *ND* 是八位字节串,它和 RD 属性、NP 属性所定义的八位字节串不相等,也和其他任何专用属性的

名字不相等。

其中 ND 是八位字节串，它和 RD 属性、NP 属性所定义的八位字节串不相等，也和其他任何专用属性的名字不相等。

C3.3.1.3 NP

/NP(nput)rdef

其中 *NP* 是八位字节串，它和 RD 属性、ND 属性所定义的八位字节串不相等，也和其他任何专用属性的名字不相等。

C3.3.1.4 BlueValues

/BlueValues[*BLUEVALUEPAIRS*]*ndef*

其中 *BLUEVALUEPAIRS* 是一个有 0 到 7 个整数对的序列。这些整数对的语义由 ISO/IEC 9541-3 给出。

C3.3.1.5 OtherBlues(OPTIONAL)

/OtherBlues[*OTHERBLUESPAIRS*]*ndef*

其中 *OTHREBLUESPAIRS* 是一个有 0 到 5 个整数对的序列。这些整数对的语义由 ISO/IEC 9541-3 给出。

C3.3.1.6 FamilyBlues(OPTIONAL)

/FamilyBlues[*FAMILYBLUESPAIRS*]*ndef*

其中 *FAMILYBLUESPAIRS* 是一个有 0 到 7 个整数对的序列。这些整数对的语义由 ISO/IEC 9541-3 给出。

C3.3.1.7 FamilyOtherBlues(OPTIONAL)

/FamilyOtherBlues[*FAMILYOTHERBLUESPAIRS*]*ndef*

其中 *FAMILYOTHERBLUESPAIRS* 是一个有 0 到 5 个整数对的序列。这些整数对的语义由 ISO/IEC 9541-3 给出。

C3.3.1.8 BlueScale(OPTIONAL)

/BlueScale*BLUESCALE ndef*

其中 *BLLUESCALE* 是数字类型。*BLUESCALE* 语义由 ISO/IEC 9541-3 定义。

C3.3.1.9 BlueShift(OPTIONAL)

/BlueShift*BLUESHIFT ndef*

其中 *BLUESHIFT* 是整数类型。其语义由 ISO/IEC 9541-3 定义。

C3.3.1.10 BlueFuzz(OPTIONAL)

/BlueFuzz*BLUEFUZZ ndef*

其中 *BLUEFUZZ* 是整数类型。其语义由 ISO/IEC 9541-3 定义。

C3.3.1.11 StdHW(OPTIONAL)

/StdHW[*STDHW*] *ndef*

其中 *STDHW* 是数字类型。其语义由 ISO/IEC 9541-3 定义。

C3.3.1.12 StdVW(OPTIONAL)

/StdVW[*STDVW*] *ndef*

其中 *STDVW* 是数字类型。其语义由 ISO/IEC 9541-3 定义。

C3.3.1.13 StemSnapH(OPTIONAL)

/StemSnapH[*STEMSNAPHVALUES*] *ndef*

其中 *STEMSNAPHVALUES* 是一个 0 到 12 个数字的序列。其语义由 ISO/IEC 9541-3 定义。

C3.3.1.14 StemSnapV(OPTIONAL)

/StemSnapV[*STEMSNAPVVALUES*] *ndef*

其中 *STEMSNAPVVALUES* 是一个有 0 到 12 个数字的序列。其语义由 ISO/IEC 9541-3 定义。

C3.3.1.15 ForceBold(OPTIONAL)

/ForceBold*FORCEBOLD ndef*

其中 *FORCEBOLD* 是布尔类型，其值必需为 true 或 false。其语义由 ISO/IEC 9541-3 定义。

C3.3.1.16 LanguageGroup(OPTIONAL)

/LanguageGroup*LANGUAGEGROUP ndef*

其中 *LANGUAGEGROUP* 是基数类型。其语义由 ISO/IEC 9541-3 定义。

C3.3.1.17 LenIV(OPTIONAL)

/LenIV*LENIV ndef*

其中 *LENIV* 是基数类型，其语义由 ISO/IEC 9541-3 定义。

C3.3.1.18 MinFeature

/MinFeature (16 16) **def**

C3.3.1.19 Passward

/password 5839 **def**

C3.3.1.20 UniqueID (OPTIONAL)

/UniqueID *UNIQUEID rdef*

其中 *UNIQUEID* 是整数类型，如果选用时，其值应该和 PerFont 属性中的 UnqueID 值相等。

C3.3.1.21 Subrs

subrs 属性由一个带定界符的字符串—加密的 Subrs 序列组成。Subrs 的定义、语义和加密方式由 ISO/IEC 9541-3 定义。subrs 属性开始定界行为：

/Subrs *SUBRSSIZE* **array**

其中 *SUBRSSIZE* 是一个基数，指出 Subrs 属性中 Subrs 的个数。

上述定界符之后是一个有 *SUBRSSIZE* 个元素的序列，其格式为：

dup *INDEX SUBRLENGTH RD ENCRYPTEDSUBR NP*

其中：

——*INDEX* 是类型为基数的唯一索引，其值是在 0 到(*SUBRSSIZE*—1)之间；

——*SUBRLENGTH* 是在字符串—加密的 Subrs 中八位字节的长度；

——*RD* 是作为部分 *RD* 属性定义的八位字节串；

——编码中在 *RD* 值和 *ENCRYPTEDSUBR* 的第一个八位字节之间有一空格字符；

——*ENCRYPTEDSUBR* 是长度为 *SUBRLENGTH* 和八位字节的字符串—加密的 Subr；

——*NP* 是作为部分 *NP* 属性定义的八位字节串。

在字形形状信息段中，由标明专用属性子段结束的 *ND* 来隐含地指出构造 Subrs 属性的 Subr 元素序列的结束。

C3.3.2 CharStrings

字形形状信息段的字符串子段由一个字形名字和加密字符串的映射序列构成。字符串的定义、语义和加密方法在 ISO/IEC 9541-3 中定义。字符串属性的开始定界行为：

