



Scintilla 文档

最后编辑 2018 年 6 月 6 日 NH

有 [Scintilla 的内部设计的概述](#)。

关于使用 [Scintilla 的一些注意事项](#)。

如何在 [Windows 上使用 Scintilla Edit Control](#)。

在 [Windows 上使用 C++ 中的 Scintilla 的简单示例](#)。

使用 [Visual Basic 中的 Scintilla 的简单示例](#)。

诱饵是在 [GTK+ 上使用 Scintilla 的一个小样本](#)。

有关如何编写词法分析器的详细说明，包括[折叠的讨论](#)。

如何在容器中实现词法分析器。

如何实现[折叠](#)。

初学者指南 [lexing](#) 和 [折叠](#)。

该[编码风格](#) 如果你想为 Scintilla 贡献代码但不是强制性的，那么在 Scintilla 和 SciTE 中使用是值得关注的。

介绍

Windows 版 Scintilla 是一个 Windows 控件。因此，它的主要编程接口是通过 Windows 消息。早期版本的 Scintilla 模拟了标准 Windows Edit 和 RichEdit 控件定义的大部分 API，但现在不推荐使用这些 API，而是赞成 Scintilla 自己的，更一致的 API。除了执行普通 Edit 控件操作的消息之外，Scintilla 还允许控制语法样式，折叠，标记，自动完成和调用提示。

GTK+ 版本也以与 Windows 版本类似的方式使用消息。这与普通的 GTK+ 实践不同，但更容易快速实施。

Scintilla 还在 OS X 和 Qt 上使用 Cocoa 构建，并遵循这些平台的惯例。

Scintilla 在 Windows 上仅提供有限的实验支持，用于从右到左的语言，如阿拉伯语和希伯来语。虽然这些语言中的文本可能看起来是正确的，但与此文本的交互可能无法像其他编辑器那样正常工作。

本文档描述了 Scintilla 使用的各个消息和通知。它没有描述如何将它们链接在一起以形成有用的编辑器。目前，研究如何使用 Scintilla 进行开发的最佳方法是了解 SciTE 如何使用它。SciTE 锻炼了 Scintilla 的大部分设施。

在下面的描述中，消息被描述为具有零个，一个或两个参数的函数调用。这两个参数是 Windows 程序员的标准 `WPARAM` 和 `LPARAM` 熟悉程度。这些参数是足以

容纳指针的整数，返回值也是一个足以包含指针的整数。尽管命令仅使用所描述的参数，因为所有消息都有两个参数，无论 **Scintilla** 是否使用它们，强烈建议将任何未使用的参数设置为 0。这样可以在将来增强消息，而不会有破坏现有代码的风险。常见的参数类型是：

bool	参数期望值为 0 false 和 1 为 true 。
int	参数是 32 位或 64 位有符号整数，具体取决于平台。相当于 intptr_t 。
const char *	参数指向传递给 Scintilla 但未修改的文本。文本可以是零终止或另一个参数可以指定字符计数，描述将使这一点清楚。
char *	参数指向 Scintilla 将填充文本的文本缓冲区。在某些情况下，另一个参数将告诉 Scintilla 缓冲区大小。在其他情况下，您必须确保缓冲区足够大以容纳所请求的文本。如果传递 NULL 指针（0），则对于 SCI_* 调用，返回分配的长度，不包括任何终止 NUL 。一些调用（标记为“ NUL-terminated ”）在结果中添加 NUL 字符，但其他调用不会：通常处理两种类型，分配比指示多一个字节并将其设置为 NUL 。
colour	使用 RGB 格式（红色，绿色，蓝色）设置颜色。每种颜色的强度设置在 0 到 255 范围内。如果你有三种这样的强度，它们组合为：红色 （绿色<< 8） （蓝色<< 16）。如果将所有强度设置为 255，则颜色为白色。如果将所有强度设置为 0，则颜色为黑色。设置颜色时，您正在发出请求。您将获得什么取决于系统的功能和当前的屏幕模式。
alpha	使用 alpha 值设置半透明度。 Alpha 范围从 0（ SC_ALPHA_TRANSPARENT ）到完全透明到 255（ SC_ALPHA_OPAQUE ）是不透明的。值 256（ SC_ALPHA_NOALPHA ）是不透明的，并且使用不支持 alpha 的代码并且可能更快。并非所有平台都支持半透明，只有一些 Scintilla 功能实现半透明。大多数功能的默认 Alpha 值是 SC_ALPHA_NOALPHA 。
<unused>	这是一个未使用的参数。将其设置为 0 将确保与未来增强功能兼容。

内容

滚动

- 文本检索和修改
- 剪切，复制和粘贴
- 选择和信息
- 滚动和自动
- 搜索和替换
- 错误处理
- 按字符或 UTF-16 代码单位
- 白色空间
- Overtyping
- 撤消和重做
- 多选和虚拟空间
- 光标

- 鼠标捕获
- 造型
- 字符表示
- 其他设置
- 标记
- 用户列表
- 键绑定
- 印刷
- 后台加载和保存
- 缩放
- **Lexer**
- 图像
- 不推荐的消息
- 建筑 **Scintilla**
- 行结尾
- 样式定义
- 边距
- 支持突出显示
- 指标
- 通话提示
- 弹出编辑菜单
- 直接访问
- 折叠
- 长线
- **Lexer 对象**
- **GTK +**
- 编辑 **Scintilla** 从不支持的消息
- 单词
- 插入，选择和热点样式
- 注释
- 选项卡和缩进指南
- 自动完成
- 键盘命令
- 宏录制
- 多个视图
- 换行
- 可访问性
- 通知
- **临时消息**
- 删除功能

具有表单名称的消息 **SCI_SETxxxxx** 通常具有伴随 **SCI_GETxxxxx**。为了节省繁琐的重复，如果 **SCI_GETxxxxx** 消息返回消息设置的值 **SCI_SETxxxxx**，**SET** 则描述 **GET** 例程并且例程留给您想象。

文本检索和修改

Scintilla 文档中的每个字节都与一个样式信息字节相关联。字符字节和样式字节的组合称为单元格。样式字节被解释为样式数组的索引。

在本文档中，即使使用多字节字符，“字符”通常也指一个字节。长度测量字节数，而不是这些字节中的字符数。

Scintilla 文档中的位置是指该角色之前的字符或间隙。文档中的第一个字符是 0，第二个字符是 1，依此类推。如果文档包含 **nLen** 字符，则最后一个字符编号为 **nLen-1**。插入符号存在于字符位置之间，可以位于第一个字符（0）之前到最后一个字符（**nLen**）之后。

在有两个字符字节构成一个字符的地方，插入符号不能到达的地方。如果文档中包含来自日语等语言的 **DBCS** 字符，或者使用回车后跟换行的 **CP / M** 标准标记行结束时，会发生这种情况。的 **INVALID_POSITION** 常数（-1）表示文档中的无效位置。

Scintilla 中的所有文本行都具有相同的高度，并且此高度是根据任何当前样式中的最大字体计算的。这种限制是为了表现；如果行的高度不同，那么涉及文本定位的计算将要求首先设置样式。

```
SCI_GETTEXT(int length, char *text) → int
SCI_SETTEXT(<unused>, const char *text)
SCI_SETSAVEPOINT
SCI_GETLINE(int line, char *text) → int
SCI_REPLACESEL(<unused>, const char *text)
SCI_SETREADONLY(bool readOnly)
SCI_GETREADONLY → bool
SCI_GETTEXTRANGE(<unused>, Sci_TextRange *tr) → int
SCI_ALLOCATE(int bytes)
SCI_ADDTEXT(int length, const char *text)
SCI_ADDSTYLEDTEXT(int length, cell *c)
SCI_APPENDTEXT(int length, const char *text)
SCI_INSERTTEXT(int pos, const char *text)
SCI_CHANGEINSERTION(int length, const char *text)
SCI_CLEARALL
SCI_DELETERANGE(int start, int lengthDelete)
SCI_CLEARDOCUMENTSTYLE
SCI_GETCHARAT(int pos) → int
SCI_GETSTYLEAT(int pos) → int
SCI_GETSTYLEDTEXT(<unused>, Sci_TextRange *tr) → int
SCI_RELEASEALLEXTENDEDSTYLES
SCI_ALLOCATEEXTENDEDSTYLES(int numberStyles) → int
SCI_TARGETASUTF8(<unused>, char *s) → int
SCI_ENCODEDFROMUTF8(const char *utf8, char *encoded) → int
SCI_SETLENGTHFORENCODE(int bytes)
```

SCI_GETTEXT (int length, char *text NUL-terminated) →int

这将返回 *length* 文档开头的-1 个字符加一个终止 0 字符。要收集文档中的所有文本，请使用 **SCI_GETLENGTH** 获取文档中的字符数（*nLen*），分配长度 *nLen+1* 字节的字符缓冲区，然后调用 **SCI_GETTEXT(nLen+1, char *text)**。如果 *text* 参数为 0，则返回应分配用于存储整个文档的长度。如果您随后保存文本，则应使用 **SCI_SETSAVEPOINT** 将文本标记为未修改。

也可以看看：[SCI_GETSELTEXT](#)，[SCI_GETCURLINE](#)，[SCI_GETLINE](#)，[SCI_GETSTYLEDTEXT](#)，[SCI_GETTEXTRANGE](#)

SCI_SETTEXT (<unused>, const char * text)

这将使用您传入的零终止文本字符串替换文档中的所有文本。

SCI_SETSAVEPOINT

此消息告诉 Scintilla 文档的当前状态未经修改。这通常在保存或加载文件时完成，因此名称为“保存点”。当 Scintilla 执行撤消和重做操作时，它通知容器它已输入或离开保存点 [SCN_SAVEPOINTREACHED](#) 和 [SCN_SAVEPOINTLEFT](#) 通知消息，允许容器知道文件是否应被视为脏。

也可以看看：[SCI_EMPTYUNDOBUFFER](#)，[SCI_GETMODIFY](#)

SCI_GETLINE (int line, char * text) →int

这将在用文本定义的缓冲区填充指定行的内容（行从 0 开始）。缓冲区不以 0 字符终止。您可以确保缓冲区足够长，以便文本使用 **SCI_LINELENGTH(int line)**。返回值是复制到缓冲区的字符数。返回的文本包括任何行尾字符。如果要求文档行范围之外的行号，则复制 0 个字符。如果 text 参数为 0，则返回应分配用于存储整行的长度。

也可以看看： **SCI_GETCURLINE**, **SCI_GETSELTEXT**, **SCI_GETTEXTRANGE**, **SCI_GETSTYLEDTEXT**, **SCI_GETTEXT**

SCI_REPLACESEL (<unused>, const char * text) 锚点和当前位置

之间当前选择的文本被 0 终止的文本字符串替换。如果锚点和当前位置相同，则将文本插入插入符号位置。插入符号位于插入的文本之后，插入符号被滚动到视图中。

SCI_SETREADONLY (bool readOnly)

SCI_GETREADONLY→bool

这些消息设置并获取文档的只读标志。如果将文档标记为只读，则尝试修改文本会导致 **SCN_MODIFYATTEMPTRO** 通知。

SCI_GETTEXTRANGE (<未使用>, Sci_TextRange * TR) →INT

此收集的位置之间的文本 cpMin 和 cpMax 对，并将其复制 lpstrText（见 struct Sci_TextRange 在 Scintilla.h）。如果 cpMax 为 -1，则将文本返回到文档的末尾。文本以 0 结尾，因此您必须提供一个至少比您希望读取的字符数多 1 个字符的缓冲区。返回值是返回文本的长度，不包括终止 0。

也可以看看： **SCI_GETSELTEXT**, **SCI_GETLINE**, **SCI_GETCURLINE**, **SCI_GETSTYLEDTEXT**, **SCI_GETTEXT**

SCI_GETSTYLEDTEXT (<unused>, Sci_TextRange * tr) →int

它将样式化文本收集到一个缓冲区中，每个单元格使用两个字节，每个字符串位于较低地址，字样位于较高地址。的位置之间的字符 cpMin，并 cpMax 拷贝到 lpstrText（见 struct Sci_TextRange 的 Scintilla.h）。两个 0 字节被添加到文本的末尾，因此 lpstrText 指向的缓冲区必须至少为 2*(cpMax-cpMin)+2 字节长。没有检查 cpMin 或的合理值 cpMax。文档外的位置返回字符代码和样式字节 0。

也可以看看： **SCI_GETSELTEXT**, **SCI_GETLINE**, **SCI_GETCURLINE**, **SCI_GETTEXTRANGE**, **SCI_GETTEXT**

SCI_ALLOCATE (int bytes)

分配足够大的文档缓冲区以存储给定数量的字节。该文件不会小于其当前内容。

SCI_ADDTEXT (int length, const char * text)

这将在当前位置插入 length 字符串 text 中的第一个字符。这将包括您可能希望停止插入操作的字符串中的任何 0。当前位置设置在插入文本的末尾，但不会滚动到视图中。

SCI_ADDSTYLEDTEXT (int length, cell * c)

这个行为就像 **SCI_ADDTEXT**，但插入样式文本。

SCI_APPENDTEXT (int length, const char * text)

这将 *length* 字符串中的第一个字符 添加 *text* 到文档的末尾。这将包括您可能希望停止操作的字符串中的任何 0。当前选择不会更改，新文本不会滚动到视图中。

SCI_INSERTTEXT (int pos, const char * text) 如果为-1，

则将零终止 *text* 字符串插入位置 *pos* 或当前位置 *pos*。如果当前位置在插入点之后，则它与其周围文本一起移动但不执行滚动。

SCI_CHANGEINSERTION (int length, const char * text)

这只能从 **SC_MOD_INSERTCHECK** 通知处理程序调用，并将插入的文本更改为提供的文本。

SCI_CLEARALL

除非文档是只读的，否则将删除所有文本。

SCI_DELETERANGE (int start, int lengthDelete)

删除文档中的一系列文本。

SCI_CLEARDOCUMENTSTYLE

当想要完全重新放置文档时，例如在选择词法分析器之后，

SCI_CLEARDOCUMENTSTYLE 可以使用它来清除所有样式信息并重置折叠状态。

SCI_GETCHARAT (int pos) →int

这将返回 *pos* 文档中的字符，如果 *pos* 为负数或超过文档末尾，则返回 0 。

SCI_GETSTYLEAT (int pos) →int

这将返回 *pos* 文档中的样式，如果 *pos* 为负或超过文档末尾，则返回 0 。

SCI_RELEASEALLEXTENDEDSTYLES

SCI_ALLOCATEEXTENDEDSTYLES (int numberStyles) →int

扩展样式用于文本边距和注释等功能以及 Scintilla 内部功能。它们在 0..255 范围之外，用于与文档字节相关联的样式字节。这些函数管理扩展样式的使用，以确保组件在定义样式时协作。**SCI_RELEASEALLEXTENDEDSTYLES** 释放容器分配的任何扩展样式。**SCI_ALLOCATEEXTENDEDSTYLES** 在字节样式值之后分配一系列样式编号，并返回第一个分配的样式的编号。在调用 **SCI_MARGINSETSTYLEOFFSET** 或 **SCI_ANNOTATIONSETSTYLEOFFSET** 之前，应分配边距和注释样式的范围。

Sci_TextRange 和 Sci_CharacterRange

这些结构被定义为是完全一样的形状的 Win32 **TEXTRANGE** 和 **CHARRANGE**，让这把 Scintilla 的一个 RichEdit 中的老代码将工作。

在未来版本中，**Sci_PositionCR** 当在所有平台上为 64 位构建 Scintilla 时，类型将被重新定义为 64 位。

```
typedef long Sci_PositionCR;

struct Sci_CharacterRange {
    Sci_PositionCR cpMin;
    Sci_PositionCR cpMax;
};

struct Sci_TextRange {
    struct Sci_CharacterRange chrg;
    char * lpstrText;
};
```

仅适用于 GTK +, Cocoa 和 Windows: 访问编码文本

SCI_TARGETASUTF8 (<unused>, char *s) →int

此方法检索编码为 UTF-8 的目标值，这是 GTK + 的默认编码，因此对于检索用于用户界面其他部分的文本非常有用，例如查找和替换对话框。返回编码文本的长度（以字节为单位）。Cocoa 使用 UTF-16，它可以很容易地从 UTF-8 转换，因此这种方法可用于从支持的各种编码中执行更复杂的转码工作。

SCI_ENCODEDFROMUTF8 (const char * utf8, char * encoded) →int

SCI_SETLENGTHFORENCODE (int bytes)

SCI_ENCODEDFROMUTF8 将 UTF-8 字符串转换为文档的编码，这对于获取查找对话框的结果很有用，例如，接收字符串可以在文档中搜索。由于文本可以包含 `nul` 个字节，因此该 **SCI_SETLENGTHFORENCODE** 方法可用于设置要转换的长度。如果设置为 -1，则通过查找 `nul` 字节来确定长度。返回转换后的字符串的长度。

搜索

有搜索文本和正则表达式的方法。大多数应用程序应使用

SCI_SEARCHINTARGET 作为其搜索实现的基础。其他呼叫增加了这个或之前实现的 **SCI_SEARCHINTARGET**。

基本正则表达式支持是有限的，只应用于简单情况和初始开发。可以通过设置 **SCFIND_CXX11REGEX** 搜索标志来使用 C ++ 运行时 <regex> 库。可以通过使用已 **NO_CXX11_REGEX** 定义的 Scintilla 编译来禁用 C ++ 11 <regex> 支持。可以将不同的正则表达式库[集成到 Scintilla 中](#)，也可以通过 **SCI_GETCHARACTERPOINTER** 直接访问缓冲区内容从容器中[调用](#)。

使用目标搜索和替换

搜索可以在目标范围内执行 **SCI_SEARCHINTARGET**，使用计数字符串来搜索空字符。它返回匹配文本范围的开始位置，或者返回 -1 表示失败，在这种情况下不移动目标。所使用的标志 **SCI_SEARCHINTARGET**，例如 **SCFIND_MATCHCASE**，

SCFIND_WHOLEWORD, SCFIND_WORDSTART, 和 SCFIND_REGEX 可以设置 SCI_SETSEARCHFLAGS。

```
SCI_SETTARGETSTART(int start)
SCI_GETTARGETSTART → position
SCI_SETTARGETEND(int end)
SCI_GETTARGETEND → position
SCI_SETTARGETRANGE(int start, int end)
SCI_TARGETFROMSELECTION
SCI_TARGETWHOLEDOCUMENT
SCI_SETSEARCHFLAGS(int searchFlags)
SCI_GETSEARCHFLAGS → int
SCI_SEARCHINTARGET(int length, const char *text) → int
SCI_GETTARGETTEXT(<unused>, char *text) → int
SCI_REPLACETARGET(int length, const char *text) → int
SCI_REPLACETARGETRE(int length, const char *text) → int
SCI_GETTAG(int tagNumber, char *tagValue) → int
```

SCI_SETTARGETSTART (int start)

SCI_GETTARGETSTART→位置

SCI_SETTARGETEND (int end)

SCI_GETTARGETEND→位置

SCI_SETTARGETRANGE (int start, int end)

这些函数设置并返回目标的开始和结束。搜索时, 您可以将 start 设置为大于 end, 以查找目标中的最后一个匹配文本, 而不是第一个匹配的文本。目标也是成功的 **SCI_SEARCHINTARGET**。

SCI_TARGETFROMSELECTION

将目标开始和结束设置为选择的开始和结束位置。

SCI_TARGETWHOLEDOCUMENT

将目标开始设置为文档的开头, 将目标结束设置为文档的结尾。

SCI_SETSEARCHFLAGS (INT searchFlags)