2 **index/CharStrings** *CHARSTRINGSSIZE* **dict dup begin**

其中 *CHARSTRINGSSIZE* 是一个基数，指出字符串属性中映射的个数。

在上述定界行之后是一个 *CHARSTRINGSSIZE* 个元素的序列，其格式为：

/GNAME CHARSTRINGLENGTH RD ENCRYPTEDCHARSTRING ND

其中：

——*GNAME* 是字形标识符，用于公用交换时在 ISO/IEC 9541 中定义，用于专用交换时是任意八位字节串；

——*CHARSTRINGLENGTH* 是字符串—加密的字符串中(属性中)八位字节的长度;

——*RD* 是作为部分 *RD* 属性定义的八位字节串;

——编码时,在 *RD* 值和 *ENCRYTEDCHARSTRING* 的第一个八位字节之间有一空格符;

——*ENCRYPTEDCHARSTRING* 是一个字符串—加密的字符串,长度为 *CHARSTRINGLENGTH* 个八位字节;

——*ND* 是作为部分 *ND* 属性定义的八位字节串。

字符串元素序列(也是字形形状信息的字符串子段)的结束定界行为:

end

end

nput

nput

dup/FontName get exch definefont pop

如果字形形状信息段没有用 **eexec** 算法加密,上述定界行也标志字形形状信息段和整个编码的结束。

当且仅当字形形状信息段中使用 **eexec** 算法加密时,下面一行出现在 **eexec** 加密的字形形状信息段的最后八位字节串中。

mark currentfile closefile

同时行

cleartomark

作为纯文本(不加密的)出现在 **eexec** 一加密形状信息段之后,它标识全部编码结束。

C3.3.3 eexec 加密算法

To Be Provided

附 录 D
(标准的附录)
用于交换的 SPDL 指定字模集

所有合格的 SPDL 实现都必须支持一个用于交换的最小标准字模集,这个字模集对 ISO Serif、ISO SanSerif 和 ISO Monospace 三种字模族的每一种都有普通体、粗体、斜体和粗斜体几种字形。这三种字模族必须是分别由以下字形构成的字模:带衬线的印刷间隔字形(typographically-spaced serifed glyphs),不带衬线的印刷间隔字形(typographically-spaced glyphs without serifs),带衬线的单间隔字形(monospaced glyphs with serifs)。本附录对每种字模描述了它所要求的字形补充信息和字号数据(metrics)。实现时可使用上述任一字模,并加上本附录中给出的字的尺度数据。

表 D1 给出了用于交换的 SPDL 标准字模集的标准字形补充信息和宽度信息,对 ISO Serif、ISO Sanserif 和 ISO Monospace 三种字模族的每一种都有普通体、粗体、斜体和粗斜体四种字模。每张表内的项包括:AFII Glyph ID(提供信息的);Glyph Description(取自 AFII Registry/LatinPublishing 1990,1 (Association for Font Information Interchange Glyph Collection for Latin Publishing, Western))字模集;每种印刷字模的字形宽度,这里假设每个字形表示和一个字模资源相关,并且 ISO/IEC 9541 RELUNITS 的值为 1000。在 ISO Monospace 字模族(包括 ISO Monospace Regular, ISO MonospaceBold, ISO MonospaceItalic 和 ISO MonospaceBoldItalic)中的字模也在下表中列出其字形补充信息,每个字形宽度为 600。

表 D1

非标准的 AFII 字形 ID (IR/八进制#)	AFII 字形描述	Serif 系列宽度				San Serif 系列宽度			
		普通体	粗体	斜体	粗斜体	普通体	粗体	斜体	粗斜体
000/040	Space(Normally nonprinting)	250	250	250	250	278	278	278	278
000/041	Exclamation mark	333	333	333	389	278	333	278	333
000/042	Quote,Double,Neutral	408	555	420	555	355	474	355	474
000/043	Number sign	500	500	500	500	556	556	556	556
000/044	General currency symbol	500	500	500	500	556	556	556	556
000/045	Percent sign	833	1000	833	833	889	889	889	889
000/046	Ampersand	778	833	778	778	667	722	667	722
000/050	Parenthesis,beginning(open),curved	333	333	333	333	333	333	333	333
000/051	Parenthesis,close(end),curved	333	333	333	333	333	333	333	333
000/052	Asterisk	500	500	500	500	389	389	389	389
000/053	Plus sign	564	570	675	570	584	584	584	584
000/054	Comma	250	250	250	250	278	278	278	278
000/056	Period	250	250	250	250	278	278	278	278
000/057	Slant	278	278	278	278	278	278	278	278
000/060	Zero	500	500	500	500	556	556	556	556
000/061	One	500	500	500	500	556	556	556	556
000/062	Two	500	500	500	500	556	556	556	556
000/063	Three	500	500	500	500	556	556	556	556
000/064	Four	500	500	500	500	556	556	556	556
000/065	Five	500	500	500	500	556	556	556	556
000/066	Six	500	500	500	500	556	556	556	556
000/067	Seven	500	500	500	500	556	556	556	556
000/070	Eight	500	500	500	500	556	556	556	556
000/071	Nine	500	500	500	500	556	556	556	556
000/072	Colon	278	333	333	333	278	333	278	333
000/073	Semicolon	278	333	333	333	278	333	278	333
000/074	Less than	564	570	675	570	584	584	584	584
000/075	Equals	564	570	675	570	584	584	584	584
000/076	Greater than	564	570	675	570	584	584	584	584
000/077	Question mark	444	500	500	500	556	611	556	611
000/100	At sign(Commercial at sign)	921	930	920	832	1015	975	1015	975
000/101	Uppercase Latin letter A	722	722	611	667	667	722	667	722
000/102	Uppercase Latin letter B	667	667	611	667	667	722	667	722
000/103	Uppercase Latin letter C	667	722	667	667	722	722	722	722
000/104	Uppercase Latin letter D	722	722	722	722	722	722	722	722
000/105	Uppercase Latin letter E	611	667	611	667	667	667	667	667
000/106	Uppercase Latin letter F	556	611	611	667	611	611	611	611
000/107	Uppercase Latin letter G	722	778	722	722	778	778	778	778
000/110	Uppercase Latin letter H	722	778	722	778	722	722	722	722
000/111	Uppercase Latin letter I	333	389	333	389	278	278	278	278
000/112	Uppercase Latin letter J	389	500	444	500	500	556	500	556
000/113	Uppercase Latin letter K	722	778	667	667	667	722	667	722
000/114	Uppercase Latin letter L	611	667	556	611	556	611	556	611
000/115	Uppercase Latin letter M	889	944	833	889	833	833	833	833

表 D1(续)

非标准的 AFII 字形 ID (IR/八进制#)	AFII 字形描述	Serif 系列宽度				San Serif 系列宽度			
		普通体	粗体	斜体	粗斜体	普通体	粗体	斜体	粗斜体
000/116	Uppercase Latin letter N	722	722	667	722	722	722	722	722
000/117	Uppercase Latin letter O	722	778	722	722	778	778	778	778
000/120	Uppercase Latin letter P	556	611	611	611	667	667	667	667
000/121	Uppercase Latin letter Q	722	778	722	722	778	778	778	778
000/122	Uppercase Latin letter R	667	722	611	667	722	722	722	722
000/123	Uppercase Latin letter S	556	556	500	556	667	667	667	667
000/124	Uppercase Latin letter T	611	667	556	611	611	611	611	611
000/125	Uppercase Latin letter U	722	722	722	722	722	722	722	722
000/126	Uppercase Latin letter V	722	722	611	667	667	667	667	667
000/127	Uppercase Latin letter W	944	1000	833	889	944	944	944	944
000/130	Uppercase Latin letter X	722	722	611	667	667	667	667	667
000/131	Uppercase Latin letter Y	722	722	556	611	667	667	667	667
000/132	Uppercase Latin letter Z	611	667	556	611	611	611	611	611
000/133	Bracket,Opening(Left)	333	333	389	333	278	333	278	333
000/134	Slant,Reverse	278	278	278	278	278	278	278	278
000/135	Bracket,Closing(Right)	333	333	389	333	278	333	278	333
000/136	Circumflex(l. c. spacing accent)	333	333	333	333	333	333	333	333
000/137	Bar,Low	500	500	500	500	556	556	556	556
000/140	Grave(l. c. spacing accent)	333	333	333	333	333	333	333	333
000/141	Lowercase Latin letter a	444	500	500	500	556	556	556	556
000/142	Lowercase Latin letter b	500	556	500	500	556	611	556	611
000/143	Lowercase Latin letter c	444	444	444	444	500	556	500	556
000/144	Lowercase Latin letter d	500	556	500	500	556	611	556	611
000/145	Lowercase Latin letter e	444	444	444	444	556	556	556	556
000/146	Lowercase Latin letter f	333	333	278	333	278	333	278	333
000/147	Lowercase Latin letter g	500	500	500	500	556	611	556	611
000/150	Lowercase Latin letter h	500	556	500	556	556	611	556	611
000/151	Lowercase Latin letter i	278	278	278	278	222	278	222	278
000/152	Lowercase Latin letter j	278	333	278	278	222	278	222	278
000/153	Lowercase Latin letter k	500	556	444	500	500	556	500	556
000/154	Lowercase Latin letter l	278	278	278	278	222	278	222	278
000/155	Lowercase Latin letter m	778	833	722	778	833	889	833	889
000/156	Lowercase Latin letter n	500	556	500	556	556	611	556	611
000/157	Lowercase Latin letter o	500	500	500	500	556	611	556	611
000/160	Lowercase Latin letter p	500	556	500	500	556	611	556	611
000/161	Lowercase Latin letter q	500	556	500	500	556	611	556	611
000/162	Lowercase Latin letter r	333	444	389	389	333	389	333	389
000/163	Lowercase Latin letter s	389	389	389	389	500	556	500	556
000/164	Lowercase Latin letter t	278	333	278	278	278	333	278	333
000/165	Lowercase Latin letter u	500	556	500	556	556	611	556	611
000/166	Lowercase Latin letter v	500	500	444	444	500	556	500	556
000/167	Lowercase Latin letter w	722	722	667	667	722	778	722	778
000/170	Lowercase Latin letter x	500	500	444	500	500	556	500	556
000/171	Lowercase Latin letter y	500	500	444	444	500	556	500	556

表 D1(续)