SCI_GETSEARCHFLAGS→INT

这些获取和设置 *searchFlags* 使用 **SCI_SEARCHINTARGET**。有几个选项标志, 包括简单的正则表达式搜索。

SCI_SEARCHINTARGET (int length, const char * text) →int

这将搜索由 **SCI_SETTARGETSTART** 和定义的目标中第一次出现的文本字符串

SCI_SETTARGETEND。文本字符串不是零终止; 大小由设定 *length*。搜索由设置的搜索标志修改 **SCI_SETSEARCHFLAGS**。如果搜索成功, 则将目标设置为找到的文本, 返回值是匹配文本的开头位置。如果搜索失败, 则结果为-1。

SCI_GETTARGETTEXT (<unused>, char * text) →int

检索目标中的值。

SCI_REPLACETARGET (int length, const char * text) →int

如果 *length* 为-1, *text* 则为零终止字符串, 否则 *length* 设置要替换目标的字符

数。更换后，目标范围是指替换文本。返回值是替换字符串的长度。
请注意，删除文档中文本的推荐方法是将目标设置为要删除的文本，并使用空字符串执行替换目标。

SCI_REPLACETARGETRE (int length, const char * text) →int

这将使用正则表达式替换目标。如果 *length* 为-1，*text* 则为零终止字符串，否则 *length* 为要使用的字符数。替换字符串被从文本串与任何序列形成 \1 通过 \9 替换从最近的正则表达式搜索标记的匹配。\\0 将替换为最近搜索中的所有匹配文本。更换后，目标范围是指替换文本。返回值是替换字符串的长度。

SCI_GETTAG (int tagNumber, char * tagValue NUL 终止) →int

在正则表达式搜索中查找标记表达式匹配的文本。如果应用程序想要解释替换字符串本身，这将非常有用。

也可以看看： [SCI_FINDTEXT](#)

searchFlags

一些搜索例程使用标志选项，其中包括简单的正则表达式搜索。通过添加它们来组合标志选项：

SCFIND_MATCHCASE	只与匹配搜索字符串大小写的文本匹配。
SCFIND_WHOLEWORD	仅当前后字符不是由定义的字符时才会出现匹配 SCI_SETWORDCHARS 。
SCFIND_WORDSTART	仅当前面的字符不是由定义的单词字符时才会发生匹配 SCI_SETWORDCHARS 。
SCFIND_REGEX	搜索字符串应解释为正则表达式。除非结合使用，否则使用 Scintilla 的基本实现 SCFIND_CXX11REGEX 。
SCFIND_POSIX	通过解释裸（和）标记部分而不是\（和\\），以更符合 POSIX 的方式处理正则表达式。 SCFIND_CXX11REGEX 设置时无效。
SCFIND_CXX11REGEX	此标志可以设置为使用 C ++ 11 <regex>而不是 Scintilla 的基本正则表达式。如果正则表达式无效，则返回-1 并将 status 设置为 SC_STATUS_WARN_REGEX 。ECMAScript 标志在 regex 对象上设置，UTF-8 文档将表现出符合 Unicode 的行为。对于 MSVC，其中 wchar_t 是 16 位，则 regular 表达式“.”将匹配单个星体平面字符。编译器之间可能存在其他差异。还必须 SCFIND_REGEX 设定。

在正则表达式中，使用 Scintilla 的基本实现，解释的特殊字符是：

- 匹配任何角色
- \\(这标志着标记匹配的区域开始。
- \\) 这标志着标记区域的结束。

<code>\n</code>	其中 <code>n</code> 是 1 到 9 指的是第一至第九标记区替换时。例如，如果搜索字符串是 <code>Fred\([1-9]\)XXX</code> 和替换字符串 <code>Sam\1YYY</code> ，则应用于 <code>Fred2XXX</code> 此将生成 <code>Sam2YYY</code> 。 <code>\0</code> 指的是所有匹配的文本。
<code>\<</code>	这与使用 <code>Scintilla</code> 的单词定义匹配单词的开头。
<code>\></code>	这与使用 <code>Scintilla</code> 的单词定义匹配单词的结尾。
<code>\x</code>	这允许您使用否则具有特殊含义的字符 <code>x</code> 。例如， <code>\[</code> 将被解释为 <code>[</code> 而不是字符集的开头。
<code>[...]</code>	这表示一组字符，例如， <code>[abc]</code> 表示字符 <code>a</code> ， <code>b</code> 或 <code>c</code> 中的任何一个。您还可以使用范围，例如 <code>[az]</code> 表示任何小写字母。
<code>[^...]</code>	集合中字符的补充。例如， <code>[^ A-Za-z]</code> 表示除字母字符外的任何字符。
<code>^</code>	这匹配一行的开头（除非在一个集合中使用，见上文）。
<code>\$</code>	这匹配一行的结尾。
<code>*</code>	这匹配 0 次或更多次。例如， <code>Sa*m</code> 相匹配 <code>Sm</code> ， <code>Sam</code> ， <code>Saam</code> ， <code>Saaam</code> 等等。
<code>+</code>	这匹配 1 次或更多次。例如， <code>Sa+m</code> 相匹配 <code>Sam</code> ， <code>Saam</code> ， <code>Saaam</code> 等等。

正则表达式仅匹配单行内的范围，从不匹配多行。

当使用 `SCFIND_CXX11REGEX` 更多功能时，通常类似于 JavaScript 中的正则表达式支持。有关支持的内容的详细信息，请参阅 C++ 运行时的文档。

```

SCI_FINDTEXT(int searchFlags, Sci_TextToFind *ft) → position
SCI_SEARCHANCHOR
SCI_SEARCHNEXT(int searchFlags, const char *text) → int
SCI_SEARCHPREV(int searchFlags, const char *text) → int

```

SCI_FINDTEXT (int searchFlags, Sci_TextToFind * ft) → position

此消息搜索文档中的文本。它不使用或移动当前选择。的 `searchFlags` 参数，控制上述搜索类型，其中包括正则表达式搜索。

您可以通过在开始之前设置搜索范围的结尾来向后搜索以查找先前出现的搜索字符串。

该 `Sci_TextToFind` 结构被定义在 `Scintilla.h`；设置 `chrg.cpMin` 并 `chrg.cpMax` 使用文档中的位置范围进行搜索。您可以通过设置 `chrg.cpMax` 小于而向后搜索 `chrg.cpMin`。将 `lpstrText` 成员设置 `Sci_TextToFind` 为指向包含搜索模式的零终止文本字符串。如果您的语言 `Sci_TextToFind` 难以使用，则应考虑使用 `SCI_SEARCHINTARGET`。

如果搜索失败，则返回值为 -1；如果成功，则返回值的开头位置。在 `chrgText.cpMin` 和 `chrgText.cpMax` 成员 `Sci_TextToFind` 与找到的文本的开始和结束位置填写。

也可以看看： `SCI_SEARCHINTARGET`

Sci_TextToFind

此结构被定义为具有与 **FINDTEXT** 将 Scintilla 视为 RichEdit 控件的旧代码的 Win32 结构完全相同的形状。

```
struct Sci_TextToFind {
    struct Sci_CharacterRange chrg; //范围搜索
    const char * lpstrText; //搜索模式（零终止）
    struct Sci_CharacterRange chrgText; //作为匹配文本的位置返回
};
```

SCI_SEARCHANCHOR

SCI_SEARCHNEXT (int searchFlags, const char * text) →int

SCI_SEARCHPREV (int searchFlags, const char * text) →int

这些消息提供可重定位搜索支持。这允许多个增量交互式搜索被宏记录，同时仍然将选择设置为找到的文本，因此查找/选择操作是自包含的。如果启用了宏录制，则这三条消息会发送[通知](#)。[SCN_MACRORECORD](#)

SCI_SEARCHANCHOR 设置当前选择的开始 **SCI_SEARCHNEXT** 和使用的搜索起始点 **SCI_SEARCHPREV**，即选择的结尾越接近文档的开头。你应该在调用 **SCI_SEARCHNEXT** 或 之前调用它 **SCI_SEARCHPREV**。

SCI_SEARCHNEXT 并 **SCI_SEARCHPREV** 搜索文本指向的零终止搜索字符串的下一个和上一个匹配项。搜索由 [searchFlags](#)。修改。

如果未找到任何内容，则返回值为-1，否则返回值是匹配文本的起始位置。选择已更新以显示匹配的文本，但不会滚动到视图中。

另见：[SCI_SEARCHINTARGET](#)， [SCI_FINDTEXT](#)

改写

[SCI_SETOVERTYPE](#)(bool overType)
[SCI_GETOVERTYPE](#) → bool

SCI_SETOVERTYPE (bool overType)

SCI_GETOVERTYPE→bool

当启用改写时，每个键入的字符都会替换文本插入符右侧的字符。禁用改写时，将在插入符号处插入字符。如果过度加工有效则 **SCI_GETOVERTYPE** 返回 **true**

(1)，否则 **false** 返回 (0)。使用 **SCI_SETOVERTYPE** 设置改写模式。

剪切，复制和粘贴

[SCI_CUT](#)
[SCI_COPY](#)
[SCI_PASTE](#)
[SCI_CLEAR](#)
[SCI_CANPASTE](#) → bool
[SCI_COPYRANGE](#)(int start, int end)

SCI_COPYTEXT(int length, const char *text)
SCI_COPYALLOWLINE
SCI_SETPASTECONVERTENDINGS(bool convert)
SCI_GETPASTECONVERTENDINGS → bool

SCI_CUT
SCI_COPY
SCI_PASTE
SCI_CLEAR
SCI_CANPASTE→bool
SCI_COPYALLOWLINE

这些命令执行将数据剪切和复制到剪贴板，从剪贴板粘贴到文档以及清除文档的标准任务。**SCI_CANPASTE** 如果文档不是只读且选择不包含受保护文本，则返回非零值。如果您需要“可以复制”或“可以剪切”，请使用 **SCI_GETSELECTIONEMPTY()**，如果存在任何非空选择范围，则表示剪贴板的复制或剪切应该起作用，这将为零。

除非文档是只读的，否则 GTK +不会真正支持 **SCI_CANPASTE** 并始终返回 **true**。

在 X 上，剪贴板是异步的，可能需要在目标和源应用程序之间发送多条消息。来自 **SCI_PASTE** 的数据不会立即到达文档。

SCI_COPYALLOWLINE 与 **SCI_COPY** 的工作方式相同，只是如果选择为空，则复制当前行。在 Windows 上，一个额外的“MSDEVLineSelect”标记被添加到剪贴板，然后用于 **SCI_PASTE** 粘贴当前行之前的整行。

SCI_COPYRANGE (int start, int end)
SCI_COPYTEXT (int length, const char * text)

SCI_COPYRANGE 将文档范围从文档 **SCI_COPYTEXT** 复制到系统剪贴板，并将提供的文本复制到系统剪贴板。

SCI_SETPASTECONVERTENDINGS (bool convert)
SCI_GETPASTECONVERTENDINGS→bool
如果设置了此属性，则粘贴文本时，任何行结束都将转换为与 **SCI_SETEOLMODE** 设置的文档行尾模式 **匹配**。默认为 **true**。

错误处理

SCI_SETSTATUS(int status)
SCI_GETSTATUS → int

SCI_SETSTATUS (int status)
SCI_GETSTATUS→int

如果发生错误，Scintilla 可能会设置一个可以检索的内部错误号 **SCI_GETSTATUS**。清除错误状态调用 **SCI_SETSTATUS(0)**。状态值从 1 到 999 是错误，状态 **SC_STATUS_WARN_START** (1000) 及以上是警告。目前定义的状态是：

SC_STATUS_OK	0	没有失败
SC_STATUS_FAILURE	1	一般失败
SC_STATUS_BADALLOC	2	记忆力已经耗尽
SC_STATUS_WARN_REGEX	1001	正则表达式无效

撤消和重做

Scintilla 具有多级撤消和重做。它将继续收集可撤消的操作，直到内存耗尽。Scintilla 保存更改文档的操作。Scintilla 不保存插入符号和选择移动，查看滚动等。键入或删除的序列被压缩为单个事务，以便更容易撤消和重做一个明智的细节级别。操作序列可以组合成作为一个单元撤消的事务。之间发生这些序列 **SCI_BEGINUNDOACTION** 和 **SCI_ENDUNDOACTION** 消息。这些事务可以嵌套，只有顶级序列作为单位撤消。

```

SCI_UNDO
SCI_CANUNDO → bool
SCI_EMPTYUNDOBUFFER
SCI_REDO
SCI_CANREDO → bool
SCI_SETUNDOCOLLECTION(bool collectUndo)
SCI_GETUNDOCOLLECTION → bool
SCI_BEGINUNDOACTION
SCI_ENDUNDOACTION
SCI_ADDUNDOACTION(int token, int flags)

```

SCI_UNDO

SCI_CANUNDO→bool

SCI_UNDO 撤消一个动作，或者如果撤消缓冲区已到达某个 **SCI_ENDUNDOACTION** 点，则所有动作都返回到相应的动作 **SCI_BEGINUNDOACTION**。

SCI_CANUNDO 如果没有要撤消的内容，则返回 0;如果有，则返回 1。您通常会使用此消息的结果来启用/禁用“编辑”菜单“撤消”命令。

SCI_REDO

SCI_CANREDO→bool

SCI_REDO 撤消上一次 **SCI_UNDO** 操作的效果。

SCI_CANREDO 如果没有要重做的操作，则返回 0;如果有重做的撤消操作，则返回 1。您通常可以使用此消息的结果来启用/禁用“编辑”菜单“重做”命令。

SCI_EMPTYUNDOBUFFER

此命令告诉 Scintilla 忘记任何已保存的撤消或重做历史记录。它还将保存点设置为撤消缓冲区的开头，因此文档似乎未经修改。这不会导致将

SCN_SAVEPOINTREACHED 通知发送到容器。

也可以看看: **SCI_SETSAVEPOINT**

SCI_SETUNDOCOLLECTION (bool collectUndo)

SCI_GETUNDOCOLLECTION → bool

您可以控制 Scintilla 是否收集撤消信息 **SCI_SETUNDOCOLLECTION**。通过 **true** (1) 收集信息和 **false** (0) 停止收集。如果停止收集，还应该使用 **SCI_EMPTYUNDOBUFFER** 以避免撤消缓冲区与缓冲区中的数据不同步。

如果使用 Scintilla 存储程序生成的文本（日志视图）或者经常删除和重新生成文本的显示窗口，您可能希望关闭保存撤消信息。

SCI_BEGINUNDOACTION

SCI_ENDUNDOACTION

将这两条消息发送给 Scintilla，以标记要作为一个操作撤消所有操作的一组操作的开始和结束，但必须生成多个操作。或者，您可以使用这些操作来标记一组操作，如果它们被撤消，您不希望将这些操作与前面或后面的操作结合使用。

SCI_ADDUNDOACTION (int token, int flags)

容器可以通过调用将自己的操作添加到撤消堆栈中，**SCI_ADDUNDOACTION** 并且 当需要撤消 () 或重做 () 操作时，**SCN_MODIFIED** 将通过 **SC_MOD_CONTAINER** 标志将通知发送到容器。提供的标记参数在通知字段中返回。

SC_PERFORMED_UNDO **SC_PERFORMED_REDO** token

例如，如果容器想要允许撤销和重做“切换书签”命令，那么

SCI_ADDUNDOACTION(line, 0) 每次执行命令时它都可以调用。然后，当它收到撤消或重做的通知时，它会切换令牌字段给定的行上的书签。如果存在需要存储到撤消堆栈中的不同类型的命令或参数，则容器应为文档维护其自己的堆栈，并使用该堆栈中的当前位置作为参数 **SCI_ADDUNDOACTION(line)**。

SCI_ADDUNDOACTION 除非用 **SCI_BEGINUNDOACTION** 和分组，否则命令不会组合到一个撤消事务中 **SCI_ENDUNDOACTION**。

flags 参数可以是 **UNDO_MAY_COALESCE** (1) 如果容器操作可以与任何插入和删除操作合并为单个复合操作，否则为 0。Coalescing 将可合并容器操作视为透明，因此仍然只将看似键入的插入组合在一起或看起来像 Backspace 或 Delete 键的多次使用的删除。

选择和信息

Scintilla 保持一个选择，在两个点，锚点和当前位置之间延伸。如果锚点和当前位置相同，则没有选定的文本。文档中的位置范围从 0（在第一个字符之前）到文档大小（在最后一个字符之后）。如果您使用消息，则没有什么可以阻止您设置位于 CRLF 对中间或 2 字节字符中间的位置。但是，键盘命令不会将插入符移动到这些位置。

SCI_GETTEXTLENGTH → int

SCI_GETLENGTH → int

SCI_GETLINECOUNT → int

SCI_LINESONSCREEN → int


```

SCI_GETMODIFY → bool
SCI_SETSEL(int anchor, int caret)
SCI_GOTOPOS(int caret)
SCI_GOTOLINE(int line)
SCI_SETCURRENTPOS(int caret)
SCI_GETCURRENTPOS → position
SCI_SETANCHOR(int anchor)
SCI_GETANCHOR → position
SCI_SETSELECTIONSTART(int anchor)
SCI_GETSELECTIONSTART → position
SCI_SETSELECTIONEND(int caret)
SCI_GETSELECTIONEND → position
SCI_SETEMPYSELECTION(int caret)
SCI_SELECTALL
SCI_LINEFROMPOSITION(int pos) → int
SCI_POSITIONFROMLINE(int line) → position
SCI_GETLINEENDPOSITION(int line) → position
SCI_LINELENGTH(int line) → int
SCI_GETCOLUMN(int pos) → int
SCI_FINDCOLUMN(int line, int column) → int
SCI_POSITIONFROMPOINT(int x, int y) → position
SCI_POSITIONFROMPOINTCLOSE(int x, int y) → position
SCI_CHARPOSITIONFROMPOINT(int x, int y) → position
SCI_CHARPOSITIONFROMPOINTCLOSE(int x, int y) → position
SCI_POINTXFROMPOSITION(<unused>, int pos) → int
SCI_POINTYFROMPOSITION(<unused>, int pos) → int
SCI_HIDESELECTION(bool hide)
SCI_GETSELTEXT(<unused>, char *text) → int
SCI_GETCURLINE(int length, char *text) → int
SCI_SELECTIONISRECTANGLE → bool
SCI_SETSELECTIONMODE(int selectionMode)
SCI_GETSELECTIONMODE → int
SCI_GETMOVEEXTENDSSELECTION → bool
SCI_GETLINESELSTARTPOSITION(int line) → position
SCI_GETLINESELENDPOSITION(int line) → position
SCI_MOVECARETINSIDEVIEW
SCI_POSITIONBEFORE(int pos) → position
SCI_POSITIONAFTER(int pos) → position
SCI_TEXTWIDTH(int style, const char *text) → int
SCI_TEXTHEIGHT(int line) → int
SCI_CHOOSECARETX
SCI_MOVESELECTEDLINESUP
SCI_MOVESELECTEDLINESDOWN
SCI_SETMOUSESELECTIONRECTANGULARSWITCH(bool mouseSelectionRectangularSwitch)
SCI_GETMOUSESELECTIONRECTANGULARSWITCH → bool

```

SCI_GETTEXTLENGTH→int

SCI_GETLENGTH→int

这两个消息都以字节为单位返回文档的长度。

SCI_GETLINECOUNT→int

返回文档中的行数。空文档包含 1 行。仅包含行结束序列的文档有 2 行。

SCI_LINESONSCREEN→int

返回屏幕上可见的完整行数。在恒定的线高度下，这是可用的垂直空间除以线

间距。除非您安排将窗口大小调整为整数行，否则视图底部可能会显示部分线条。

SCI_GETMODIFY→bool

如果文档被修改则返回非零值，如果未修改则返回 0。文档的修改状态由相对于保存点的撤消位置确定。[SCI_SETSAVEPOINT](#) 通常在将数据保存到文件时设置保存点。