非标准的 AFII 字形 ID (IR/八进制#)	AFII 字形描述	Serif 系列宽度				San Serif 系列宽度			
		普通体	粗体	斜体	粗斜体	普通体	粗体	斜体	粗斜体
000/172	Lowercase Latin letter z	444	444	389	389	500	500	500	500
000/173	Brace,beginning(open)	480	394	400	348	334	389	334	389
000/174	Bar,vertical,single	200	220	275	220	260	280	260	280
000/175	Brace,closing(end)	480	394	400	348	334	389	334	389
000/176	Tilde(l. c. spacing accent)	333	333	333	333	333	333	333	333
000/241	Exclamation mark(Spanish) Inverted	333	333	389	389	333	333	333	333
000/242	Cent sign	500	500	500	500	556	556	556	556
000/243	Pound-Sterling sign	500	500	500	500	556	556	556	556
000/244	Dollar sign	500	500	500	500	556	556	556	556
000/245	Yen currency symbol(Japanese)	500	500	500	500	556	556	556	556
000/247	Section sign	500	500	500	500	556	556	556	556
000/251	Quotation mark,beginning,single,curved	333	333	333	333	222	278	222	278
000/252	Quotation mark,beginning,double,curved	444	500	556	500	333	500	333	500
000/253	Guillemet(European quotation),left,double	500	500	500	500	556	556	556	556
000/260	Degree sign	400	400	400	400	400	400	400	400
000/261	Plus or minus sign	564	570	675	570	584	584	584	584
000/262	Squared	300	300	300	300	333	333	333	333
000/263	Cubed	300	300	300	300	333	333	333	333
000/264	Multiply sign	564	570	675	570	584	584	584	584
000/265	Micro sign	500	556	500	576	556	611	556	611
000/266	Paragraph sign	453	540	523	500	537	556	537	556
000/267	Dot,Centered	250	250	250	250	278	278	278	278
000/270	Divide sign	564	570	675	570	584	584	584	584
000/271	Quotation mark,closing,single,curved	333	333	333	333	222	278	222	278
000/272	Quotation mark,closing,double,curved	444	500	556	500	333	500	333	500
000/273	Guillemet(European quotation),right,double	500	500	500	500	556	556	556	556
000/274	Fraction,one quarter,en set	750	750	750	750	834	834	834	834
000/275	Fraction,one half,en set	750	750	750	750	834	834	834	834
000/276	Fraction,three,quarters,en set	750	750	750	750	834	834	834	834
000/277	Question mark,Spanish,inverted	444	500	500	500	611	611	611	611
000/321	Reference 1	300	300	300	300	333	333	333	333
000/322	Registered sign	760	747	760	747	737	737	737	737
000/323	Copyright sign	760	747	760	747	737	737	737	737
000/324	Trademark sign	980	1000	980	1000	1000	1000	1000	1000
000/341	AE digraph,Uppercase	889	1000	889	944	1000	1000	1000	1000
000/342	D with stroke,Uppercase(Croatian)	722	722	722	722	722	722	722	722
000/343	Ordinal Indicator,Feminine(Spanish)	276	300	276	266	370	370	370	370
000/350	L with stroke,Uppercase(Polish)	611	667	556	611	556	611	556	611
000/351	O with slash,Uppercase(Norwegian,Danish)	722	778	722	722	778	778	778	778
000/352	OE digraph,Uppercase	889	1000	944	944	1000	1000	1000	1000
000/353	Ordinal Indicator,Masculine(Spanish)	310	330	310	300	365	365	365	365
000/354	Thorn,Uppercase(Icelandic)	556	611	611	611	667	667	667	667
000/361	ae digraph,Lowercase	667	722	667	722	889	889	889	889
000/363	Eth,Lowercase(Icelandic)	500	500	500	500	556	611	556	611

表 D1(续)

非标准的 AFII 字形 ID (IR/八进制#)	AFII 字形描述	Serif 系列宽度				San Serif 系列宽度			
		普通体	粗体	斜体	粗斜体	普通体	粗体	斜体	粗斜体
000/365	i,dotless,Lowercase(Turkish)	278	278	278	278	278	278	278	278
000/370	I with stroke,Lowercase(Polish)	278	278	278	278	222	278	222	278
000/371	o with slash,Lowercase(Norwegian,Danish)	500	500	500	500	611	611	611	611
000/372	oe digraph,Lowercase	722	722	667	722	944	944	944	944
000/373	s,Double	500	556	500	500	611	611	611	611
000/374	Thorn,Lowercase(Icelandic)	500	556	500	500	556	611	556	611
041/076	Hyphen	333	333	333	333	333	333	333	333
041/104	Leader,three-dot on an em body	1000	1000	889	1000	1000	1000	1000	1000
042/300	Up arrowhead	469	581	422	570	469	584	469	584
043/042	Diaeresis(l. c. spacing accent)	333	333	333	333	333	333	333	333
043/043	Acute(l. c. spacing accent)	333	333	333	333	333	333	333	333
043/044	Macron	333	333	333	333	333	333	333	333
043/045	Breve(l. c. spacing accent)	333	333	333	333	333	333	333	333
043/046	Over-dot(l. c. spacing accent)	333	333	333	333	333	333	333	333
043/050	Over-ring(l. c. spacing accent)	333	333	333	333	333	333	333	333
043/051	Double acute(l. c. spacing accent)	333	333	333	333	333	333	333	333
043/052	Ogonek(l. c. spacing undermark)	333	333	333	333	333	333	333	333
043/053	Hachek(l. c. spacing accent)	333	333	333	333	333	333	333	333
043/054	Cedilla(l. c. spacing undermark)	333	333	333	333	333	333	333	333
043/262	Quote,Single,Left,Lowered	333	333	333	333	222	278	222	278
356/055	Minus sign	564	570	675	606	584	584	584	584
356/176	Similar to,Type 1	541	520	541	570	584	584	584	584
356/044	Dash,En	500	500	500	500	556	556	556	556
357/045	Dash,Em	1000	1000	889	1000	1000	1000	1000	1000
357/047	Quote,Single,Neutral	180	278	214	278	191	238	191	238
357/050	Quote,Double,Left,Lowered	444	500	556	500	333	500	333	500
357/052	Guillemet,Single,Left quote	333	333	333	333	333	333	333	333
357/053	Guillemet,Single,Right quote	333	333	333	333	333	333	333	333
357/060	Dagger	500	500	500	500	556	556	556	556
357/061	Dagger,double	500	500	500	500	556	556	556	556
357/101	Per thousand	1000	1000	1000	1000	1000	1000	1000	1000
357/146	Bullet,centered	350	350	350	350	350	350	350	350
357/152	End of Line Symbol;Also;Not	564	570	675	606	584	584	584	584
357/153	Bar,Vertical,Broken	200	220	275	220	260	280	260	280
357/242	Florin	500	500	500	500	556	556	556	556
360/044	Ligature fi	556	556	500	556	500	611	500	611
360/045	Ligature fl	556	556	500	556	500	611	500	611
361/041	Grave A	722	722	611	667	667	722	667	722
361/042	Acute A	722	722	611	667	667	722	667	722
361/043	Circumflex A	722	722	611	667	667	722	667	722
361/044	Tilde A	722	722	611	667	667	722	667	722
361/047	Diaeresis A	722	722	611	667	667	722	667	722
361/050	Ring A	722	722	611	667	667	722	667	722
361/055	Cedilla C	667	722	667	667	722	722	722	722