如果您需要在文档被修改时收到通知，Scintilla 会通知容器它已进入或离开带有 [SCN_SAVEPOINTREACHED](#) 和 [通知消息](#) 的保存点。[SCN_SAVEPOINTLEFT](#)

SCI_SETSEL (int anchor, int caret)

此消息设置锚点和当前位置。如果 *caret* 是否定的，则表示文档的结尾。如果 *anchor* 是否定的，则表示删除任何选择（即将锚点设置为相同的位置 *caret*）。此操作后，插入符号将滚动到视图中。

SCI_GOTOPOS (int caret)

这将删除所有选择，设置插入符号 *caret* 并滚动视图以在必要时显示插入符号。它相当于 [SCI_SETSEL\(caret, caret\)](#)。锚位置设置与当前位置相同。

SCI_GOTOLINE (int line)

这将删除所有选择并在行号的开头设置插入符 *line* 并滚动视图（如果需要）以使其可见。锚位置设置与当前位置相同。如果 *line* 在文档中的行之外（第一行为 0），则行集是第一行或最后一行。

SCI_SETCURRENTPOS (int caret)

设置当前位置并在锚点和当前位置之间创建选择。插入符号不会滚动到视图中。

也可以看看：[SCI_SCROLLCARET](#)

SCI_GETCURRENTPOS→position

返回当前位置。

SCI_SETANCHOR (int anchor)

这将设置锚点位置并在锚点位置和当前位置之间创建一个选择。插入符号不会滚动到视图中。

也可以看看：[SCI_SCROLLCARET](#)

SCI_GETANCHOR→position

返回当前锚位置。

SCI_SETSELECTIONSTART (int anchor)

SCI_SETSELECTIONEND (int caret)

这些基于锚位置小于当前位置的假设来设置选择。它们不会使插入符号可见。该表显示了使用这些消息后锚点的位置和当前位置。

新的价值	锚	插入符号
SCI_SETSELECTIONSTART	<i>anchor</i>	Max(<i>anchor</i> , <i>current</i>)
SCI_SETSELECTIONEND	Min(<i>anchor</i> , <i>caret</i>)	<i>caret</i>

也可以看看: [SCI_SCROLLCARET](#)

SCI_GETSELECTIONSTART→位置

SCI_GETSELECTIONEND→position

这些返回选择的开始和结束，而不考虑当前位置的哪一端以及哪个是锚点。

SCI_GETSELECTIONSTART 返回当前位置或锚位置中较小的一个。

SCI_GETSELECTIONEND 返回两个值中较大的一个。

SCI_SETEMPTYSELECTION (int caret)

这将删除所有选择并设置插入符号 *caret*。插入符号不会滚动到视图中。

SCI_SELECTALL

选择文档中的所有文本。当前位置不会滚动到视图中。

SCI_LINEFROMPOSITION (int pos) →int

此消息返回包含 *pos* 文档中位置的行。如果 *pos* ≤ 0，则返回值为 0。如果 *pos* 超出文档末尾，则返回值是最后一行。

SCI_POSITIONFROMLINE (int line) →position

返回与行首相对应的文档位置。如果 *line* 为负，则返回保持选择开始的行的位置。如果 *line* 大于文档中的行，则返回值为-1。如果 *line* 等于文档中的行数（即超过最后一行的 1 行），则返回值是文档的结尾。

SCI_GETLINEENDPOSITION (int line) →position

这将返回行尾的位置，在任何行结束字符之前。如果 *line* 是文档中的最后一行（没有任何行尾字符）或更大，则结果是文档的大小。如果 *line* 为否定，则结果未定义。

SCI_LINELENGTH (int line) →int

返回行的长度，包括任何行结束符。如果 *line* 为负数或超出文档中的最后一行，则结果为 0。如果您希望行的长度不包括任何行尾字符，请使用

[SCI_GETLINEENDPOSITION\(line\)](#) - [SCI_POSITIONFROMLINE\(line\)](#)。

SCI_GETSELTEXT (<unused>, char *text NUL-terminated) →int

将当前选定的文本和终止的 0 字节复制到 *text* 缓冲区。缓冲区大小应通过调用 *text* 参数的 NULL 指针来确定 [SCI_GETSELTEXT\(0,0\)](#)。这允许矩形和不连续的选择以及简单的选择。有关如何复制多个矩形选择和虚拟空间的信息，请参阅[多个选择](#)。

也可以看看: [SCI_GETCURLINE](#), [SCI_GETLINE](#), [SCI_GETTEXT](#), [SCI_GETSTYLEDTEXT](#), [SCI_GETTEXTRANGE](#)

SCI_GETCURLINE (int length, char * text NUL-terminated) →int

这将检索包含插入符号的行的文本，并返回插入符号行内的位置。传递 **char* text** 指向一个足够大的缓冲区来保存您想要检索的文本和终止 0 字符。设置 **length** 为缓冲区的长度，该长度必须至少为 1 才能保存终止 0 字符。如果 **text** 参数为 0，则返回应分配用于存储整个当前行的长度。

也可以看看：[SCI_GETSELTEXT](#)，[SCI_GETLINE](#)，[SCI_GETTEXT](#)，[SCI_GETSTYLEDTEXT](#)，[SCI_GETTEXTRANGE](#)

SCI_SELECTIONISRECTANGLE→bool

如果当前选择处于矩形模式，则返回 1，否则返回 0。

SCI_SETSELECTIONMODE (int selectionMode)

SCI_GETSELECTIONMODE→int

这两个函数设置并获得选择模式，可以是流 (**SC_SEL_STREAM= 0**) 或矩形 (**SC_SEL_RECTANGLE= 1**) 或行 (**SC_SEL_LINES= 2**) 或细矩形 (**SC_SEL_THIN= 3**)。在这些模式下设置时，常规插入符号移动将扩展或减少选择，直到通过具有相同值或具有相同值的调用取消模式 **SCI_CANCEL**。即使通过鼠标或常规扩展移动进行选择，**get** 函数也会返回当前模式。**SC_SEL_THIN** 是输入矩形选择后的模式，确保不选择任何字符。

SCI_GETMOVEEXTENDSSELECTION→bool

如果常规插入符号移动将扩展或减少选择，则返回 1，否则返回 0。

SCI_SETSELECTIONMODE 在打开和关闭之间切换此设置。

SCI_GETLINESELSTARTPOSITION (int line) →位置

SCI_GETLINESELENDPOSITION (int line) →position

检索给定行的选择的开始和结束位置，**INVALID_POSITION** 如果此行没有选择则返回。

SCI_MOVECARETINSIDEVIEW

如果插入符号位于视图的顶部或底部，则会将其移动到当前位置可见的最近的行。任何选择都会丢失。

SCI_POSITIONBEFORE (int pos) →位置

SCI_POSITIONAFTER (int pos) →位置

这些消息在考虑当前代码页的情况下返回文档中另一个位置之前和之后的位置。返回的最小位置为 0，最大值是文档中的最后位置。如果使用多字节字符内的位置调用，则返回该字符的开头/结尾的位置。

SCI_TEXTWIDTH (int style, const char * text) →int

这将返回给定的字符串的像素宽度 **style**，例如，可以用来决定行号边距的宽度，以便显示给定的数量数字。

SCI_TEXTHEIGHT (int line) →int

这将返回特定行的高度（以像素为单位）。目前所有线都是相同的高度。

SCI_GETCOLUMN (int pos) →int

此消息返回 *pos* 文档中位置的列号，其中考虑了制表符的宽度。这将返回之前行上最后一个选项卡的列号，以及最后 *pos* 一个选项卡和之间的字符数 *pos*。如果该行上没有制表符，则返回值是直到该行位置的字符数。在这两种情况下，双字节字符都算作单个字符。这可能仅适用于等宽字体。

SCI_FINDCOLUMN (int line, int column) →int

此消息返回 考虑选项卡宽度的 *column* 的位置 *line*。它将多字节字符视为单个列。列号与行号从 0 开始。

SCI_POSITIONFROMPOINT (int x, int y) →position

SCI_POSITIONFROMPOINTCLOSE (int x, int y) →position

SCI_POSITIONFROMPOINT 找到与点最接近的字符位置并且

SCI_POSITIONFROMPOINTCLOSE 类似但如果该点在窗口之外或者不接近任何字符，则返回-1。

SCI_CHARPOSITIONFROMPOINT (int x, int y) →位置

SCI_CHARPOSITIONFROMPOINTCLOSE (int x, int y) →position

SCI_CHARPOSITIONFROMPOINT 找到与点最接近的字符并且

SCI_CHARPOSITIONFROMPOINTCLOSE 类似但如果该点在窗口之外或者不接近任何字符，则返回-1。这类似于以前的方法但是找到字符而不是字符间位置。

SCI_POINTXFROMPOSITION (<unused>, int pos) →int

SCI_POINTYFROMPOSITION (<unused>, int pos) →int

这些消息返回 *pos* 文档中位置处文本的 x 和 y 显示像素位置。

SCI_HIDESELECTION (bool hide)

正常状态是通过将其绘制为 **SCI_SETSELEFORE** 和，使选择可见 **SCI_SETSELBACK**。但是，如果隐藏选择，则将其绘制为普通文本。

SCI_CHOOSECARETX

Scintilla 会记住用户明确水平移动到的最后位置的 x 值，然后在垂直移动时使用此值，例如使用向上和向下键。此消息将插入符号的当前 x 位置设置为记住的值。

SCI_MOVESELECTEDLINESUP

将选定的行向上移动一行，在选择后移动上面的行。选择将自动扩展到选择的第一行的开头和选择的最后一行的结尾。如果未选择任何内容，则将选择光标当前所在的行。

SCI_MOVESELECTEDLINESDOWN

将选定的行向下移动一行，在选择之前移动下面的行。选择将自动扩展到选择的第一行的开头和选择的最后一行的结尾。如果未选择任何内容，则将选择光标当前所在的行。

**SCI_SETMOUSESELECTIONRECTANGULARSWITCH (bool
mouseSelectionRectangularSwitch)**

SCI_GETMOUSESELECTIONRECTANGULARSWITCH→bool

启用或禁用在使用鼠标进行选择时切换到矩形选择模式的功能。启用此选项后，可以通过按相应的修改键将流模式下的鼠标选择切换为矩形模式。然后，即使再次释放修改键，它们也会坚持矩形模式。关闭此选项后，鼠标选择将始终保持选择开始的模式。默认情况下，它处于关闭状态。

按字符或 UTF-16 代码单位

大多数 Scintilla API 使用字节位置，但某些应用程序希望使用基于计数（UTF-32）字符或（UTF-16）代码单元的位置，或者需要与按字符或代码单位编写的其他代码进行通信。仅使用字节位置，这可能需要检查许多字节来计算文档中的字符或代码单元，但在某些情况下可以通过以字符或代码单元开始索引行来加速。

```
SCI_POSITIONRELATIVE(int pos, int relative) → position
SCI_POSITIONRELATIVECODEUNITS(int pos, int relative) → position
SCI_COUNTCHARACTERS(int start, int end) → int
SCI_COUNTCODEUNITS(int start, int end) → int
SCI_GETLINECHARACTERINDEX → int
SCI_ALLOCATELINECHARACTERINDEX(int lineCharacterIndex)
SCI_RELEASELINECHARACTERINDEX(int lineCharacterIndex)
SCI_LINEFROMINDEXPOSITION(int pos, int lineCharacterIndex) → int
SCI_INDEXPOSITIONFROMLINE(int line, int lineCharacterIndex) → position
```

SCI_POSITIONRELATIVE (int pos, int relative) →position

计算参数位置之前或之后的整个字符数并返回该位置。返回的最小位置为 0，最大值是文档中的最后位置。如果该位置超过文档结束，则返回 0。

SCI_COUNTCHARACTERS (int start, int end) →int

返回两个位置之间的整个字符数。

SCI_POSITIONRELATIVECODEUNITS (int pos, int relative) →位置

SCI_COUNTCODEUNITS (int start, int end) →int

这些是 UTF-16 版本 **SCI_POSITIONRELATIVE** 并 **SCI_COUNTCHARACTERS** 以 UTF-16 代码单位工作。

SCI_GETLINECHARACTERINDEX→int

如果有任何索引处于活动状态，则返回。如果整个字符被索引或者如果 UTF-16 代码单元被索引 **SC_LINECHARACTERINDEX_NONE(0)**,

SC_LINECHARACTERINDEX_UTF32(1)则 它可以是或者是一个或多个

SC_LINECHARACTERINDEX_UTF16(2)。目前只支持 UTF-8 文档的字符索引。

SCI_ALLOCATELINECHARACTERINDEX (int lineCharacterIndex)

SCI_RELEASELINECHARACTERINDEX (int lineCharacterIndex)

使用相同的枚举分配或释放一个或多个索引 **SCI_GETLINECHARACTERINDEX**。应用

程序的不同方面可能需要不同时期的索引，并应为这些时段分配。索引使用额外的内存，因此释放它们可以帮助减少内存，但它们也需要时间来重新计算。Scintilla 还可以分配索引以支持可访问性或输入法编辑器等功能。一次只为文档创建每种类型的一个索引。

SCI_LINEFROMINDEXPOSITION (int pos, int lineCharacterIndex) → int
SCI_INDEXPOSITIONFROMLINE (int line, int lineCharacterIndex)
→ position

可以通过 **SCI_LINEFROMINDEXPOSITION** 使用 **SC_LINECHARACTERINDEX_UTF32(1)** 或之一 调用找到特定字符或代码单元的文档行 **SC_LINECHARACTERINDEX_UTF16(2)**。逆操作，通过 **SCI_INDEXPOSITIONFROMLINE** 使用相同的 *lineCharacterIndex* 参数调用，从文档开始，以字符或代码单位查找文档行的起始位置。

多选和虚拟空间

```
SCI_SETMULTIPLESELECTION(bool multipleSelection)
SCI_GETMULTIPLESELECTION → bool
SCI_SETADDITIONALSELECTIONTYPING(bool additionalSelectionTyping)
SCI_GETADDITIONALSELECTIONTYPING → bool
SCI_SETMULTIPASTE(int multiPaste)
SCI_GETMULTIPASTE → int
SCI_SETVIRTUALSPACEOPTIONS(int virtualSpaceOptions)
SCI_GETVIRTUALSPACEOPTIONS → int
SCI_SETRECTANGULARSELECTIONMODIFIER(int modifier)
SCI_GETRECTANGULARSELECTIONMODIFIER → int
```

```
SCI_GETSELECTIONS → int
SCI_GETSELECTIONEMPTY → bool
SCI_CLEARSELECTIONS
SCI_SETSELECTION(int caret, int anchor)
SCI_ADDSELECTION(int caret, int anchor)
SCI_DROPSELECTIONN(int selection)
SCI_SETMAINSELECTION(int selection)
SCI_GETMAINSELECTION → int
```

```
SCI_SETSELECTIONNCARET(int selection, int caret)
SCI_GETSELECTIONNCARET(int selection) → position
SCI_SETSELECTIONNCARETVIRTUALSPACE(int selection, int space)
SCI_GETSELECTIONNCARETVIRTUALSPACE(int selection) → int
SCI_SETSELECTIONNANCHOR(int selection, int anchor)
SCI_GETSELECTIONNANCHOR(int selection) → position
SCI_SETSELECTIONNANCHORVIRTUALSPACE(int selection, int space)
SCI_GETSELECTIONNANCHORVIRTUALSPACE(int selection) → int
SCI_SETSELECTIONNSTART(int selection, int anchor)
SCI_GETSELECTIONNSTART(int selection) → position
SCI_SETSELECTIONNEND(int selection, int caret)
SCI_GETSELECTIONNEND(int selection) → position
```

```
SCI_SETRECTANGULARSELECTIONCARET(int caret)
SCI_GETRECTANGULARSELECTIONCARET → position
SCI_SETRECTANGULARSELECTIONCARETVIRTUALSPACE(int space)
SCI_GETRECTANGULARSELECTIONCARETVIRTUALSPACE → int
SCI_SETRECTANGULARSELECTIONANCHOR(int anchor)
SCI_GETRECTANGULARSELECTIONANCHOR → position
```

```

SCI_SETRECTANGULARSELECTIONANCHORVIRTUALSPACE(int space)
SCI_GETRECTANGULARSELECTIONANCHORVIRTUALSPACE → int

SCI_SETADDITIONALSELALPHA(alpha alpha)
SCI_GETADDITIONALSELALPHA → int
SCI_SETADDITIONALSELFORE(colour fore)
SCI_SETADDITIONALSELBACK(colour back)
SCI_SETADDITIONALCARETFORE(colour fore)
SCI_GETADDITIONALCARETFORE → colour
SCI_SETADDITIONALCARETSBLINK(bool additionalCaretBlink)
SCI_GETADDITIONALCARETSBLINK → bool
SCI_SETADDITIONALCARETSVISIBLE(bool additionalCaretVisible)
SCI_GETADDITIONALCARETSVISIBLE → bool

SCI_SWAPMAINANCHORCARET
SCI_ROTATESELECTION
SCI_MULTIPLESELECTADNEXT
SCI_MULTIPLESELECTADDEACH

```

一次可能有多个选择活动。通过在使用鼠标拖动时按住 **Ctrl** 键进行更多选择。最近的选择是主要选择，并确定自动显示文档的哪个部分。除主选择之外的任何选择都称为附加选择。上一节中的调用对主选项进行操作。始终至少有一个选择。选择可以简化为主要选择，**SCI_CANCEL** 通常映射到 **Esc** 键。

矩形选择作为多个选择处理，尽管记住原始矩形范围，以便可以对矩形选择不同地处理后续操作。例如，粘贴矩形选择将每个片段放在垂直列中。

虚拟空间是超出每条线末端的空间。可以将插入符移动到虚拟空间中，但是在文档键入或使用某些其他文本插入命令之前，不会向文档添加真实空间。

将不连续的选择复制到剪贴板时，每个选择都会按顺序添加到剪贴板文本中，不会出现分隔符。对于矩形选择，在每行的文本后添加文档的行尾。矩形选择始终从顶行复制到底部，而不是按选择顺序复制。不复制虚拟空间。

SCI_SETMULTIPLESELECTION (bool multipleSelection)
SCI_GETMULTIPLESELECTION → bool

启用或禁用多项选择。禁用多个选择时，在用鼠标拖动时按住 **Ctrl** 键不能选择多个范围。

SCI_SETADDITIONALSELECTIONTYPING (bool additionalSelectionTyping)
SCI_GETADDITIONALSELECTIONTYPING → bool

无论是同时打字，新行，左/右/上/下光标，退格键，删除，主页还是结束，都可以同时进行多项选择。还允许选择和字和行删除命令。

SCI_SETMULTIPASTE (int multiPaste)
SCI_GETMULTIPASTE → int

当粘贴到多个选择中时，粘贴的文本可以仅使用 **SC_MULTIPASTE_ONCE= 0** 进入主选择，或进入每个选择 **SC_MULTIPASTE_EACH= 1**。**SC_MULTIPASTE_ONCE** 是默认值。

SCI_SETVIRTUALSPACEOPTIONS (int virtualSpaceOptions)

SCI_GETVIRTUALSPACEOPTIONS→int

可以为矩形选择或在其他情况下或两者中启用或禁用虚拟空间。有三个位标志

SCVS_RECTANGULARSELECTION= 1, **SCVS_USERACCESSIBLE= 2** 和

SCVS_NOWRAPLINESSTART= 4, 可以单独设置。 **SCVS_NONE= 0**, 默认值, 禁用所有虚拟空间使用。

SCVS_NOWRAPLINESSTART 防止左箭头移动和选择包装到上一行。这通常与虚拟空间结合使用, 但是是独立设置, 因此无需虚拟空间。

SCI_SETRECTANGULARSELECTIONMODIFIER (int 修饰符)

SCI_GETRECTANGULARSELECTIONMODIFIER→int

在 GTK + 和 Qt 上, 用于指示与鼠标拖动组合时应创建矩形选区的键可以设置。三个可能的值是 **SCMOD_CTRL= 2**, **SCMOD_ALT= 4** (默认值) 或 **SCMOD_SUPER= 8**。由于 **SCMOD_ALT** 窗口管理器可能已经使用过窗口管理器, 因此窗口管理器可能需要进行配置以允许此选择。 **SCMOD_SUPER** 通常是系统相关的修饰键, 例如 Windows 键盘上的 Left Windows 键或 Mac 上的 Command 键。

SCI_GETSELECTIONS→int

返回当前活动的选择数。始终至少有一个选择。

SCI_GETSELECTIONEMPTY→bool

如果每个选定范围为空, 则返回 1, 否则返回 0。

SCI_CLEARSELECTIONS

将唯一的空选择设置为 0 作为唯一选择。

SCI_SETSELECTION (INT 插入符号, INT 锚)

从设置单个选择 *anchor* 来 *caret* 作为唯一的选择。

SCI_ADDSELECTION (INT 插入符号, 诠释锚)

从添加一个新的选择 *anchor*, 以 *caret* 作为主要的选择保留了所有其他选择其他选择。由于始终至少有一个选择, 要设置选择列表, 应添加第一个选择,

SCI_SETSELECTION 并随后添加选择 **SCI_ADDSELECTION**

SCI_DROPSELECTIONN (int selection)

如果有多个选择, 则删除指示的选择。如果这是主要选择, 则将前一个选择作为主要选项, 如果是第一个选择, 则最后一个选择成为主选。如果只有一个选择, 或者没有选择 *selection*, 则没有效果。

SCI_SETMAINSELECTION (int selection)

SCI_GETMAINSELECTION→int

其中一个选项是主选项, 用于确定自动显示的文本范围。主要选择可以以不同颜色显示或以不同样式的插入符号显示。只有已存在的选择才能成为主要选择。

SCI_SETSELECTIONNCARET (int selection, int caret)
SCI_GETSELECTIONNCARET (int selection) →位置
SCI_SETSELECTIONNCARETVIRTUALSPACE (int selection, int space)
SCI_GETSELECTIONNCARETVIRTUALSPACE (int selection) →int
SCI_SETSELECTIONNANCHOR (int selection, int anchor)
SCI_GETSELECTIONNANCHOR (int selection) →位置
SCI_SETSELECTIONNANCHORVIRTUALSPACE (int selection, int space)
SCI_GETSELECTIONNANCHORVIRTUALSPACE (int selection) →int
设置或查询每个已存在选择的插入符号和锚点的虚拟空间的位置和数量。

SCI_SETSELECTIONNSTART (int selection, int anchor)
SCI_GETSELECTIONNSTART (int selection) →位置
SCI_SETSELECTIONNEND (int selection, int caret)
SCI_GETSELECTIONNEND (int selection) →position
设置或查询每个已存在选择的开始和结束位置。主要用于查询每个范围的文本。
选择参数从零开始。

SCI_SETRECTANGULARSELECTIONCARET (INT 脱字符号)
SCI_GETRECTANGULARSELECTIONCARET →位置
SCI_SETRECTANGULARSELECTIONCARETVIRTUALSPACE (INT 空间)
SCI_GETRECTANGULARSELECTIONCARETVIRTUALSPACE →INT
SCI_SETRECTANGULARSELECTIONANCHOR (INT 锚)
SCI_GETRECTANGULARSELECTIONANCHOR →位置
SCI_SETRECTANGULARSELECTIONANCHORVIRTUALSPACE (INT 空间)
SCI_GETRECTANGULARSELECTIONANCHORVIRTUALSPACE →INT
集或查询为矩形的插入符号和锚的位置和虚拟空间量选择。设置矩形选择后，
会将其分解为多个选项，每行一个。

SCI_SETADDITIONALSELALPHA (阿尔法阿尔法)
SCI_GETADDITIONALSELALPHA →INT
SCI_SETADDITIONALSELFORE (颜色前)
SCI_SETADDITIONALSELBACK (颜色背面)
修改的附加选择的外观，使他们能够从具有其外观与设置主选择来区分
SCI_SETSELALPHA， **SCI_GETSELALPHA**， **SCI_SETSELFORE**， 和 **SCI_SETSELBACK**。
SCI_SETADDITIONALSELFORE 并且 **SCI_SETADDITIONALSELBACK** 调用无效，直到
SCI_SETSELFORE 并 **SCI_SETSELBACK** 使用 useSelection * Color 值设置为 true 来调用。
后续调用 **SCI_SETSELFORE**， **SCI_SETSELBACK** 并将覆盖函数设置的值
SCI_SETADDITIONALSEL*。

SCI_SETADDITIONALCARETFORE (颜色前)
SCI_GETADDITIONALCARETFORE →颜色
SCI_SETADDITIONALCARETSBLINK (布尔 additionalCaretBlink)
SCI_GETADDITIONALCARETSBLINK →布尔
修改的额外插入记号的外观，使他们能够从具有其外观与设置主插入符进行区

分 `SCI_SETCARETFORE`， `SCI_GETCARETFORE`， `SCI_SETCARETPERIOD`， 和 `SCI_GETCARETPERIOD`。

SCI_SETADDITIONALCARETSVISIBLE (`bool additionalCaretVisible`)
SCI_GETADDITIONALCARETSVISIBLE→`bool`

确定是否显示其他插入符号（默认值 `true`）。

SCI_SWAPMAINANCHORCARET
SCI_ROTATESELECTION
SCI_MULTIPLESELECTADDNEXT
SCI_MULTIPLESELECTADDEACH

可以将这些命令分配给键，以便可以操作多个选择。 `SCI_SWAPMAINANCHORCARET` 将插入符号移动到主选择的另一端。 `SCI_ROTATESELECTION` 使下一个选择成为主要选择。

`SCI_MULTIPLESELECTADDNEXT` 将目标中主要选择的下一次出现添加到主要选项集中。如果当前选择为空，则选择插入符号周围的单词。使用当前 `searchFlags` 应用程序可以选择区分大小写和单词搜索选项。

`SCI_MULTIPLESELECTADDEACH` 类似 `SCI_MULTIPLESELECTADDNEXT` 但添加了多个匹配而不是一个。

滚动和自动滚动

`SCI_SETFIRSTVISIBLELINE(int displayLine)`
`SCI_GETFIRSTVISIBLELINE` → `int`
`SCI_SETXOFFSET(int xOffset)`
`SCI_GETXOFFSET` → `int`
`SCI_LINESCROLL(int columns, int lines)`
`SCI_SCROLLCARET`
`SCI_SCROLLRANGE(int secondary, int primary)`
`SCI_SETXCARETPOLICY(int caretPolicy, int caretSlop)`
`SCI_SETYCARETPOLICY(int caretPolicy, int caretSlop)`
`SCI_SETVISIBLEPOLICY(int visiblePolicy, int visibleSlop)`
`SCI_SETHSCROLLBAR(bool visible)`
`SCI_GETHSCROLLBAR` → `bool`
`SCI_SETVSCROLLBAR(bool visible)`
`SCI_GETVSCROLLBAR` → `bool`
`SCI_SETSCROLLWIDTH(int pixelWidth)`
`SCI_GETSCROLLWIDTH` → `int`
`SCI_SETSCROLLWIDTHTRACKING(bool tracking)`
`SCI_GETSCROLLWIDTHTRACKING` → `bool`
`SCI_SETENDATLASTLINE(bool endAtLastLine)`
`SCI_GETENDATLASTLINE` → `bool`

SCI_SETFIRSTVISIBLELINE (`int displayLine`)
SCI_GETFIRSTVISIBLELINE→`int`

这些消息检索并设置 Scintilla 视图中第一个可见行的行号。文档中的第一行编号为 0.该值是可见行而不是文档行。

SCI_SETXOFFSET (`int xOffset`)
SCI_GETXOFFSET→`int`

它 *xoffset* 是文本视图开头的水平滚动位置（以像素为单位）。值 0 是正常位置，第一个文本列在视图的左侧可见。

SCI_LINESCROLL (int columns, int lines)

这将尝试按您指定的列数和行数滚动显示。正线值增加了屏幕顶部的行号（即，就用户而言，它们向上移动文本），负线值则相反。

列度量是默认样式中空格的宽度。正值会增加视图左边缘的列（即，就用户而言，它们会向左移动文本）。负值反过来。

也可以看看： [SCI_SETXOFFSET](#)

SCI_SCROLLCARET

如果当前位置（如果没有选择时这是插入符号）不可见，则滚动视图以使其根据当前插入符号策略可见。

SCI_SCROLLRANGE (int secondary, int primary)

将参数位置及它们之间的范围滚动到视图中，优先考虑主要位置，然后是次要位置。行为类似于 [SCI_SCROLLCARET](#) 使用主要位置而不是插入符号。然后努力确保次要位置和其间的范围也是可见的。这可用于使搜索匹配可见。

SCI_SETXCARETPOLICY (int caretPolicy, int caretSlop)

SCI_SETYCARETPOLICY (int caretPolicy, int caretSlop)

这些设置了插入符号策略。的值 *caretPolicy* 是的组合 **CARET_SLOP**, **CARET_STRICT**, **CARET_JUMPS** 和 **CARET_EVEN**。

CARET_SLOP	如果设置，我们可以定义一个 slop 值: <i>caretSlop</i> 。此值定义了一个不需要的区域（UZ），其中插入符号是...不需要的。该区域定义为垂直边缘附近的像素数，以及水平边缘附近的多条线。通过使插入符远离边缘，可以在其上下文中看到它。这使得可以完全看到插入符号所在的标识符，并且可以看到当前行跟随它的一些行，这些行通常依赖于该行。
CARET_STRICT	如果设置，则 CARET_SLOP 严格执行策略设置。如果 <i>caretSlop</i> 没有设置，则插入符号在显示屏上居中，如果设置则不能进入 UZ <i>caretSlop</i> 。
CARET_JUMPS	如果设置，显示将更加积极地移动，以便在再次应用策略之前，插入符号可以在相同方向上移动更长时间。'3UZ'表示法用于表示 UZ 的大小三倍于距离边距的距离。
CARET_EVEN	如果没有设置，则左侧和底部 UZ 分别向上和向下 UZ 延伸，而不是具有对称的 UZ。这样，我们倾向于显示有用的信息：大多数代码所在的行的开头，以及插入符号后面的行，例如函数体。

泥 浆	严 格	跳 跃	甚 至	Caret 可以去保证金	达到极限（走出能见度 或进入 UZ）显示是.....
--------	--------	--------	--------	--------------	-------------------------------

泥浆	严格	跳跃	甚至	Caret 可以去保证金	达到极限（走出能见度或进入 UZ）显示是.....
0	0	0	0	是	移动到插入顶部/右侧
0	0	0	1	是	感动了一个位置
0	0	1	0	是	移动到插入顶部/右侧
0	0	1	1	是	以插入符为中心
0	1	-	0	Caret 始终位于显示屏的顶部/右侧	-
0	1	-	1	不，插入总是居中	-
1	0	0	0	是	移动把插入不对称的 UZ
1	0	0	1	是	移动把插入了 UZ
1	0	1	0	是	移动到插入顶部或右边缘 3UZ 的插入符号
1	0	1	1	是	转移到 3UZ 边缘的插入符号
1	1	-	0	Caret 总是在 UZ 的顶部/右边缘	-
1	1	0	1	不，离开 UZ	感动了一个位置
1	1	1	0	不，离开 UZ	转移到 3UZ 边缘的插入符号

SCI_SETVISIBLEPOLICY (int visiblePolicy, int visibleSlop)

这决定了 [SCI_ENSUREVISIBLEENFORCEPOLICY](#) 调用时如何确定垂直定位。它需要 [VISIBLE_SLOP](#) 和 [VISIBLE_STRICT](#) 策略参数的标志。它在操作上类似于 [SCI_SETYCARETPOLICY\(int caretPolicy, int caretSlop\)](#)。

SCI_SETHSCROLLBAR (bool 可见)

SCI_GETHSCROLLBAR→bool

只有在假定宽度需要时才显示水平滚动条。如果您不想看到它，请致电 [SCI_SETHSCROLLBAR\(0\)](#)。用于 [SCI_SETHSCROLLBAR\(1\)](#)再次启用它。
[SCI_GETHSCROLLBAR](#) 返回当前状态。默认状态是在需要时显示它。

另请参见：[SCI_SETSCROLLWIDTH](#)。

SCI_SETVSCROLLBAR (bool 可见)

SCI_GETVSCROLLBAR→bool

默认情况下，在需要时始终显示垂直滚动条。您可以选择隐藏或显示它
[SCI_SETVSCROLLBAR](#) 并获取当前状态 [SCI_GETVSCROLLBAR](#)。

也可以看看：[SCI_LINESCROLL](#)

SCI_SETSCROLLWIDTH (int pixelWidth)

SCI_GETSCROLLWIDTH→int

为了提高性能，Scintilla 不会测量文档的显示宽度来确定水平滚动条的属性。相

反，使用假定的宽度。这些消息设置并获取 Scintilla 假定的文档宽度（以像素为单位）。默认值为 2000。要确保可以滚动当前可见行的宽度，请使用 `SCI_SETSCROLLWIDTHTRACKING`

SCI_SETSCROLLWIDTHTRACKING (bool tracking)

SCI_GETSCROLLWIDTHTRACKING → bool

如果启用了滚动宽度跟踪，则调整滚动宽度以确保当前显示的所有行都可以完全滚动。此模式永远不会将滚动宽度调整为更窄。

SCI_SETENDATLASTLINE (bool endAtLastLine)

SCI_GETENDATLASTLINE → bool

SCI_SETENDATLASTLINE 设置滚动范围，以便最大滚动位置在视图底部有最后一行（默认）。将其设置为 **false** 允许在最后一行下方滚动一页。

白色空间

```
SCI_SETVIEWWS(int viewWS)
SCI_GETVIEWWS → int
SCI_SETWHITESPACEFORE(bool useSetting, colour fore)
SCI_SETWHITESPACEBACK(bool useSetting, colour back)
SCI_SETWHITESPACESIZE(int size)
SCI_GETWHITESPACESIZE → int
SCI_SETTABDRAWMODE(int tabDrawMode)
SCI_GETTABDRAWMODE → int
SCI_SETEXTRAASCENT(int extraAscent)
SCI_GETEXTRAASCENT → int
SCI_SETEXTRADESCENT(int extraDescent)
SCI_GETEXTRADESCENT → int
```

SCI_SETVIEWWS (int viewWS)

SCI_GETVIEWWS → int

可以使白色空间可见，这对于空白很重要的语言（如 Python）非常有用。空格字符显示为小的居中点和制表符作为指向右侧的光箭头。还有一些方法可以控制行尾字符的显示。这两条消息设置并获得空白区域显示模式。该 **viewWS** 论点可以是以下之一：

SCWS_INVISIBLE	0 正常显示模式，空白显示为空背景颜色。
SCWS_VISIBLEALWAYS	1 白色空格字符绘制为点和箭头，
SCWS_VISIBLEAFTERINDENT	2 用于缩进的空白区域通常显示，但在第一个可见字符后，它显示为点和箭头。
SCWS_VISIBLEONLYININDENT	3 用于缩进的空白区域显示为点和箭头。

使用任何其他 **viewWS** 值的效果未定义。

SCI_SETWHITESPACEFORE (bool useSetting, color fore)

SCI_SETWHITESPACEBACK (bool useSetting, color back)

默认情况下，可见空白的颜色由使用的词法分析器决定。可以全局设置所有可

见空白区域的前景色和/或背景色，用 **SCI_SETWHITESPACEFORE** 和 覆盖词法分析器的颜色 **SCI_SETWHITESPACEBACK**。

SCI_SETWHITESPACESIZE (int size)

SCI_GETWHITESPACESIZE→int

SCI_SETWHITESPACESIZE 设置用于标记空格字符的点的的大小。该

SCI_GETWHITESPACESIZE 消息检索当前大小。

SCI_SETTABDRAWMODE (int tabDrawMode)

SCI_GETTABDRAWMODE→int

这两条消息获取并设置当空白区域可见时如何绘制制表符。该 *tabDrawMode* 论点可以是以下之一：

SCTD_LONGARROW	0 箭头的默认模式一直延伸到 tabstop。
-----------------------	-------------------------

SCTD_STRIKEOUT	1 水平线延伸到 tabstop。
-----------------------	-------------------

使用任何其他 *tabDrawMode* 值的效果未定义。

SCI_SETEXTRAASCENT (int extraAscent)

SCI_GETEXTRAASCENT→int

SCI_SETEXTRADESCENT (int extraDescent)

SCI_GETEXTRADESCENT→int

使用“基线”上每个字符的基础绘制文本。线条的高度是从任何样式在基线上方延伸的最大值（其“上升”）中找到的，添加到任何样式在基线下方延伸的最大值（其“下降”）。可以将空间添加到最大上升（**SCI_SETEXTRAASCENT**）和最大下降（**SCI_SETEXTRADESCENT**）以允许线之间的更多空间。这可以使文本更容易阅读或容纳下划线或突出显示。

光标

SCI_SETCURSORS (int cursorType)

SCI_GETCOURSORS→int

SCI_SETCURSORS (int cursorType)

SCI_GETCOURSORS→int

光标通常以上下文敏感的方式选择，因此它在边缘上的颜色与在文本上的不同。执行慢动作时，您可能希望更改为等待光标。您可以使用设置光标类型

SCI_SETCURSORS。该 *cursorType* 参数可以是：

SC_CURSORNORMAL	-1 显示正常光标。
------------------------	------------

SC_CURSORWAIT	4 当鼠标悬停或由 Scintilla 窗口拥有时，将显示等待光标。
----------------------	------------------------------------

光标值 1 到 7 具有已定义的光标，但只能 **SC_CURSORWAIT** 有效控制。其他值 *cursorType* 导致指针显示。该 **SCI_GETCOURSORS** 消息返回您设置的最后一个光标类型，或者 **SC_CURSORNORMAL** (-1) 如果您尚未设置光标类型。

鼠标捕获

SCI_SETMOUSEDOWNCAPTURES (bool capture)
SCI_GETMOUSEDOWNCAPTURES→bool
SCI_SETMOUSEWHEELCAPTURES (bool capture)
SCI_GETMOUSEWHEELCAPTURES→bool

SCI_SETMOUSEDOWNCAPTURES (bool 捕获)
SCI_GETMOUSEDOWNCAPTURES→bool

当在 Scintilla 内部按下鼠标时，它被捕获，因此将来的鼠标移动事件发送到 Scintilla。可以关闭此行为 **SCI_SETMOUSEDOWNCAPTURES(0)**。

SCI_SETMOUSEWHEELCAPTURES (bool 捕获)
SCI_GETMOUSEWHEELCAPTURES→bool

在 Windows 上 **WM_MOUSEWHEEL**，即使鼠标指针远离 Scintilla 编辑器窗口，Scintilla 也会捕获所有消息（如果它具有焦点）。可以更改此行为，**SCI_SETMOUSEWHEELCAPTURES(0)** 以便 Scintilla 将 **WM_MOUSEWHEEL** 消息传递到其父窗口。如果鼠标指针位于编辑器窗口上，Scintilla 仍会对鼠标滚轮作出反应。

行结尾

Scintilla 可以处理主要的行结束约定，并且根据设置和当前词法分析器还支持其他 Unicode 行结束。

Scintilla 可以解释任何 Macintosh (\ r)，Unix (\ n) 和 Windows (\ r \ n) 行结束。当用户按下 Enter 键时，其中一个行结束字符串将插入缓冲区。Windows 中的默认值为 \ r \ n，Unix 中的默认值为 \ n，但可以使用该 **SCI_SETEOLMODE** 消息进行更改。您还可以将整个文档转换为其中一个行结尾 **SCI_CONVERTEOLS**。最后，您可以选择显示行结尾 **SCI_SETVIEWEOL**。