表 D1(续)

非标准的 AFII 字形 ID (IR/八进制#)	AFII 字形描述	Serif 系列宽度				San Serif 系列宽度			
		普通体	粗体	斜体	粗斜体	普通体	粗体	斜体	粗斜体
361/060	Grave E	611	667	611	667	667	667	667	667
361/061	Acute E	611	667	611	667	667	667	667	667
361/062	Circumflex E	611	667	611	667	667	667	667	667
361/065	Diaeresis E	611	667	611	667	667	667	667	667
361/076	Grave 1	333	389	333	389	278	278	278	278
361/077	Acute 1	333	389	333	389	278	278	278	278
361/100	Circumflex 1	333	389	333	389	278	278	278	278
361/104	Diaeresis 1	333	389	333	389	278	278	278	278
361/114	Tilde N	722	722	667	722	722	722	722	722
361/117	Grave O	722	778	722	722	778	778	778	778
361/120	Acute O	722	778	722	722	778	778	778	778
361/121	Circumflex O	722	778	722	722	778	778	778	778
361/122	Tilde O	722	778	722	722	778	778	778	778
361/124	Diaeresis O	722	778	722	722	778	778	778	778
361/134	Hachek S	556	556	500	556	667	667	667	667
361/137	Grave U	722	722	722	722	722	722	722	722
361/140	Acute U	722	722	722	722	722	722	722	722
361/141	Circumflex U	722	722	722	722	722	722	722	722
361/145	Diaeresis U	722	722	722	722	722	722	722	722
361/153	Acute Y	722	722	556	611	667	667	667	667
361/155	Diaeresis Y	722	722	556	611	667	667	667	667
361/160	Hachek Z	611	667	556	611	611	611	611	611
361/241	Grave a	444	500	500	500	556	556	556	556
361/242	Acute a	444	500	500	500	556	556	556	556
361/243	Circumflex a	444	500	500	500	556	556	556	556
361/244	Tilde a	444	500	500	500	556	556	556	556
361/247	Diaeresis a	444	500	500	500	556	556	556	556
361/250	Ring a	444	500	500	500	556	556	556	556
361/255	Cedilla c	444	444	444	444	500	556	500	556
361/260	Grave e	444	444	444	444	556	556	556	556
361/261	Acute e	444	444	444	444	556	556	556	556
361/262	Circumflex e	444	444	444	444	556	556	556	556
361/265	Diaeresis e	444	444	444	444	556	556	556	556
361/276	Grave i	278	278	278	278	278	278	278	278
361/277	Acute i	278	278	278	278	278	278	278	278
361/300	Circumflex i	278	278	278	278	278	278	278	278
361/304	Diaeresis i	278	278	278	278	278	278	278	278
361/314	Tilde n	500	556	500	556	556	611	556	611
361/317	Grave o	500	500	500	500	556	611	556	611
361/320	Acute o	500	500	500	500	556	611	556	611
361/321	Circumflex o	500	500	500	500	556	611	556	611
361/322	Tilde o	500	500	500	500	556	611	556	611
361/324	Diaeresis o	500	500	500	500	556	611	556	611
361/334	Hachek s	389	389	389	389	500	556	500	556

表 D1(完)

非标准的 AFII 字形 ID (IR/八进制#)	AFII 字形描述	Serif 系列宽度				San Serif 系列宽度			
		普通体	粗体	斜体	粗斜体	普通体	粗体	斜体	粗斜体
361/337	Grave u	500	556	500	556	556	611	556	611
361/340	Acute u	500	556	500	556	556	611	556	611
361/341	Circumflex u	500	556	500	556	556	611	556	611
361/345	Diaeresis u	500	556	500	556	556	611	556	611
361/353	Acute y	500	500	444	444	500	556	500	556
361/355	Diaeresis y	500	500	444	444	500	556	500	556
361/360	Hachek z	444	444	389	389	500	500	500	500
375/257	Fraction bar	167	167	167	167	167	167	167	167

附 录 E
(标准的附录)
字母索引映射表

本附录定义了两种标准的字母索引映射——指定标准的 GlyphIndexMap/Latin1Publishing 和标准的 GlyphIndexMap/Latin1PublishingA。

表 E1 字形索引表/拉丁版

索引	字形 ID	AFII 描述(非标准的)	索引	字形 ID	AFII 描述(非标准的)
0	.notdef	unused	24	.notdef	unused
1	.notdef	unused	25	.notdef	unused
2	.notdef	unused	26	.notdef	unused
3	.notdef	unused	27	.notdef	unused
4	.notdef	unused	28	.notdef	unused
5	.notdef	unused	29	.notdef	unused
6	.notdef	unused	30	.notdef	unused
7	.notdef	unused	31	.notdef	unused
8	.notdef	unused	32	afii:20	SPACE
9	.notdef	unused	33	afii:21	EXCLAMATION MARK
10	.notdef	unused	34	afii:22	QUOTATION MARK
11	.notdef	unused	35	afii:23	NUMBER SIGN
12	.notdef	unused	36	afii:24	DOLLAR SIGN
13	.notdef	unused	37	afii:25	PERCENT SIGN
14	.notdef	unused	38	afii:26	AMPERSAND
15	.notdef	unused	39	afii:27	APOSTROPHE
16	.notdef	unused	40	afii:28	LEFT PARENTHESIS
17	.notdef	unused	41	afii:29	RIGHT PARENTHESIS
18	.notdef	unused	42	afii:2a	ASTERISK
19	.notdef	unused	43	afii:2b	PLUS SIGN
20	.notdef	unused	44	afii:2c	COMMA
21	.notdef	unused	45	afii:2d	HYPHEN,MINUS SIGN
22	.notdef	unused	46	afii:2e	FULL STOP
23	.notdef	unused	47	afii:2f	SOLIDUS

表 E1(续)

索引	字形 ID	AFII 描述(非标准的)	索引	字形 ID	AFII 描述(非标准的)
48	afii:30	DIGIT ZERO	95	afii:5f	LOW LINE
49	afii:31	DIGIT ONE	96	afii:60	GRAVE ACCENT
50	afii:32	DIGIT TWO	97	afii:61	SMALL LETTER a
51	afii:33	DIGIT THREE	98	afii:62	SMALL LETTER b
52	afii:34	DIGIT FOUR	99	afii:63	SMALL LETTER c
53	afii:35	DIGIT FIVE	100	afii:64	SMALL LETTER d
54	afii:36	DIGIT SIX	101	afii:65	SMALL LETTER e
55	afii:37	DIGIT SEVEN	102	afii:66	SMALL LETTER f
56	afii:38	DIGIT EIGHT	103	afii:67	SMALL LETTER g
57	afii:39	DIGIT NINE	104	afii:68	SMALL LETTER h
58	afii:3a	COLON	105	afii:69	SMALL LETTER i
59	afii:3b	SEMICOLON	106	afii:6a	SMALL LETTER j
60	afii:3c	LESS-THAN SIGN	107	afii:6b	SMALL LETTER k
61	afii:3d	EQUALS SIGN	108	afii:6c	SMALL LETTER l
62	afii:3e	GREATER—THAN SIGN	109	afii:6d	SMALL LETTER m
63	afii:3f	QUESTION MARK	110	afii:6e	SMALL LETTER n
64	afii:40	COMMERCIAL AT	111	afii:6f	SMALL LETTER o
65	afii:41	CAPITAL LETTER A	112	afii:70	SMALL LETTER p
66	afii:42	CAPITAL LETTER B	113	afii:71	SMALL LETTER q
67	afii:43	CAPITAL LETTER C	114	afii:72	SMALL LETTER r
68	afii:44	CAPITAL LETTER D	115	afii:73	SMALL LETTER s
69	afii:45	CAPITAL LETTER E	116	afii:74	SMALL LETTER t
70	afii:46	CAPITAL LETTER F	117	afii:75	SMALL LETTER u
71	afii:47	CAPITAL LETTER G	118	afii:76	SMALL LETTER v
72	afii:48	CAPITAL LETTER H	119	afii:77	SMALL LETTER w
73	afii:49	CAPITAL LETTER I	120	afii:78	SMALL LETTER x
74	afii:4a	CAPITAL LETTER J	121	afii:79	SMALL LETTER y
75	afii:4b	CAPITAL LETTER K	122	afii:7a	SMALL LETTER z
76	afii:4c	CAPITAL LETTER L	123	afii:7b	LEFT CURLY BRACE
77	afii:4d	CAPITAL LETTER M	124	afii:7c	VERTICAL LINE
78	afii:4e	CAPITAL LETTER N	125	afii:7d	RIGHT CURLY BRACE
79	afii:4f	CAPITAL LETTER O	126	afii:7e	TILDE
80	afii:50	CAPITAL LETTER P	127	.notdef	unused
81	afii:51	CAPITAL LETTER Q	128	.notdef	unused
82	afii:52	CAPITAL LETTER R	129	.notdef	unused
83	afii:53	CAPITAL LETTER S	130	.notdef	unused
84	afii:54	CAPITAL LETTER T	131	.notdef	unused
85	afii:55	CAPITAL LETTER U	132	.notdef	unused
86	afii:56	CAPITAL LETTER V	133	.notdef	unused
87	afii:57	CAPITAL LETTER W	134	.notdef	unused
88	afii:58	CAPITAL LETTER X	135	.notdef	unused
89	afii:59	CAPITAL LETTER Y	136	.notdef	unused
90	afii:5a	CAPITAL LETTER Z	137	.notdef	unused
91	afii:5b	LEFT SQUARE BRACKET	138	.notdef	unused
92	afii:5c	REVERSE SOLIDUS	139	.notdef	unused
93	afii:5d	RIGHT SQUARE BRACKET	140	.notdef	unused
94	afii:5e	CIRCUMFLEX ACCENT	141	.notdef	unused

表 E1(续)