对于 UTF-8 编码，当打开 Unicode 行结束并且当前词法分析器也支持 Unicode 行结束时，可以选择解释另外三个 Unicode 行结束，Next Line (**NEL=U+0085**)，Line Separator (**LS=U+2028**) 和 Paragraph Separator (**PS=U+2029**)。

SCI_SETEOLMODE (int eolMode)
SCI_GETEOLMODE→int
SCI_CONVERTEOLS (int eolMode)
SCI_SETVIEWEOL (bool visible)
SCI_GETVIEWEOL→bool
SCI_GETLINEENDTYPESSUPPORTED→int
SCI_SETLINEENDTYPESALLOWED (int lineEndBitSet)
SCI_GETLINEENDTYPESALLOWED→int
SCI_GETLINEENDTYPESACTIVE→int

SCI_SETEOLMODE (int eolMode)

SCI_GETEOLMODE→int

SCI_SETEOLMODE 设置用户按下 Enter 键时添加到文档中的字符。您可以设置 *eolMode* 为 **SC_EOL_CRLF** (0)，**SC_EOL_CR** (1) 或 **SC_EOL_LF** (2) 之一。该 **SCI_GETEOLMODE** 消息检索当前状态。

SCI_CONVERTEOLS (int eolMode)

此消息更改文档中所有行尾字符以匹配 *eolMode*。有效值为：**SC_EOL_CRLF** (0)，**SC_EOL_CR** (1) 或 **SC_EOL_LF** (2)。

SCI_SETVIEWEOL (bool 可见)

SCI_GETVIEWEOL→bool

通常，行尾字符是隐藏的，但 **SCI_SETVIEWEOL** 允许您通过设置 *visible true* (或 *false*) 来显示 (或隐藏) 它们。行的字符的端部的可见呈现类似于 **(CR)**，**(LF)** 或 **(CR)(LF)**。**SCI_GETVIEWEOL** 返回当前状态。

SCI_GETLINEENDTYPESSUPPORTED→int

SCI_GETLINEENDTYPESSUPPORTED 报告当前词法分析器支持的不同类型的行结尾。虽然目前只有一个选择 **SC_LINE_END_TYPE_DEFAULT** (0) 或 **SC_LINE_END_TYPE_UNICODE** (1)，但这有点设置。这些值也被与 Unicode 行结束有关的其他消息使用。

SCI_SETLINEENDTYPESALLOWED (int lineEndBitSet)

SCI_GETLINEENDTYPESALLOWED→int

默认情况下，只解释 ASCII 行结束。可以请求 Unicode 行结束，**SCI_SETLINEENDTYPESALLOWED(SC_LINE_END_TYPE_UNICODE)** 但除非 *lexer* 还允许 Unicode 行结束，否则这将无效。**SCI_GETLINEENDTYPESALLOWED** 返回当前状态。

SCI_GETLINEENDTYPESACTIVE→int

SCI_GETLINEENDTYPESACTIVE 报告当前由 Scintilla 解释的行结束集。是的 **SCI_GETLINEENDTYPESSUPPORTED** & **SCI_GETLINEENDTYPESALLOWED**。

话

支持选择，导航和搜索单词。

单词是来自特定字符集连续字符序列。4 个类别定义单词：单词，空格，标点符号和行结尾，每个类别在单词函数中起作用。双击选择该点的单词，可以是单词，标点符号或空格字节的序列。双击不会选择行结束，但会充当单词分隔符。

单词是根据字符定义的，每个类别中的字符集可以在一定程度上进行自定义。**NUL** 字符 (0) 始终是空格，因为设置类别的 API 使用 **NUL** 终止的字符串。对于单字节编码，可以将类别分配给任何字符 (1 到 0xFF)。对于多字节编码，可以将类别分配给 1 到 0x7F 的字符，静态行为来自 0x80。对于 UTF-8，0x80

中的字符将使用基于其 Unicode 常规类别的类别。对于亚洲编码，代码页 932,936,949,950 和 1361，来自 0x80 的字符被视为字符。

编程语言中的标识符通常是带有大写的单词序列（aCamelCaseIdentifier）或用于标记单词边界的下划线（an_under_bar_ident）。该 SCI_WORDPART*命令用于字部位之间移动：SCI_WORDPARTLEFT，SCI_WORDPARTLEFTTEXTEND，SCI_WORDPARTRIGHT，和 SCI_WORDPARTRIGHTTEXTEND。

```
SCI_WORDENDPOSITION (int pos, bool onlyWordCharacters) →int
SCI_WORDSTARTPOSITION (int pos, bool onlyWordCharacters) →int
SCI_ISRANGWORD (int start, int end) →bool
SCI_SETWORDCHARS (<unused>, const char * characters)
SCI_GETWORDCHARS (<unused>, char * characters ) →int
SCI_SETWHITESPACECHARS (<unused>, const char * characters)
SCI_GETWHITESPACECHARS (<unused>, char * characters) →int
SCI_SETPUNCTUATIONCHARS (<unused>, const char * characters)
SCI_GETPUNCTUATIONCHARS (<unused>, char * characters) →int
SCI_SETCHARSDEFAULT
```

SCI_WORDENDPOSITION (int pos, bool onlyWordCharacters) →int
SCI_WORDSTARTPOSITION (int pos, bool onlyWordCharacters) →int
这些消息使用与 Scintilla 内部使用的相同的单词定义返回单词的开头和结尾。您可以设置自己的字符列表，这些字符计为单词 SCI_SETWORDCHARS。位置设置开始或搜索，在搜索结束时向前搜索，向搜索开始时向后搜索。

SCI_ISRANGWORD (int start, int end) →bool
范围是 start..end 是一个单词还是一组单词？此消息检查开始是否处于字开始转换，并且该结束处于字结束转换。它不会检查范围内是否有任何空格。

```
SCI_ISRANGWORD (int start, int end) →bool
```

设置 onlyWordCharacters 为 true (1) 以停止搜索搜索方向上的第一个非单词字符。如果 onlyWordCharacters 是 false (0)，则搜索方向上的第一个字符将搜索类型设置为单词或非单词，并且搜索在第一个不匹配字符处停止。搜索也由文档的开头或结尾终止。

如果“w”代表单词字符和“。”表示非单词字符和“|”表示位置和/ true 或 false 状态 onlyWordCharacters:

初始状态	结束，真的	结束，假	开始，真的	开始，假
..wwwwwwwwww WWwwwwww WW ..
.... WW WW WWWW WWWW WWWW WWWW
..ww ww ww WW WW

初始状态	结束，真的	结束，假	开始，真的	开始，假
WW ..	WW ..	WW ..	WW ..	WW ..
..wwww WW	..ww WW	..ww WWww WW ..
WW		

SCI_SETWORDCHARS (<unused>, const char * characters)

此消息定义哪些字符是单词类别的成员。在处理此功能之前，字符类别设置为默认值。例如，如果您在字符集中不允许使用“_”，请使用：

```
SCI_SETWORDCHARS(0,
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789");
```

SCI_GETWORDCHARS (<unused>, char * characters) →int

这将使用单词中包含的所有字符填充 characters 参数。characters 参数必须足够大才能容纳所有字符。如果 characters 参数为 0，则返回应分配用于存储整个集合的长度。

对于多字节编码，此 API 不会返回 0x80 及更高版本的有意义值。

SCI_SETWHITESPACECHARS (<unused>, const char * characters)

SCI_GETWHITESPACECHARS (<unused>, char * characters) →int

类似于 SCI_SETWORDCHARS，此消息允许用户定义 Scintilla 认为哪些字符为空格。设置空白字符允许用户微调 Scintilla 的行为，例如将光标移动到单词的开头或结尾；例如，通过将标点符号字符定义为空格，当用户按下 ctrl + left 或 ctrl + right 时，它们将被跳过。应该调用此函数，SCI_SETWORDCHARS 因为它会将空白字符重置为默认设置。SCI_GETWHITESPACECHARS 行为与...相似 SCI_GETWORDCHARS。

SCI_SETPUNCTUATIONCHARS (<unused>, const char * characters)

SCI_GETPUNCTUATIONCHARS (<unused>, char * characters) →int

类似于 SCI_SETWORDCHARS 和 SCI_SETWHITESPACECHARS，此消息允许用户定义 Scintilla 认为标点符号的字符。SCI_GETPUNCTUATIONCHARS 行为与...相似 SCI_GETWORDCHARS。

SCI_SETCHARSDEFAULT

使用默认的单词和空白字符集。这将空格设置为空格，制表符和其他字符，代码小于 0x20，单词字符设置为字母数字和“_”。

Word 键盘命令是：

- SCI_WORDLEFT
- SCI_WORDLEFTTEXTEND
- SCI_WORDRIGHT
- SCI_WORDRIGHTTEXTEND
- SCI_WORDLEFTTEND
- SCI_WORDLEFTTENDEXTEND
- SCI_WORDRIGHTTEND
- SCI_WORDRIGHTTENDEXTEND

- **SCI_WORDPARTLEFT**
- **SCI_WORDPARTLEFTTEXTEND**
- **SCI_WORDPARTRIGHT**
- **SCI_WORDPARTRIGHTTEXTEND**
- **SCI_DELWORDLEFT**
- **SCI_DELWORDRIGHT**
- **SCI_DELWORDRIGHTEND**

造型

样式消息允许您为文本指定样式。如果某个标准词法分析器可以满足您的样式需求，或者您可以自己编写样式，那么词法分析器可能是设置文档样式的最简单方法。如果您选择使用容器进行样式设置，则可以使用该 **SCI_SETLEXER** 命令进行选择 **SCLEX_CONTAINER**，在这种情况下，每次文本需要样式进行显示时，都会向容器发送通知。作为另一种选择，您可以使用空闲时间来设置文档样式。即使您使用词法分析器，也可以使用样式命令来标记编译器检测到的错误。可以使用以下命令。 **SCN_STYLENEEDED**

```
SCI_GETENDSTYLED → position
SCI_STARTSTYLING(int start, int unused)
SCI_SETSTYLING(int length, int style)
SCI_SETSTYLINGEX(int length, const char *styles)
SCI_SETIDLESTYLING(int idleStyling)
SCI_GETIDLESTYLING → int
SCI_SETLINESTATE(int line, int state)
SCI_GETLINESTATE(int line) → int
SCI_GETMAXLINESTATE → int
```

SCI_GETENDSTYLED→位置

Scintilla 会记录可能正确设置样式的最后一个字符。如果对其后面的字符进行样式设置并向后移动，如果对其前面的文档进行了更改，则会向前移动。在绘制文本之前，检查此位置以查看是否需要任何样式，如果需要，则

SCN_STYLENEEDED 向容器发送通知消息。容器可以发送 **SCI_GETENDSTYLED** 到需要开始样式的位置。Scintilla 总是会要求整行风格。

SCI_STARTSTYLING (int start, int unused)

通过将样式位置设置 *start* 为开始来准备样式。未使用的参数在早期版本中使用，但现在被忽略。之后 **SCI_STARTSTYLING**，**SCI_SETSTYLING** 为每个词法实体发送多个消息以设置样式或发送 **SCI_SETSTYLINGEX** 到块中的样式。

SCI_SETSTYLING (int length, int style)

此消息设置 *length* 从样式位置开始的字符样式，然后增加样式位置 *length*，为下一个调用做好准备。 **SCI_STARTSTYLING** 应该在第一次调用之前调用。

SCI_SETSTYLINGEX (int length, const char * styles)

作为替代 **SCI_SETSTYLING**，它将相同的样式应用于每个字节，您可以使用此消

息指定 *Length* 样式位置中每个字节的样式，然后增加样式位置 *Length*，准备好为下次通话。 **SCI_STARTSTYLING** 应该在第一次调用之前调用。

SCI_SETIDLESTYLING (int idleStyling)

SCI_GETIDLESTYLING→int

默认情况下，**SC_IDLESTYLING_NONE** (0)，在显示之前对所有当前可见的文本执行语法样式设置。在非常大的文件上，这可能会使向下滚动速度变慢。对于 **SC_IDLESTYLING_TOVISIBLE** (1)，在显示之前执行少量样式化，然后在后台逐步执行进一步的样式作为空闲时间任务。这可能导致文本最初显示为无色，然后，一段时间后，它会被着色。当前可见部分之后的文本可以在背景中用 **SC_IDLESTYLING_AFTERVERISIBLE** (2) 设置样式。要在背景中的可见文本之前和之后设置样式，请使用 **SC_IDLESTYLING_ALL** (3)。

由于包装还需要执行样式并且还使用空闲时间，因此当文档显示为包装时，此设置无效。

SCI_SETLINESTATE (int line, int state)

SCI_GETLINESTATE (int line) →int

除了为每个字符存储的 8 位词法状态外，还为每一行存储了一个整数。这可以用于更长寿的解析状态，例如当前脚本语言在 ASP 页面中的含义。使用 **SCI_SETLINESTATE** 设置整数值，**SCI_GETLINESTATE** 来获取值。更改该值会生成 **SC_MOD_CHANGELINESTATE** 通知。

SCI_GETMAXLINESTATE→int

返回具有任何行状态的最后一行。

风格定义

虽然上面提到的样式设置消息会更改与文本关联的样式编号，但这些消息定义了如何直观地解释这些样式编号。有 256 种 **lexer** 样式可以设置，编号为 0 到 **STYLE_MAX** (255)。还有一些从 32 开始的预定义编号样式，**STYLE_**定义了以下*常量。

STYLE_DEFAULT	32	此样式定义了 SCI_STYLECLEARALL 使用消息时所有样式都接收的属性。
STYLE_LINENUMBER	33	此样式设置用于在行号边距中显示行号的文本的属性。为此样式设置的背景颜色还为未设置任何折叠蒙版位的所有边距设置背景颜色。也就是说，任何边距为 mask & SC_MASK_FOLDERS0 。 SCI_SETMARGINMASKN 有关蒙版的更多信息，请参阅。
STYLE_BRACELIGHT	34	此样式设置在使用 SCI_BRACEHIGHLIGHT 消息突出显示大括号时以及使用突出显示相应缩进时使用的属性 SCI_SETHIGHLIGHTGUIDE 。

STYLE_BRACEBAD	35	此样式设置在使用 SCI_BRACEBADLIGHT 消息标记不匹配的大括号时使用的显示属性。
STYLE_CONTROLCHAR	36	此样式设置绘制控制字符时使用的字体。仅使用字体，大小，粗体，斜体和字符集属性，而不使用颜色属性。另见： SCI_SETCONTROLCHARSYMBOL 。
STYLE_INDENTGUIDE	37	此样式设置绘制缩进指南时使用的前景色和背景色。
STYLE_CALLTIP	38	通话提示通常使用由其定义的字体属性 STYLE_DEFAULT 。使用 SCI_CALLTIPUSESTYLE 原因调用提示来改为使用此样式。仅使用字体名称，字体大小，前景色和背景色以及字符集属性。
STYLE_FOLDDISPLAYTEXT	39	这是用于绘制附加到折叠文本的文本标签的样式。
STYLE_LASTPREDEFINED	39	为了使客户端代码更容易发现预定义的样式范围，将其设置为上次预定义样式的样式编号。
STYLE_MAX	255	这不是样式，而是可以设置的最大样式的数量。可以使用 STYLE_LASTPREDEFINED 和之间的样式 STYLE_MAX 。

对于每种样式，您可以设置字体名称，大小和粗体，斜体和下划线，前景和背景颜色以及字符集的使用。您还可以选择隐藏具有给定样式的文本，将所有字符显示为大写或小写，并从行中的最后一个字符填充到行的末尾（对于嵌入式语言）。还有一个实验属性可以将文本设置为只读。

完全取决于您如何使用样式。如果要使用语法着色，可以使用样式 0 表示空格，样式 1 表示数字，样式 2 表示关键字，样式 3 表示字符串，样式 4 表示预处理器，样式 5 表示操作符，等等。

```

SCI_STYLERESETDEFAULT
SCI_STYLECLEARALL
SCI_STYLESETFONT(int style, const char *fontName)
SCI_STYLEGETFONT(int style, char *fontName) → int
SCI_STYLESETSIZE(int style, int sizePoints)
SCI_STYLEGETSIZE(int style) → int
SCI_STYLESETSIZEFRACTIONAL(int style, int sizeHundredthPoints)
SCI_STYLEGETSIZEFRACTIONAL(int style) → int
SCI_STYLESETBOLD(int style, bool bold)
SCI_STYLEGETBOLD(int style) → bool
SCI_STYLESETWEIGHT(int style, int weight)
SCI_STYLEGETWEIGHT(int style) → int
SCI_STYLESETITALIC(int style, bool italic)
SCI_STYLEGETITALIC(int style) → bool
SCI_STYLESETUNDERLINE(int style, bool underline)
SCI_STYLEGETUNDERLINE(int style) → bool
SCI_STYLESETFORE(int style, colour fore)
SCI_STYLEGETFORE(int style) → colour
SCI_STYLESETBACK(int style, colour back)
SCI_STYLEGETBACK(int style) → colour

```

```

SCI_STYLESETEOLFILLED(int style, bool eolFilled)
SCI_STYLEGETEOLFILLED(int style) → bool
SCI_STYLESETCHARACTERSET(int style, int characterSet)
SCI_STYLEGETCHARACTERSET(int style) → int
SCI_STYLESETCASE(int style, int caseVisible)
SCI_STYLEGETCASE(int style) → int
SCI_STYLESETVISIBLE(int style, bool visible)
SCI_STYLEGETVISIBLE(int style) → bool
SCI_STYLESETCHANGEABLE(int style, bool changeable)
SCI_STYLEGETCHANGEABLE(int style) → bool
SCI_STYLESETHOTSPOT(int style, bool hotspot)
SCI_STYLEGETHOTSPOT(int style) → bool

```

SCI_STYLERESETDEFAULT 当 Scintilla 初始化时，此消息重置 **STYLE_DEFAULT** 为其状态。

SCI_STYLECLEARALL

此消息将所有样式设置为具有相同的属性 **STYLE_DEFAULT**。如果您正在设置 Scintilla 进行语法着色，则您设置的词法样式很可能非常相似。设置样式的一种方法是：

1. 设置 **STYLE_DEFAULT** 所有样式的常用功能。
2. 用于 **SCI_STYLECLEARALL** 将此复制到所有样式。
3. 设置使词法样式不同的样式属性。

```

SCI_STYLESETFONT (int style, const char * fontName)
SCI_STYLEGETFONT (int style, char * fontName NUL-terminated) →int
SCI_STYLESETSIZE (int style, int sizePoints)
SCI_STYLEGETSIZE (int style) →int
SCI_STYLESETSIZEFRACTIONAL (int style, int sizeHundredthPoints)
SCI_STYLEGETSIZEFRACTIONAL (int 样式) →int
SCI_STYLESETBOLD (int style, bool bold)
SCI_STYLEGETBOLD (int style) →bool
SCI_STYLESETWEIGHT (int style, int weight)
SCI_STYLEGETWEIGHT (int style) →int
SCI_STYLESETITALIC (int style, bool italic)
SCI_STYLEGETITALIC (int style) →bool

```

这些 messages (plus **SCI_STYLESETCHARACTERSET**) 设置用于将您请求的字体与可用字体匹配的字体属性。

这 *fontName* 是一个包含字体名称的零终止字符串。在 Windows 下，仅使用名称的前 32 个字符，名称解码为 UTF-8，名称不区分大小写。对于内部缓存，Scintilla 按名称跟踪字体，并且关心字体名称的大小写，因此请保持一致。在 GTK + 上，Pango 用于显示文本，名称直接发送到 Pango 而不进行转换。在 Qt 上，名称被解码为 UTF-8。在 Cocoa 上，名称被解码为 MacRoman。

通过将大小乘以 100 ()，可以将大小设置为整数个点，**SCI_STYLESETSIZE** 或者使用小数点大小，以百分之一点为 **SCI_STYLESETSIZEFRACTIONAL** 单位

SC_FONT_SIZE_MULTIPLIER。例如，设置了 9.4 点的文本大小
SCI_STYLESETSIZEFRACTIONAL(<style>, 940)。

可以使用 **SCI_STYLESETBOLD** 或设置字体的粗细或粗体 **SCI_STYLESETWEIGHT**。重量是 1 到 999 之间的数字，1 表示非常轻，999 表示非常重。虽然可以使用任何值，但字体通常仅支持 2 到 4 个权重，其中三个权重通常足以具有符号名称：
SC_WEIGHT_NORMAL (400)，**SC_WEIGHT_SEMIBOLD** (600) 和 **SC_WEIGHT_BOLD** (700)。该 **SCI_STYLESETBOLD** 消息采用布尔参数，0 选择 **SC_WEIGHT_NORMAL** 和 1 **SC_WEIGHT_BOLD**。

SCI_STYLESETUNDERLINE (int style, bool underline)

SCI_STYLEGETUNDERLINE (int style) →bool

您可以设置要加下划线的样式。下划线以前景色绘制。具有包含下划线属性的样式的所有字符都带有下划线，即使它们是空格。

SCI_STYLESETFORE (int style, color fore) SCI_STYLEGETFORE (int style) →color

SCI_STYLESETBACK (int style, color back)

SCI_STYLEGETBACK (int style) →color

文本以前景色绘制。未被角色占据的每个角色单元格中的空间以背景颜色绘制。

SCI_STYLESETEOLFILLED (int style, bool eolFilled)

SCI_STYLEGETEOLFILLED (int style) →bool

如果行中的最后一个字符具有设置了此属性的样式，则直到窗口右边缘的行的其余部分将填充背景颜色集为了最后一个角色。当文档包含其他语言（如带有嵌入式 JavaScript 的 HTML 页面）的嵌入式部分时，这非常有用。通过设置 *eolFilled* 到 **true** 和一致的背景颜色（从背景颜色的 HTML 样式设置不同），以所有的 JavaScript 风格则 JavaScript 的部分将会从 HTML 容易辨别。

SCI_STYLESETCHARACTERSET (int style, int characterSet)

SCI_STYLEGETCHARACTERSET (int style) →int

您可以将样式设置为使用与默认字符集不同的字符集。这些字符集可能有用的地方是注释和文字字符串。例如，**SCI_STYLESETCHARACTERSET(SCE_C_STRING, SC_CHARSET_RUSSIAN)** 确保俄语中的字符串能够在 C 和 C ++ 中正确显示（**SCE_C_STRING** 是 C 和 C ++ 词法分析器用来显示文字字符串的样式编号;它的值为 6）。此功能在 Windows 和 GTK + 上的工作方式不同。
默认字符集是 **SC_CHARSET_DEFAULT**。

SC_CHARSET_ANSI 并且 **SC_CHARSET_DEFAULT**，除非代码页设置指定的 Windows 欧洲代码页 1252。

字符集	视窗	GTK +	可可
SC_CHARSET_ANSI	✓	✓	✓ (8859-1)
SC_CHARSET_ARABIC	✓		✓

SC_CHARSET_BALTIC	✓		✓
SC_CHARSET_CHINESEBIG5	✓		✓
SC_CHARSET_DEFAULT	✓	✓ (8859-1)	✓ (8859-1)
SC_CHARSET_EASTEUROPE	✓	✓	✓
SC_CHARSET_GB2312	✓	✓	✓
SC_CHARSET_GREEK	✓		✓
SC_CHARSET_HANGUL	✓	✓	✓
SC_CHARSET_HEBREW	✓		✓
SC_CHARSET_JOHAB	✓		✓
SC_CHARSET_MAC	✓		✓
SC_CHARSET_OEM	✓		✓
SC_CHARSET_RUSSIAN	✓ (cp1251)	✓ (koi8-r)	✓ (cp1251)
SC_CHARSET_SHIFTJIS	✓	✓	✓
SC_CHARSET_SYMBOL	✓		✓
SC_CHARSET_THAI	✓		✓
SC_CHARSET_TURKISH	✓		✓
SC_CHARSET_VIETNAMESE	✓		✓
SC_CHARSET_OEM866		✓ (cp866)	
SC_CHARSET_CYRILLIC		✓ (cp1251)	✓ (cp1251)
SC_CHARSET_8859_15		✓	✓

SCI_STYLESETCASE (int style, int caseVisible)

SCI_STYLEGETCASE (int style) →int

值 **caseVisible** 确定文本的显示方式。您可以设置大写 (**SC_CASE_UPPER**, 1) 或小写 (**SC_CASE_LOWER**, 2) 或驼峰大小写 (**SC_CASE_CAMEL**, 3) 或正常显示 (**SC_CASE_MIXED**, 0)。这不会更改存储的文本，只会更改它的显示方式。

SCI_STYLESETVISIBLE (int style, bool visible)

SCI_STYLEGETVISIBLE (int style) →bool

Text 通常可见。但是，您可以通过将其 **visible** 设置为 0 来完全隐藏它。这可以用于隐藏 HTML 或 XML 中的嵌入格式说明或超文本关键字。用户操作不能删除不可见文本，但应用程序可能通过调用 **SCI_DELETERANGE** 删除不可见文本。

SCI_STYLESETCHANGEABLE (int style, bool changeable)

SCI_STYLEGETCHANGEABLE (int style) →bool

这是一个实验性且未完全实现的样式属性。默认设置已 **changeable** 设置，**true** 但设置后，**false** 它将文本设置为只读。用户不能在不可更改的文本内移动插入符号，并且用户不能删除不可更改的文本。应用程序可以通过调用 **SCI_DELETERANGE** 删除不可更改的文本。

SCI_STYLESETHOTSPOT (int style, bool hotspot)

SCI_STYLEGETHOTSPOT (int style) → bool

此样式用于标记可以检测鼠标单击的文本范围。光标变为切换热点，前景和背景颜色可能会更改，并且下划线似乎表示这些区域对点击敏感。这可用于允许到其他文档的超链接。

插入符号，选择和热点样式

通过更改前景色和/或背景色来显示选择。如果未设置其中一个，则不会更改该属性以进行选择。默认设置是通过将背景更改为浅灰色并使前景与未选择时相同来显示选择。如果没有选择，则当前插入点由文本插入符标记。这是一条垂直线，通常会闪烁，以吸引用户注意。

```
SCI_SETSELEFORE(bool useSetting, colour fore)
SCI_SETSELBACK(bool useSetting, colour back)
SCI_SETSELALPHA(alpha alpha)
SCI_GETSELALPHA → int
SCI_SETSELEOLFILLED(bool filled)
SCI_GETSELEOLFILLED → bool
SCI_SETCARETFORE(colour fore)
SCI_GETCARETFORE → colour
SCI_SETCARETLINEVISIBLE(bool show)
SCI_GETCARETLINEVISIBLE → bool
SCI_SETCARETLINEBACK(colour back)
SCI_GETCARETLINEBACK → colour
SCI_SETCARETLINEBACKALPHA(alpha alpha)
SCI_GETCARETLINEBACKALPHA → int
SCI_SETCARETLINEFRAME(int width)
SCI_GETCARETLINEFRAME → int
SCI_SETCARETLINEVISIBLEALWAYS(bool alwaysVisible)
SCI_GETCARETLINEVISIBLEALWAYS → bool
SCI_SETCARETPERIOD(int periodMilliseconds)
SCI_GETCARETPERIOD → int
SCI_SETCARETSTYLE(int caretStyle)
SCI_GETCARETSTYLE → int
SCI_SETCARETWIDTH(int pixelWidth)
SCI_GETCARETWIDTH → int
SCI_SETHOTSPOTACTIVEFORE(bool useSetting, colour fore)
SCI_GETHOTSPOTACTIVEFORE → colour
SCI_SETHOTSPOTACTIVEBACK(bool useSetting, colour back)
SCI_GETHOTSPOTACTIVEBACK → colour
SCI_SETHOTSPOTACTIVEUNDERLINE(bool underline)
SCI_GETHOTSPOTACTIVEUNDERLINE → bool
SCI_SETHOTSPOTSINGLELINE(bool singleLine)
SCI_GETHOTSPOTSINGLELINE → bool
SCI_SETCARETSTICKY(int useCaretStickyBehaviour)
SCI_GETCARETSTICKY → int
SCI_TOGGLECARETSTICKY
```

SCI_SETSELEFORE (bool useSetting, color fore)

SCI_SETSELBACK (bool useSetting, color back)

您可以选择使用这两条消息覆盖默认选择着色。如果你设置你所提供的颜色用

于 *useSelection*Colour* 给 **true**。如果设置为 **false**，则使用默认的样式着色，*fore* 或者 *back* 参数无效。

SCI_SETSELALPHA (**alpha** alpha)

SCI_GETSELALPHA→int

通过设置 alpha 值，可以在选择背景颜色中半透明地绘制选区。

SCI_SETSELEOLFILLED (**bool** 填充)

SCI_GETSELEOLFILLED→bool

通过设置此属性，可以将选择绘制到右侧边框。

SCI_SETCARETFORE (**颜色** 前)

SCI_GETCARETFORE→颜色

插入符号的颜色可以设置 **SCI_SETCARETFORE** 和检索 **SCI_GETCARETFORE**。

SCI_SETCARETLINEVISIBLE (**bool** show)

SCI_GETCARETLINEVISIBLE→bool

SCI_SETCARETLINEBACK (**颜色** 返回)

SCI_GETCARETLINEBACK→颜色

SCI_SETCARETLINEBACKALPHA (**alpha** alpha)

SCI_ETETCARETLINEBACKALPHA→int

SCI_SETCARETLINEFRAME (**int** width)

SCI_GETCARETLINEFRAME→int

您可以选择使包含插入符号的行的背景颜色与这些消息不同。为此，请使用，设置所需的背景颜色 **SCI_SETCARETLINEBACK**，然后使用

SCI_SETCARETLINEVISIBLE(true)以启用效果。您可以使用取消效果

SCI_SETCARETLINEVISIBLE(false)。他们俩 **SCI_GETCARET***函数返回状态和颜色。

当一条线具有否则会改变背景颜色的标记时，这种形式的背景着色具有最高优先级。插入符号线也可以半透明地绘制，这允许透过其他背景颜色。这是通过调用 **SCI_SETCARETLINEBACKALPHA** 设置 alpha（半透明）值来完成的。当 alpha 不是 **SC_ALPHA_NOALPHA** 时，插入符号行将在所有其他要素之后绘制，因此将影响所有其他要素的颜色。或者 **SCI_SETCARETLINEFRAME** 可以用于显示框架的插入线而不是填充整个背景。设置宽度 != 0 以启用此选项，设置宽度= 0 以禁用它。

SCI_SETCARETLINEVISIBLEALWAYS (**bool** alwaysVisible)

SCI_GETCARETLINEVISIBLEALWAYS→bool

选择即使窗口未对焦，也始终可以看到插入线。

SCI_SETCARETLINEVISIBLEALWAYS(false)插入符号行仅在窗口处于焦点时可见的默认行为。

SCI_SETCARETPERIOD (**int** periodMilliseconds)

SCI_GETCARETPERIOD→int

可以设置插入符闪烁的速率，用于 **SCI_SETCARETPERIOD** 确定插入符号在更改状态之前可见或不可见的时间（以毫秒为单位）。将句点设置为 0 会停止插入符号闪烁。默认值为 500 毫秒。 **SCI_GETCARETPERIOD** 返回当前设置。

SCI_SETCARETSTYLE (int caretStyle)

SCI_GETCARETSTYLE→int

插入符号的样式可以设置 **SCI_SETCARETSTYLE** 为行符号 (**CARETSTYLE_LINE** = 1)，块插入符 (**CARETSTYLE_BLOCK** = 2) 或根本不绘制

(**CARETSTYLE_INVISIBLE** = 0)。默认值为线条插入符号

(**CARETSTYLE_LINE** = 1)。您可以使用 **SCI_SETCARETSTYLE** 确定当前的插入符样式设置

SCI_GETCARETSTYLE。

块字符成功地绘制了大多数组合和多字节字符序列，但是当光标位于这些字符时，某些字体（如泰语字体（以及可能的其他字体））有时会显得很奇怪，这可能导致仅绘制光标字符序列的一部分。这在 Windows 平台上最为显着。

SCI_SETCARETWIDTH (int pixelWidth)

SCI_GETCARETWIDTH→int

线条插入符号的宽度可以设置 **SCI_SETCARETWIDTH** 为 0,1,2 或 3 像素的值。默认宽度为 1 像素。你可以回读当前的宽度 **SCI_GETCARETWIDTH**。宽度为 0 使插入符不可见（在版本 1.50 处添加），类似于将插入符样式设置为

CARETSTYLE_INVISIBLE（尽管不可互换）。当光标样式设置为线条插入符号模式时，此设置仅影响光标的宽度，它不会影响块插入符的宽度。

SCI_SETHOTSPOTACTIVEFORE (布尔 useSetting, 颜色脱颖而出)

SCI_GETHOTSPOTACTIVEFORE→颜色

SCI_SETHOTSPOTACTIVEBACK (布尔 useSetting, 颜色回)

SCI_GETHOTSPOTACTIVEBACK→颜色

SCI_SETHOTSPOTACTIVEUNDERLINE (布尔下划线)

SCI_GETHOTSPOTACTIVEUNDERLINE→布尔

SCI_SETHOTSPOTSINGLELINE (布尔单线)

SCI_GETHOTSPOTSINGLELINE→BOOL

当光标悬停在样式与热点文字属性集，可以修改默认着色并使用这些设置绘制下划线。单行模式会阻止热点包装到下一行。

SCI_SETCARETSTICKY (int useCaretStickyBehaviour)

SCI_GETCARETSTICKY→int

SCI_TOGGLECARETSTICKY

这些消息设置，获取或切换 **caretSticky** 设置，该设置控制何时保存线上插入符号的最后位置。

设置为 **SC_CARETSTICKY_OFF** (0) 时，粘滞标志关闭；所有文本更改（以及所有插入符号位置更改）将在移动到不同行时记住插入符号的新水平位置。这是默认值。

设置为 **SC_CARETSTICKY_ON** (1) 时，粘性标记打开，唯一会导致编辑器记住水平插入位置的是用鼠标或键盘移动插入符（左/右箭头键，主页/结束键等）。

当设置为 `SC_CARETSTICKY_WHITESPACE` (2) 时, 除了一个特殊情况外, 插入符的作用类似于模式 0 (粘性); 插入空格或制表符时。(包括仅粘贴空格/制表符 - 撤消, 重做等不会出现此行为..)。

`SCI_TOGGLECARETSTICKY` 从交换机 `SC_CARETSTICKY_ON` 和 `SC_CARETSTICKY_WHITESPACE` 向 `SC_CARETSTICKY_OFF` 和从 `SC_CARETSTICKY_OFF` 到 `SC_CARETSTICKY_ON`。

字符表示

某些字符 (如控制字符和无效字节) 没有可视字形或使用难以区分的字形。

控制字符 (代码小于 32 的字符, 或某些编码中的 128 到 159 之间的字符) 由 Scintilla 使用它们在圆角矩形中倒置的助记符显示。这些助记符来自信号的早期, 尽管仍然使用了一些 (例如, `LF` = 换行, `BS` = 后退空间, `CR` = 回车率)。

对于低'C0'值: "NUL", "SOH", "STX", "ETX", "EOT", "ENQ", "ACK", "BEL", "BS", "HT", "LF",
", "VT", "FF", "CR", "SO", "SI", "DLE", "DC1", "DC2", "DC3", "DC4", "NAK", "SYN", "ETB", "CAN", "EM", "SUB", "ESC", "FS", "GS", "RS", "US".

对于高'C1'值: "PAD", "HOP", "BPH", "NBH", "IND", "NEL", "SSA", "ESA", "HTS", "HTJ", "VTS",
", "PLD", "PLU", "RI", "SS2", "SS3", "DCS", "PU1", "PU2", "STS", "CCH", "MW", "SPA", "EPA", "SOS", "SGCI", "SCI", "CSI", "ST", "OSC", "PM", "APC".

无效字节以类似的方式显示, 其中"x"后跟其十六进制值, 如"xFE".

```
SCI_SETREPRESENTATION(const char *encodedCharacter, const char *representation)
SCI_GETREPRESENTATION(const char *encodedCharacter, char *representation) → int
SCI_CLEARREPRESENTATION(const char *encodedCharacter)
SCI_SETCONTROLCHARSYMBOL(int symbol)
SCI_GETCONTROLCHARSYMBOL → int
```

SCI_SETREPRESENTATION (const char * encodedCharacter, const char * representation)

SCI_GETREPRESENTATION (const char * encodedCharacter, char *表示 NUL 终止) →int

SCI_CLEARREPRESENTATION (const char * encodedCharacter)

任何字符, 包括通常显示为助记符的字符, 都可以用字符串表示倒圆角矩形。

例如，欧姆符号 ΩU+ 2126 看起来非常类似于希腊语 Omega 字符 ΩU+ 03C9，因此，对于 UTF-8 编码，要将欧姆符号区分为“U +2126Ω”，可以进行以下调用：
SCI_SETREPRESENTATION("\xe2\x84\xa6", "U+2126 \xe2\x84\xa6")

encodedCharacter 参数是当前编码中一个字符的字节的 NUL 终止字符串。这不能用于设置多字符串的表示。

NUL (0) 字符是一种特殊情况，因为 **encodedCharacter** 参数是 NUL 终止的，NUL 字符被指定为空字符串。

SCI_SETCONTROLCHARSYMBOL (int 符号)

SCI_GETCONTROLCHARSYMBOL→int

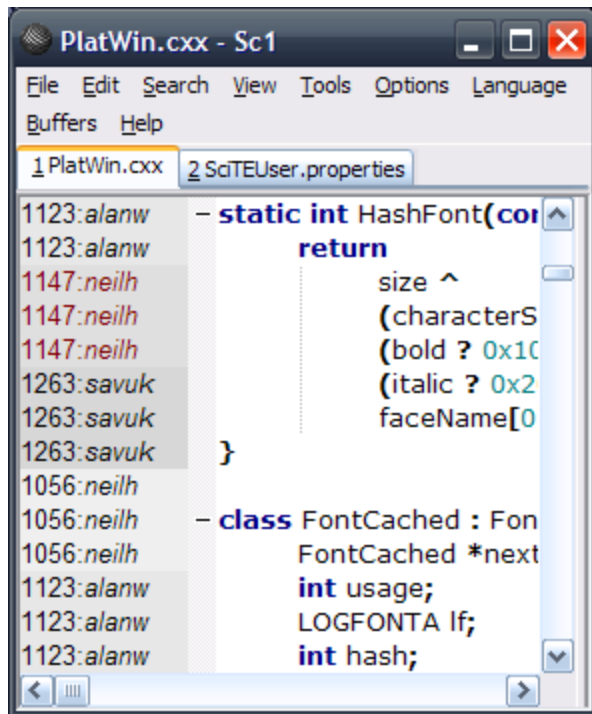
助记符可以用指定的符号替换，ASCII 码的范围是 32 到 255.如果设置的符号值小于 32，则所有控制字符都显示为助记符。您设置的符号将以为该字符设置的样式的字体呈现。您可以使用 **SCI_GETCONTROLCHARSYMBOL** 消息回读当前符号。默认符号值为 0。