索引	字形 ID	AFII 描述(非标准的)	索引	字形 ID	AFII 描述(非标准的)
142	.notdef	unused	184	afii:462c	CEDILLA
143	.notdef	unused	185	afii:d1	SUPERSCRIPT ONE
144	.notdef	unused	186	afii:eb	MASCULINE ORDINAL INDICATOR
145	.notdef	unused			
146	.notdef	unused	187	afii:bb	RIGHT ANGLE QUOTATION MARK
147	.notdef	unused			
148	.notdef	unused	188	afii:bc	VULGAR FRACTION ONE QUARTER
149	.notdef	unused			
150	.notdef	unused	189	afii:bd	VULGAR FRACTION ONE HALF
151	.notdef	unused	190	afii:be	VULGAR FRACTION THREE QUARTERS
152	.notdef	unused			
153	.notdef	unused	191	afii:bf	INVERTED QUESTION MARK
154	.notdef	unused	192	afii:1e221	CAPITAL LETTER A WITH GRAVE ACCENT
155	.notdef	unused			
156	.notdef	unused	193	afii:1e222	CAPITAL LETTER A WITH ACUTE ACCENT
157	.notdef	unused			
158	.notdef	unused	194	afii:1e223	CAPITAL LETTER A WITH CIRCUMFLEX ACCENT
159	.notdef	unused			
160	afii:1de21	NO-BREAK SPACE	195	afii:1e224	CAPITAL LETTER A WITH TILDE
161	afii:a1	INVERTED EXCLAMATION MARK	196	afii:1e227	CAPITAL LETTER A WITH DIAERESIS
162	afii:a2	CENT SIGN			
163	afii:a3	POUND SIGN	197	afii:1e228	CAPITAL LETTER A WITH RING ABOVE
164	afii:24	CURRENCY SIGN			
165	afii:a5	YEN SIGN	198	afii:e1	CAPITAL DIPTHONG A WITHE
166	afii:1de6b	BROKEN BAR	199	afii:1e22d	CAPITAL LETTER C WITH CEDILLA
167	afii:a7	PARAGRAPH SIGN, SECTION SIGN	200	afii:1e230	CAPITAL LETTER E WITH GRAVE ACCENT
168	afii:4622	DIAERESIS			
169	afii:d3	COPYRIGHT SIGN	201	afii:1e231	CAPITAL LETTER E WITH ACUTE ACCENT
170	afii:e3	FEMININE ORDINAL INDICATOR			
171	afii:ab	LEFT ANGLE QUOTATION MARK	202	afii:1e232	CAPITAL LETTER E WITH CIRCUMFLEX ACCENT
172	afii:1de6a	NOT SIGN			
173	afii:1de23	SOFT HYPHEN	203	afii:1e235	CAPITAL LETTER E WITH DIAERESIS
174	afii:d2	REGISTERED TRADE MARK SIGN	204	afii:1e23e	CAPITAL LETTER I WITH GRAVE ACCENT
175	afii:4624	MACRON			
176	afii:4628	RING ABOVE,DEGREE SIGN	205	afii:1e23f	CAPITAL LETTER I WITH ACUTE ACCENT
177	afii:b1	PLUS-MINUS SIGN			
178	afii:b2	SUPERSCRIPT TWO	206	afii:1e240	CAPITAL LETTER I WITH CIRCUMFLEX ACCENT
179	afii:b3	SUPERSCRIPT THREE			
180	afii:4623	ACUTE ACCENT	207	afii:1e244	CAPITAL LETTER I WITH DIAERESIS
181	afii:b5	MICRO SIGN			
182	afii:b6	PILCROW SIGN	208	afii:e2	CAPITAL ICELANDIC LETTER ETH
183	afii:b7	MIDDLE DOT			

表 E1(完)

索引	字形 ID	AFII 描述(非标准的)	索引	字形 ID	AFII 描述(非标准的)
209	afii:1e24c	CAPITAL LETTER N WITH TILDE	233	afii:1e2b1	SMALL LETTER e WITH ACUTE ACCENT
210	afii:1e24f	CAPITAL LETTER O WITH GRAVE ACCENT	234	afii:1e2b2	SMALL LETTER e WITH CIRCUMFLEX ACCENT
211	afii:1e250	CAPITAL LETTER O WITH ACUTE ACCENT	235	afii:1e2b5	SMALL LETTER e WITH DIAERESIS
212	afii:1e251	CAPITAL LETTER O WITH CIRCUMFLEX ACCENT	236	afii:1e2be	SMALL LETTER i WITH GRAVE ACCENT
213	afii:1e252	CAPITAL LETTER O WITH TILDE	237	afii:1e2bf	SMALL LETTER i WITH ACUTE ACCENT
214	afii:1e254	CAPITAL LETTER O WITH DIAERESIS	238	afii:1e2c0	SMALL LETTER i WITH CIRCUMFLEX ACCENT
215	afii:b4	MULTIPLI CATION SIGN	239	afii:1e2c4	SMALL LETTER i WITH DIAERESIS
216	afii:e9	CAPITAL LETTER O WITH OBLIQUE STROKE	240	afii:f3	SMALL ICELANDIC LETTER ETH
217	afii:1e25f	CAPITAL LETTER U WITH GRAVE ACCENT	241	afii:1e2cc	SMALL LETTER n WITH TILDE
218	afii:1e260	CAPITAL LETTER U WITH ACUTE ACCENT	242	afii:1e2cf	SMALL LETTER o WITH GRAVE ACCENT
219	afii:1e261	CAPITAL LETTER U WITH CIRCUMFLEX ACCENT	243	afii:1e2d0	SMALL LETTER o WITH ACUTE ACCENT
220	afii:1e265	CAPITAL LETTER U WITH DIAERESIS	244	afii:1e2d1	SMALL LETTER o WITH CIRCUMFLEX ACCENT
221	afii:1e26b	CAPITAL LETTER Y WITH ACUTE ACCENT	245	afii:1e2d2	SMALL LETTER o WITH TILDE
222	afii:ec	CAPITAL ICELANDIC LETTER THORN	246	afii:1e2d4	SMALL LETTER o WITH DIAERESIS
223	afii:fb	SMALL GERMAN LETTER SHARPS	247	afii:b8	DIVISION SIGN
224	afii:1e2a1	SMALL LETTER a WITH GRAVE ACCENT	248	afii:f9	SMALL LETTER o WITH OBLIQUE STROKE
225	afii:1e2a2	SMALL LETTER a WITH ACUTE ACCENT	249	afii:1e2df	SMALL LETTER u WITH GRAVE ACCENT
226	afii:1e2a3	SMALL LETTER a WITH CIRCUMFLEX ACCENT	250	afii:1e2e0	SMALL LETTER u WITH ACUTE ACCENT
227	afii:1e2a4	SMALL LETTER a WITH TILDE	251	afii:1e2e1	SMALL LETTER u WITH CIRCUMFLEX ACCENT
228	afii:1e2a7	SMALL LETTER a WITH DIAERESIS	252	afii:1e2e5	SMALL LETTER u WITH DIAERESIS
229	afii:1e2a8	SMALL LETTER a WITH RING ABOVE	253	afii:1e2eb	SMALL LETTER y WITH ACUTE ACCENT
230	afii:f1	SMALL DIPTHONG a WITH e	254	afii:fc	SMALL ICELANDIC LETTER THORN
231	afii:1e2ad	SMALL LETTER c WITH CEDILLA	255	afii:1e2ed	SMALL LETTER y WITH DIAERESIS
232	afii:1e2b0	SMALL LETTER e WITH GRAVE ACCENT			