边距

文本显示左侧可能有多个边距加上文本两侧的间隙。最初分配的 5 个边距从 0 到 **SC_MAX_MARGIN** (4) 编号，但可以通过调用来更改 **SCI_SETMARGINS**。可以将每个边距设置为仅显示符号，行号或文本 **SCI_SETMARGINTYPEN**。文本边距也可以显示符号。可以在每个边距中显示的标记设置为 **SCI_SETMARGINMASKN**。任何与可见边距无关的标记都将显示为文本中背景颜色的变化。可以为每个边距设置宽度（以像素为单位）。宽度为零的边距将被完全忽略。您可以选择是否在边距中单击鼠标会向容器发送 **SCN_MARGINCLICK** 或 **SCN_MARGINRIGHTCLICK** 通知，还是选择一行文本。

使用有效范围之外的保证金编号无效。默认情况下，边距 0 设置为显示行号，但宽度为 0，因此它被隐藏。边距 1 设置为显示非折叠符号，宽度为 16 像素，因此可见。边距 2 设置为显示折叠符号，但宽度为 0，因此它被隐藏。当然，您可以将边距设置为您想要的任何值。

用于显示修订和责备信息的样式文本边距：



```

SCI_SETMARGINS(int margins)
SCI_GETMARGINS → int
SCI_SETMARGINTYPEN(int margin, int marginType)
SCI_GETMARGINTYPEN(int margin) → int
SCI_SETMARGINWIDTHN(int margin, int pixelWidth)
SCI_GETMARGINWIDTHN(int margin) → int
SCI_SETMARGINMASKN(int margin, int mask)
SCI_GETMARGINMASKN(int margin) → int
SCI_SETMARGINSENSITIVEN(int margin, bool sensitive)
SCI_GETMARGINSENSITIVEN(int margin) → bool
SCI_SETMARGINCURSORN(int margin, int cursor)
SCI_GETMARGINCURSORN(int margin) → int
SCI_SETMARGINBACKN(int margin, colour back)
SCI_GETMARGINBACKN(int margin) → colour
SCI_SETMARGINLEFT(<unused>, int pixelWidth)
SCI_GETMARGINLEFT → int
SCI_SETMARGINRIGHT(<unused>, int pixelWidth)
SCI_GETMARGINRIGHT → int
SCI_SETFOLDMARGINCOLOUR(bool useSetting, colour back)
SCI_SETFOLDMARGINHICOLOUR(bool useSetting, colour fore)
SCI_MARGINSETTEXT(int line, const char *text)
SCI_MARGINGETTEXT(int line, char *text) → int
SCI_MARGINSETSTYLE(int line, int style)
SCI_MARGINGETSTYLE(int line) → int
SCI_MARGINSETSTYLES(int line, const char *styles)
SCI_MARGINGETSTYLES(int line, char *styles) → int
SCI_MARGINTEXTCLEARALL
SCI_MARGINSETSTYLEOFFSET(int style)
SCI_MARGINGETSTYLEOFFSET → int
SCI_SETMARGINOPTIONS(int marginOptions)
SCI_GETMARGINOPTIONS → int

```

SCI_SETMARGINS (int margin)

SCI_GETMARGINS → int

分配边距数或查找当前分配的边距数。

SCI_SETMARGINTYPEN (int margin, int marginType)

SCI_GETMARGINTYPEN (int margin) → int

这两个例程设置并获取边距的类型。margin 参数应为 0,1,2,3 或 4。您可以使用预定义常量 **SC_MARGIN_SYMBOL** (0) 和 **SC_MARGIN_NUMBER** (1) 将边距设置为行号或符号边距。具有应用程序定义文本的边距可以使用 **SC_MARGIN_TEXT** (4) 或 **SC_MARGIN_RTEXT** (5) 来对文本进行右对齐。按照惯例，边距 0 用于行号，接下来的两个用于符号。您还可以将常量 **SC_MARGIN_BACK** (2)，**SC_MARGIN_FORE** (3) 和 **SC_MARGIN_COLOUR** (6) 用于符号边距，将其背景颜色设置为与 **STYLE_DEFAULT** 背景和前景色或指定颜色相匹配。

SCI_SETMARGINWIDTHN (int margin, int pixelWidth)

SCI_GETMARGINWIDTHN (int margin) → int

这些例程设置并获取边距的宽度（以像素为单位）。宽度为零的边距是不可见的。默认情况下，Scintilla 为宽度为 16 像素的符号设置边距 1，因此如果您不确定哪些是合适的，这是一个合理的猜测。行号边距宽度应考虑文档中的行数和行号样式。你可以使用类似的东西 **SCI_TEXTWIDTH(STYLE_LINENUMBER, "_99999")** 来获得合适的宽度。

SCI_SETMARGINMASKN (int margin, int mask)

SCI_GETMARGINMASKN (int margin) → int

掩码是 32 位值。每个位对应于 32 个逻辑符号中的一个，这些符号可以在为符号启用的边缘中显示。有一个有用的常量 **SC_MASK_FOLDERS** (0xFE000000 或 -33554432)，它是用于表示折叠的 7 个逻辑符号的掩码。您可以为 32 个逻辑符号中的每一个分配各种符号和颜色，有关详细信息，请参阅[标记](#)。如果 **(mask & SC_MASK_FOLDERS) == 0**，边距背景颜色由样式 33 (**STYLE_LINENUMBER**) 控制。

您可以使用逻辑标记添加到行 **SCI_MARKERADD**。如果一条线具有相关标记，该标记未出现在具有非零宽度的任何边距的蒙版中，则标记会更改该线的背景颜色。例如，假设您决定使用逻辑标记 10 来标记具有语法错误的行，并且您希望通过更改背景颜色来显示这些行。该标记的掩码向左移动 10 次 ($1 \ll 10$)，即 0x400。如果确保其掩码中没有符号边距包含 0x400，则带有标记的任何行都会更改背景颜色。

设置非折叠边距 1 使用 **SCI_SETMARGINMASKN(1, ~SC_MASK_FOLDERS)**，这是 Scintilla 设置的默认值。设置折叠边距 2 使用 **SCI_SETMARGINMASKN(2, SC_MASK_FOLDERS)**。 **~SC_MASK_FOLDERS** 是十六进制的 0xFFFFFFFF 或十进制的 33554431。当然，您可能需要在边距中显示所有 32 个符号，在这种情况下使用 **SCI_SETMARGINMASKN(margin, -1)**。

SCI_SETMARGINSENSITIVEN (int margin, bool sensitive)

SCI_GETMARGINSENSITIVEN (int margin) → bool

五个边距中的每一个都可以设置为对鼠标点击敏感或不敏感。敏感边距中的单击会向容器发送 [SCN_MARGINCLICK](#) 或通知。不敏感的边距作为选择边距，可以轻松选择线条范围。默认情况下，所有边距都不敏感。 [SCN_MARGINRIGHTCLICK](#)

SCI_SETMARGINCURSORN (int margin, int cursor)

SCI_GETMARGINCURSORN (int margin) →int

反向箭头光标通常显示在所有边距上。这可以更改为正常箭头，

SCI_SETMARGINCURSORN(margin, SC_CURSORARROW)或者恢复为反向箭头

SCI_SETMARGINCURSORN(margin, SC_CURSORREVERSEARROW)。

SCI_SETMARGINBACKN (int margin, color back)

SCI_GETMARGINBACKN (int margin) →color

类型的边距 **SC_MARGIN_COLOUR** 可以设置颜色 **SCI_SETMARGINBACKN**。

SCI_SETMARGINLEFT (<unused>, int pixelWidth)

SCI_GETMARGINLEFT→int

SCI_SETMARGINRIGHT (<unused>, int pixelWidth)

SCI_GETMARGINRIGHT→int

这些消息设置并获取文本两侧空白边距的宽度（以像素为单位）。默认是每侧一个像素。

SCI_SETFOLDMARGINCOLOUR (bool useSetting, color back)

SCI_SETFOLDMARGINHICOLOUR (bool useSetting, color fore)

这些消息允许更改折叠边距的颜色和折叠边距高光。在 Windows 上，折叠边距颜色默认为:: GetSysColor (COLOR_3DFACE)，折叠边距高亮颜色为::

GetSysColor (COLOR_3DHIGHLIGHT)。

SCI_MARGINSETTEXT (int line, const char * text)

SCI_MARGINGETTEXT (int line, char * text) →int

SCI_MARGINSETSTYLE (int line, int style)

SCI_MARGINGETSTYLE (int line) →int

SCI_MARGINSETSTYLES (int line, const char * styles)

SCI_MARGINGETSTYLES (int line , char * styles) →int

SCI_MARGINTEXTCLEARALL

使用 **SC_MARGIN_TEXT** 或 **SC_MARGIN_RTEXT** 类型创建文本边距。可以为每一行设置不同的字符串 **SCI_MARGINSETTEXT**。一行上的整个文本边距可以以特定样式显示， **SCI_MARGINSETSTYLE** 或者每个字符可以单独设置样式

SCI_MARGINSETSTYLES，使用字节数组，每个字节设置相应文本字节的样式类似于 **SCI_SETSTYLINGEX**。设置文本边距将导致 [SC_MOD_CHANGEMARGIN](#) 发送通知。

在文本边距中只有一些样式属性处于活动状态：font, size / sizeFractional, bold / weight, italics, fore, back 和 characterSet。

SCI_MARGINSETSTYLEOFFSET (int style)

SCI_MARGINGETSTYLEOFFSET→int

通过设置样式偏移量，可以将标注样式与标准文本样式完全分开。例如，

SCI_MARGINSETSTYLEOFFSET(256)允许边距样式从 256 到 511 编号，因此它们不会与词法分析器设置的样式重叠。在查找样式之前，每个样式编号都设置了 **SCI_MARGINSETSTYLE** 或者 **SCI_MARGINSETSTYLES** 添加了偏移量。

随时调用 **SCI_ALLOCATEEXTENDEDSTYLES** 前 **SCI_MARGINSETSTYLEOFFSET** 和使用结果作为参数 **SCI_MARGINSETSTYLEOFFSET**。

SCI_SETMARGINOPTIONS (int marginOptions)

SCI_GETMARGINOPTIONS→int

通过启用适当的位标志来定义边距选项。目前，只有一个标志可用

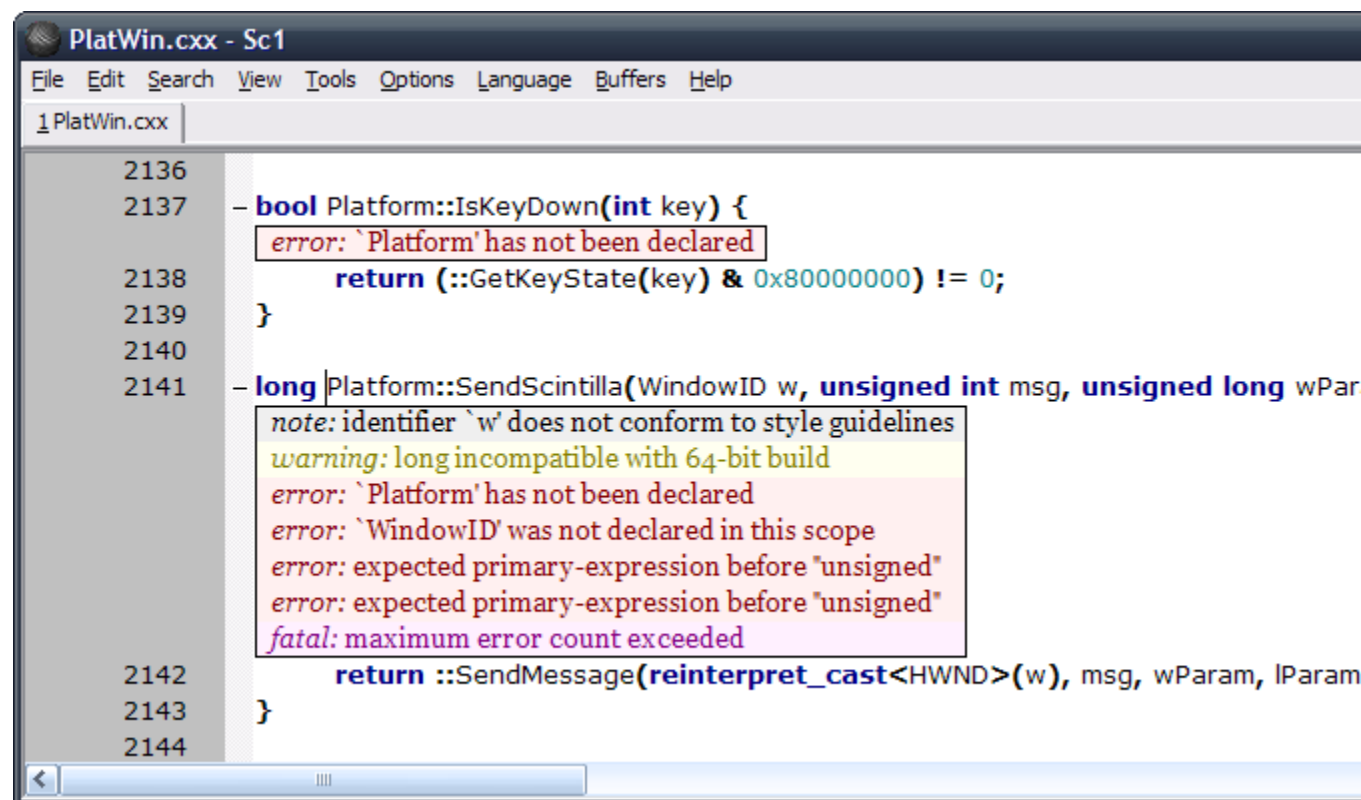
SC_MARGINOPTION_SUBLINESELECT= 1，它控制在单击前面的边距时如何选择包裹的线条。如果 **SC_MARGINOPTION_SUBLINESELECT** 设置，则仅选择包裹线的子线，否则选择整个包裹线。**SC_MARGINOPTION_NONE** 默认情况下，保证金选项设置为 = 0。

注释

注释是每行可编辑文本下方的只读文本行。注释可以由多个以'\ n'分隔的行组成。注释可用于显示用于调试的代码的汇编程序版本，或在内联中显示诊断消息或在合并工具中排列不同版本的文本。

注释计为方法 **SCI_VISIBLEFROMDOCLINE** 和 显示行 **SCI_DOCLINEFROMVISIBLE**

用于内联诊断的注释：



```

SCI_ANNOTATIONSETTEXT(int line, const char *text)
SCI_ANNOTATIONGETTEXT(int line, char *text) → int
SCI_ANNOTATIONSETSTYLE(int line, int style)
SCI_ANNOTATIONGETSTYLE(int line) → int
SCI_ANNOTATIONSETSTYLES(int line, const char *styles)
SCI_ANNOTATIONGETSTYLES(int line, char *styles) → int
SCI_ANNOTATIONGETLINES(int line) → int
SCI_ANNOTATIONCLEARALL
SCI_ANNOTATIONSETVISIBLE(int visible)
SCI_ANNOTATIONGETVISIBLE → int
SCI_ANNOTATIONSETSTYLEOFFSET(int style)
SCI_ANNOTATIONGETSTYLEOFFSET → int

```

```

SCI_ANNOTATIONSETTEXT (int line, const char * text)
SCI_ANNOTATIONGETTEXT (int line, char * text) →int
SCI_ANNOTATIONSETSTYLE (int line, int style)
SCI_ANNOTATIONGETSTYLE (int line) →int
SCI_ANNOTATIONSETSTYLES (int line, const char * styles)
SCI_ANNOTATIONGETSTYLES (int line , char * styles) →int
SCI_ANNOTATIONGETLINES (int line) →int
SCI_ANNOTATIONCLEARALL

```

可以为每一行设置不同的字符串 **SCI_ANNOTATIONSETTEXT**。 **SCI_ANNOTATIONSETTEXT** 使用 NULL 指针清除注释调用。一行上的整个文本注释可以以特定样式显示， **SCI_ANNOTATIONSETSTYLE** 或者每个字符可以单独地设置样式 **SCI_ANNOTATIONSETSTYLES** 它使用一个字节数组，每个字节设置相应文本字节的样式类似于 **SCI_SETSTYLINGEX**。必须首先设置文本，因为它指定注释的时间长度，以便读取样式的字节数。设置注释将导致 **SC_MOD_CHANGEANNOTATION** 发送通知。

可以检索注释一行的行数 **SCI_ANNOTATIONGETLINES**。可以清除所有行的注释， **SCI_ANNOTATIONCLEARALL** 相当于清除每一行（设置为 0），然后删除用于此功能的其他内存。

注释中只有一些样式属性处于活动状态：font， size / sizeFractional， bold / weight， italics， fore， back 和 characterSet。

```

SCI_ANNOTATIONSETVISIBLE (int visible)
SCI_ANNOTATIONGETVISIBLE→int

```

注释可以在视图中显示，并且可见时可以选择显示样式。这两条消息设置并获得注释显示模式。该 *visible* 论点可以是以下之一：

ANNOTATION_HIDDEN	0 不显示注释。
ANNOTATION_STANDARD	1 注释是左对齐的，没有装饰。
ANNOTATION_BOXED	2 注释缩进以匹配文本并被框包围。
ANNOTATION_INDENTED	3 缩进缩进以匹配文本。

```

SCI_ANNOTATIONSETSTYLEOFFSET (int style)
SCI_ANNOTATIONGETSTYLEOFFSET→int

```

通过设置样式偏移，可以将注释样式与标准文本样式完全分开。例如，**SCI_ANNOTATIONSETSTYLEOFFSET(512)**允许注释样式从 512 到 767 编号，因此它们不会与词法分析器设置的样式重叠（如果边距偏移为 256，则为边距）。在查找样式之前，每个样式编号都设置了 **SCI_ANNOTATIONSETSTYLE** 或者 **SCI_ANNOTATIONSETSTYLES** 添加了偏移量。

随时调用 **SCI_ALLOCATEEXTENDEDSTYLES** 前 **SCI_ANNOTATIONSETSTYLEOFFSET** 和使用结果作为参数 **SCI_ANNOTATIONSETSTYLEOFFSET**。

其他设置

```
SCI_SETBUFFEREDDRAW(bool buffered)
SCI_GETBUFFEREDDRAW → bool
SCI_SETPHASESDRAW(int phases)
SCI_GETPHASESDRAW → int
SCI_SETTECHNOLOGY(int technology)
SCI_GETTECHNOLOGY → int
SCI_SETFONTQUALITY(int fontQuality)
SCI_GETFONTQUALITY → int
SCI_SETCODEPAGE(int codePage)
SCI_GETCODEPAGE → int
SCI_SETIMEINTERACTION(int imeInteraction)
SCI_GETIMEINTERACTION → int
SCI_SETBIDIRECTIONAL(int bidirectional)
SCI_GETBIDIRECTIONAL → int
SCI_GRABFOCUS
SCI_SETFOCUS(bool focus)
SCI_GETFOCUS → bool
```

要转发消息(**WM_XXXX**, **WPARAM**, **LPARAM**)给 Scintilla 的，你可以使用 **SendMessage(hScintilla, WM_XXXX, WPARAM, LPARAM)**其中 **hScintilla** 的把手的是你的编辑器创建的 Scintilla 的窗口。

虽然我们在 Windows 中转发消息的主题，但顶级窗口应该将任何 **WM_SETTINGCHANGE** 消息转发给 Scintilla（这当前用于收集鼠标设置的更改，但将来可以用于其他用户界面项）。

SCI_SETBUFFEREDDRAW (bool buffered)
SCI_GETBUFFEREDDRAW→bool

这些消息打开或关闭缓冲绘图并报告缓冲的绘图状态。缓冲绘图将每一行绘制成位图而不是直接绘制到屏幕上，然后将位图复制到屏幕上。这可以避免闪烁，但确实需要更长的时间。默认情况下，绘图在 Win32 和 GTK +上缓冲，并且不在 Cocoa 和 Qt上缓冲。Cocoa 不支持缓冲绘图。