表 E2　字形索引表/拉丁版 A

索引	字形 ID	索引	字形 ID	索引	字形 ID	索引	字形 ID
0	.notdef	47	slash	94	asciicircum	141	.notdef
1	.notdef	48	zero	95	underscore	142	.notdef
2	.notdef	49	one	96	quoteleft	143	.notdef
3	.notdef	50	two	97	a	144	dotlessi
4	.notdef	51	three	98	b	145	grave
5	.notdef	52	four	99	c	146	acute
6	.notdef	53	five	100	d	147	circumflex
7	.notdef	54	six	101	e	148	tilde
8	.notdef	55	seven	102	f	149	macron
9	.notdef	56	eight	103	g	150	breve
10	.notdef	57	nine	104	h	151	dotaccent
11	.notdef	58	colon	105	i	152	dieresis
12	.notdef	59	semicolon	106	j	153	.notdef
13	.notdef	60	less	107	k	154	ring
14	.notdef	61	equal	108	l	155	cedilla
15	.notdef	62	greater	109	m	156	.notdef
16	.notdef	63	question	110	n	157	hungarumlaut
17	.notdef	64	at	111	o	158	ogonek
18	.notdef	65	A	112	p	159	caron
19	.notdef	66	B	113	q	160	space
20	.notdef	67	C	114	r	161	exclamdown
21	.notdef	68	D	115	s	162	cent
22	.notdef	69	E	116	t	163	sterling
23	.notdef	70	F	117	u	164	currency
24	.notdef	71	G	118	v	165	yen
25	.notdef	72	H	119	w	166	brokenbar
26	.notdef	73	I	120	x	167	section
27	.notdef	74	J	121	y	168	dieresis
28	.notdef	75	K	122	z	169	copyright
29	.notdef	76	L	123	braceleft	170	ordfeminine
30	.notdef	77	M	124	bar	171	guillemotleft
31	.notdef	78	N	125	braceright	172	logicalnot
32	space	79	O	126	asciitilde	173	hyphen
33	exclam	80	P	127	.notdef	174	registered
34	quotedbl	81	Q	128	.notdef	175	macron
35	numbersign	82	R	129	.notdef	176	degree
36	dollar	83	S	130	.notdef	177	plusminus
37	percent	84	T	131	.notdef	178	twosuperior
38	ampersand	85	U	132	.notdef	179	threesuperior
39	quoteright	86	V	133	.notdef	180	acute
40	parenleft	87	W	134	.notdef	181	mu
41	parenright	88	X	135	.notdef	182	paragraph
42	asterisk	89	Y	136	.notdef	183	periodcentered
43	plus	90	Z	137	.notdef	184	cedilla
44	comma	91	bracketleft	138	.notdef	185	onesuperior
45	minus	92	backslash	139	.notdef	186	ordmasculine
46	period	93	bracketright	140	.notdef	187	guillemotright

表 E2(完)

索引	字形 ID	索引	字形 ID	索引	字形 ID	索引	字形 ID
188	onequarter	205	lacute	222	Thorn	239	idieresis
189	onehalf	206	lcircumflex	223	germandbls	240	eth
190	threequarters	207	ldieresis	224	agrave	241	ntilde
191	questiondown	208	Eth	225	aacute	242	ograve
192	Agrave	209	Ntilde	226	acircumflex	243	oacute
193	Aacute	210	Ograve	227	atilde	244	ocircumflex
194	Acircumflex	211	Oacute	228	adieresis	245	otilde
195	Atilde	212	Ocircumflex	229	aring	246	odieresis
196	Adieresis	213	Otilde	230	ae	247	divide
197	Aring	214	Odieresis	231	ccedilla	248	oslash
198	AE	215	multiply	232	egrave	249	ugrave
199	Ccedilla	216	Oslash	233	eacute	250	uacute
200	Egrave	217	Ugrave	234	ecircumflex	251	ucircumflex
201	Eacute	218	Uacute	235	edieresis	252	udieresis
202	Ecircumflex	219	Ucircumflex	236	igrave	253	yacute
203	Edieresis	220	Udieresis	237	iacute	254	thorn
204	lgrave	221	Yacute	238	icircumflex	255	ydieresis

中 华 人 民 共 和 国
国 家 标 准
信息技术 文本通信
标准页面描述语言(SPDL)
GB/T 16648—1996
*
中国标准出版社出版
北京复兴门外三里河北街16号
邮政编码:100045
电 话:68522112
中国标准出版社秦皇岛印刷厂印刷
新华书店北京发行所发行 各地新华书店经售

*
开本 880×1230 1/16 印张 14¾ 字数 465 千字
1997 年 11 月第一版 1997 年 11 月第一次印刷
印数 1—1 000
*
书号:155066·1-14268 定价 70.00 元
*
标 目 323—029