当前平台执行窗口缓冲，因此关闭此选项几乎总是更好。对于 Win32 和 GTK +，客户端代码应在初始化时关闭缓冲。有一些较旧的平台和不寻常的模式，缓冲可能仍然有用。

SCI_SETPHASESDRAW (int 阶段)

SCI_GETPHASESDRAW→int

有几个顺序可以绘制文本区域，提供速度之间的权衡，即使文本像素与其他元素重叠，也可以看到所有文本像素。

在单相绘图 (**SC_PHASES_ONE**) 中，一个样式中的每个字符集都与其背景一起绘制。如果一个字符悬于一个运行的结尾，例如在“ V_”中，其中“ V”与“_”的风格不同，那么这可能导致“ V” 的右侧被透支“_”的背景将其切断。

不推荐使用单相绘图，应用程序不应使用它。

两阶段绘图 (**SC_PHASES_TWO**) 通过首先绘制线条的所有背景然后以透明模式绘制文本来修复此问题。线条是单独绘制的，没有线条会与另一条线条重叠，因此任何与另一条线重叠的像素（如字符上的极端上升线和下降线）都将被切断。除非打开缓冲绘图或平台自然缓冲，否则两相绘图可能比单相闪烁更多。默认是绘制为两个阶段。

多阶段绘图 (**SC_PHASES_MULTIPLE**) 多次绘制整个区域，每个特征绘制一次，在层或阶段中构建外观。在任何文本之前绘制所有线条的彩色背景，然后在此组合背景上以透明模式绘制所有文本，而不将文本剪切到线条边界。这允许极端的上升和下降器溢出到相邻的线中。此模式与缓冲绘图不兼容，并且就像

SC_PHASES_TWO 打开缓冲绘图一样。多相绘图比两相绘图慢。使用

SCI_SETLAYOUTCACHE(SC_CACHE_PAGE) 或更高级别设置布局缓存 可以确保多阶段绘制不会明显变慢。

SCI_SETTECHNOLOGY (int technology)

SCI_GETTECHNOLOGY→int

技术属性允许在不同的绘图 API 和选项之间进行选择。在大多数平台上，唯一的选择是 **SC_TECHNOLOGY_DEFAULT** (0)。在 Windows Vista 或更高版本中，可以选择 **SC_TECHNOLOGY_DIRECTWRITE** (1)， **SC_TECHNOLOGY_DIRECTWRITERETAIN** (2) 或 **SC_TECHNOLOGY_DIRECTWRITEDC** (3) 使用 Direct2D 和 DirectWrite API 进行更高质量的抗锯齿绘制。 **SC_TECHNOLOGY_DIRECTWRITERETAIN** 不同之处在于

SC_TECHNOLOGY_DIRECTWRITE 请求在呈现之后保留框架，这可以防止某些卡和驱动器上的绘图失败。 **SC_TECHNOLOGY_DIRECTWRITEDC** 与 **SC_TECHNOLOGY_DIRECTWRITE** 使用 DirectWrite 绘制到 GDI DC 不同。由于 Direct2D 可以缓冲绘图，因此可以关闭 Scintilla 的缓冲 **SCI_SETBUFFEREDDRAW(0)**。

SCI_SETFONTQUALITY (int fontQuality)

SCI_GETFONTQUALITY→int

管理字体质量（抗锯齿方法）。目前，下列值在 Windows 上可用：

SC_EFF_QUALITY_DEFAULT（向下兼容） **SC_EFF_QUALITY_NON_ANTIALIASED**，
SC_EFF_QUALITY_ANTIALIASED， **SC_EFF_QUALITY_LCD_OPTIMIZED**。

如果需要将更多选项压缩到此属性中，则只有 **SC_EFF_QUALITY_MASK** (0xf) 定义的有限位数才会用于质量。

SCI_SETCODEPAGE (int codePage)

SCI_GETCODEPAGE→int

Scintilla 支持 UTF-8，日语，中文和韩语 DBCS 以及 Latin-1 等单字节编码。

UTF-8 (**SC_CP_UTF8**) 是默认值。使用此消息并 **codePage** 设置为代码页编号，将 Scintilla 设置为使用代码页信息，以确保将多个字节字符视为一个字符而不是多个字符。这也可以阻止插入符号在多字节字符的字节之间移动。不要使用此消息在不同的单字节字符集之间进行选择 - 使用 **SCI_STYLESETCHARACTERSET**。调用 **codePage** 设置为零以禁用多字节支持。

代码页 **SC_CP_UTF8** (65001) 将 Scintilla 设置为 Unicode 模式，文档被视为以 UTF-8 表示的字符序列。在被 OS 绘制之前，文本将转换为平台的正常 Unicode 编码，因此可以显示希伯来语，阿拉伯语，西里尔语和汉字。可以使用在一个水平空间中垂直堆叠的两个字符（例如泰语）的语言将主要起作用，但是存在一些问题，其中字符被单独绘制而导致视觉故障。不支持双向文本。

代码页可以设置为 65001 (UTF-8)，932 (日语 Shift-JIS)，936 (简体中文 GBK)，949 (韩国统一韩语代码)，950 (繁体中文 Big5) 或 1361 (韩国 Johab)。

SCI_SETIMEINTERACTION (int imeInteraction)

SCI_GETIMEINTERACTION→int

当输入中文，日文或韩文文本时，可能会显示输入法编辑器 (IME)。IME 可以是出现在 Scintilla 上方的额外窗口，也可以由 Scintilla 本身作为文本显示。在某些平台上，可以选择两种技术。窗口化 IME **SC_IME_WINDOWED** (0) 在外观和行为上可能与其他应用中的 IME 更相似。内联 IME **SC_IME_INLINE** (1) 可以更好地使用一些 Scintilla 特征，例如矩形和多重选择。

可以选择窗口行为，**SCI_SETIMEINTERACTION(SC_IME_WINDOWED)** 并使用内联行为 **SCI_SETIMEINTERACTION(SC_IME_INLINE)**。在某些情况下，Scintilla 可能会忽略此调用。例如，某些语言可能仅支持内联行为。

当内联 IME 模式处于活动状态时，在最终确定之前暂时添加字符，并为每个字符发送 **SCN_CHARADDED** 通知。

这些双向功能是实验性的和不完整的。

SCI_SETBIDIRECTIONAL (int 双向)

SCI_GETBIDIRECTIONAL→int

有些语言，如阿拉伯语和希伯来语，是从右到左，而不是从左到右，就像英语一样。使用多种语言的文档可能包含两个方向，这称为“双向”。默认文本方向可以从右到左或从左到右。Scintilla 仅在某些平台上正确显示双向文本。目前，使用 DirectWrite 在 Win32 上和使用 Cocoa 的 macOS 上实现了对双向文本的实验支持。只有 UTF-8 文档才会显示双向行为，并且仅在 **SC_BIDIRECTIONAL_L2R** 模式下显示。某些功能（如虚拟空间）可能无法与双向文本一起使用，或者仅在某些情况下可能有效。

双向文本还有额外的处理和存储成本。由于某些应用程序可能不想支付成本，因此必须通过调用 `SCI_SETBIDIRECTIONAL(SC_BIDIRECTIONAL_L2R)` (1) 从左到右选择默认方向或 `SCI_SETBIDIRECTIONAL(SC_BIDIRECTIONAL_R2L)` (2) 默认从右到左显式启用双向支持。在 Win32 中，这应该在技术设置后进行 `SC_TECHNOLOGY_DIRECTWRITE`，`SC_TECHNOLOGY_DIRECTWRITERETAIN` 或 `SC_TECHNOLOGY_DIRECTWRITEDC`。

如果调用成功 `SCI_GETBIDIRECTIONAL` 将返回相同的值，否则 `SC_BIDIRECTIONAL_DISABLED` 返回 (0)。

SCI_GRABFOCUS

SCI_SETFOCUS (bool 焦点)

SCI_GETFOCUS → bool

Scintilla 可以被告知抓住这个消息的焦点。这在 GTK + 上需要更多，其中焦点处理比在 Windows 上更复杂。

可以设置内部焦点标志 `SCI_SETFOCUS`。具有复杂焦点要求的客户端使用此功能，例如拥有自己的窗口以获得真正的焦点，但需要指示 Scintilla 具有逻辑焦点。

支持突出显示

```
SCI_BRACEHIGHLIGHT(int posA, int posB)
SCI_BRACEBADLIGHT(int pos)
SCI_BRACEHIGHLIGHTINDICATOR(bool useSetting, int indicator)
SCI_BRACEBADLIGHTINDICATOR(bool useSetting, int indicator)
SCI_BRACEMATCH(int pos, int maxReStyle) → position
```

SCI_BRACEHIGHLIGHT (int posA, int posB)

“大括号突出显示样式”最多可突出显示两个字符，定义为样式编号

`STYLE_BRACELIGHT` (34)。如果您启用了缩进指南，您可能还希望突出显示与括号对应的缩进。您可以找到列，`SCI_GETCOLUMN` 并突出显示缩进

`SCI_SETHIGHLIGHTGUIDE`。

SCI_BRACEBADLIGHT (int pos)

如果没有匹配的大括号，那么大括号的 **badlighting** 样式 `style STYLE_BRACEBAD` (35) 可用于显示不匹配的大括号。使用 `INVALID_POSITION` (-1) 的位置删除突出显示。

SCI_BRACEHIGHLIGHTINDICATOR (bool useSetting, int indicator)

使用指定的指示器突出显示匹配的大括号，而不是更改其样式。

SCI_BRACEBADLIGHTINDICATOR (bool useSetting, int indicator)

使用指定的指示器突出显示非匹配的大括号，而不是更改其样式。

SCI_BRACEMATCH (INT POS, INT maxReStyle) → 位置

的 `SCI_BRACEMATCH` 消息给定找到的相应匹配的括号 *pos*，一根支撑的位置。处理

的大括号字符是'(', ')', '[', ']', '{', '}', '<'和'>'。搜索是从左大括号向前搜索，从右大括号向后搜索。如果位置上的字符不是大括号字符，或者找不到匹配的大括号，则返回值为-1。否则，返回值是匹配括号的位置。

仅当匹配括号的样式与起始大括号相同或匹配大括号超出样式结束时才会出现匹配。嵌套大括号处理正确。该 *maxReStyle* 参数当前必须为 0 - 将来可能会使用该参数来限制括号搜索的长度。

标签和缩进指南

缩进（行开头的空格）经常被程序员用来澄清程序结构，在某些语言中，例如 Python，它可能是语言语法的一部分。选项卡通常用于编辑器中以插入制表符或填充带有空格的文本，直到下一个选项卡。

当 Scintilla 布置一段文本时，制表符后面的文本通常会显示在左侧的下一个 TABWIDTH 列的多个位置。但是，也可以为每行明确设置 tabstops（以像素为单位）。

可以将 Scintilla 设置为以特殊方式处理行开头处的空白区域中的制表符和退格键：插入制表符将该行缩进到下一个缩进位置，而不是仅在当前字符位置插入制表符并且退格时不需要而不是删除一个字符。Scintilla 还可以显示缩进指南（垂直线），以帮助您生成代码。

```
SCI_SETTABWIDTH(int tabWidth)
SCI_GETTABWIDTH → int
SCI_CLEARTABSTOPS(int line)
SCI_ADDTABSTOP(int line, int x)
SCI_GETNEXTTABSTOP(int line, int x) → int
SCI_SETUSETABS(bool useTabs)
SCI_GETUSETABS → bool
SCI_SETINDENT(int indentSize)
SCI_GETINDENT → int
SCI_SETTABINDENTS(bool tabIndents)
SCI_GETTABINDENTS → bool
SCI_SETBACKSPACEUNINDENTS(bool bsUnIndents)
SCI_GETBACKSPACEUNINDENTS → bool
SCI_SETLINEINDENTATION(int line, int indentation)
SCI_GETLINEINDENTATION(int line) → int
SCI_GETLINEINDENTPOSITION(int line) → position
SCI_SETINDENTATIONGUIDES(int indentView)
SCI_GETINDENTATIONGUIDES → int
SCI_SETHIGHLIGHTGUIDE(int column)
SCI_GETHIGHLIGHTGUIDE → int
```

SCI_SETTABWIDTH (int tabWidth)

SCI_GETTABWIDTH→int

SCI_SETTABWIDTH 将制表符的大小设置为空格字符大小的倍数 **STYLE_DEFAULT**。默认选项卡宽度为 8 个字符。标签大小没有限制，但小于 1 或大值的值可能会产生不良影响。

SCI_CLEARTABSTOPS (int line)

SCI_ADDTABSTOP (int line, int x)

SCI_GETNEXTTABSTOP (int line, int x) →int

SCI_CLEARTABSTOPS 清除一行上的显式 tabstops。**SCI_ADDTABSTOP** 在距左侧指定的距离处添加一个显式的 tabstop（以像素为单位），并 **SCI_GETNEXTTABSTOP** 在给定的 x 位置后获得下一个显式的 tabstop 位置，如果没有，则为零。更改制表位会生成 **SC_MOD_CHANGETABSTOPS** 通知。

SCI_SETUSETABS (bool useTabs)

SCI_GETUSETABS →bool

SCI_SETUSETABS 确定缩进是应该由制表符和空格的混合创建还是纯粹基于空格。设置 *useTabs* 为 **false**（0）以创建空格中的所有制表符和缩进。默认是 **true**。您可以使用 **SCI_GETCOLUMN** 获取位置的列来考虑选项卡的宽度。

SCI_SETINDENT (int indentSize)

SCI_GETINDENT →int

SCI_SETINDENT 根据空间宽度设置缩进的大小 **STYLE_DEFAULT**。如果将宽度设置为 0，则缩进大小与制表符大小相同。缩进大小没有限制，但小于 0 或大值的值可能会产生不良影响。

SCI_SETTABINDENTS (bool tabIndents)

SCI_GETTABINDENTS →bool

SCI_SETBACKSPACEUNINDENTS (bool bsUnIndents)

SCI_GETBACKSPACEUNINDENTS →bool

在缩进空白区域内，制表符和退格键可以缩进和取消，而不是插入制表符或删除带 **SCI_SETTABINDENTS** 和和字符的字符 **SCI_SETBACKSPACEUNINDENTS**。

SCI_SETLINEINDENTATION (int line, int indentation)

SCI_GETLINEINDENTATION (int line) →int

可以使用 **SCI_GETLINEINDENTATION** 发现和设置行上的缩进量

SCI_SETLINEINDENTATION。缩进在字符列中测量，对应于空格字符的宽度。

SCI_GETLINEINDENTPOSITION (int line) →position

返回行缩进结束时的位置。

SCI_SETINDENTATIONGUIDES (int indentView)

SCI_GETINDENTATIONGUIDES →int

缩进指南是虚线垂直线，每个缩进尺寸列都显示在缩进空白区域内。它们可以很容易地看出哪些结构排列在一起，特别是当它们扩展到多个页面时。样式 **STYLE_INDENTGUIDE**（37）用于指定缩进指南的前景色和背景色。

有 4 个缩进指南视图。**SC_IV_NONE** 关闭此功能，但其他 3 个状态确定指南在空行上显示的距离。

SC_IV_NONE

没有显示缩进指南。

SC_IV_REAL	缩进指南显示在真正的缩进空白区域内。
	缩进指南显示在实际缩进之外，直到下一个非空行的级别。
SC_IV_LOOKFORWARD	如果前一个非空行是折叠标题，则显示缩进指南比该行多一个缩进级别。此设置适用于 Python。
SC_IV_LOOKBOTH	缩进指南显示在实际缩进之外，直到下一个非空行或前一个非空行的级别，以较大者为准。此设置适用于大多数语言。

SCI_SETHIGHLIGHTGUIDE (int column)

SCI_GETHIGHLIGHTGUIDE→int

当发生大括号突出显示时，对应于大括号的缩进指南可以用大括号突出显示样式突出显示 **STYLE_BRACELIGHT** (34)。设置 **column** 为 0 可取消此突出显示。

标记

有 32 个标记，编号为 0 到 **MARKER_MAX** (31)，您可以将它们的任意组合分配给文档中的每一行。标记出现在文本左侧的[选择边距](#)中。如果选择边距设置为零宽度，则改变整行的背景颜色。标记号 25 至 31 由 Scintilla 在折叠边缘中使用，并且具有 **SC_MARKNUM_**例如形式*的符号名称 **SC_MARKNUM_FOLDEROPEN**。

标记号 0 到 24 没有预定义的功能; 您可以使用它们来标记语法错误或当前执行点，断点或您需要标记的任何内容。如果您不需要折叠，可以将所有 32 用于任何目的。

每个标记号都有一个与之关联的符号。您还可以为每个标记编号设置前景色和背景色，这样您可以多次使用相同的符号，并为不同的用途使用不同的颜色。Scintilla 有一组你可以指定的符号 (**SC_MARK_***)，或者你可以使用字符。默认情况下，所有 32 个标记都设置为 **SC_MARK_CIRCLE** 黑色前景和白色背景。

标记按其编号顺序绘制，因此编号较高的标记出现在较低编号的标记之上。标记通过跟踪线条开始移动的位置来尝试移动文本。删除行时，其标记通过 **OR** 操作与下一行的标记组合。

```

SCI_MARKERDEFINE(int markerNumber, int markerSymbol)
SCI_MARKERDEFINEPIXMAP(int markerNumber, const char *pixmap)
SCI_RGBAIMAGESETWIDTH(int width)
SCI_RGBAIMAGESETHEIGHT(int height)
SCI_RGBAIMAGESETSCALE(int scalePercent)
SCI_MARKERDEFINERGBAIMAGE(int markerNumber, const char *pixels)
SCI_MARKERSYMBOLDEFINED(int markerNumber) → int
SCI_MARKERSETFORE(int markerNumber, colour fore)
SCI_MARKERSETBACK(int markerNumber, colour back)
SCI_MARKERSETBACKSELECTED(int markerNumber, colour back)
SCI_MARKERENABLEHIGHLIGHT(bool enabled)
SCI_MARKERSETALPHA(int markerNumber, alpha alpha)
SCI_MARKERADD(int line, int markerNumber) → int
SCI_MARKERADDSSET(int line, int markerSet)
SCI_MARKERDELETE(int line, int markerNumber)
SCI_MARKERDELETEALL(int markerNumber)

```



```

SCI_MARKERGET(int line) → int
SCI_MARKERNEXT(int lineStart, int markerMask) → int
SCI_MARKERPREVIOUS(int lineStart, int markerMask) → int
SCI_MARKERLINEFROMHANDLE(int markerHandle) → int
SCI_MARKERDELETEHANDLE(int markerHandle)

```

SCI_MARKERDEFINE (int markerNumber, int markerSymbol)

此消息将 0 到 31 范围内的标记号与其中一个标记符号或 ASCII 字符相关联。目前可用的通用标志符号是：

```

SC_MARK_CIRCLE, SC_MARK_ROUNDRECT, SC_MARK_ARROW, SC_MARK_SMALLRECT,
SC_MARK_SHORTARROW, SC_MARK_EMPTY, SC_MARK_ARROWDOWN, SC_MARK_MINUS,
SC_MARK_PLUS, SC_MARK_ARROWS, SC_MARK_DOTDOTDOT, SC_MARK_BACKGROUND,
SC_MARK_LEFTRECT, SC_MARK_FULLRECT, SC_MARK_BOOKMARK, 和 SC_MARK_UNDERLINE。

```

该 **SC_MARK_BACKGROUND** 标记仅更改的行的背景色。该 **SC_MARK_FULLRECT** 符号反映了这一点，只改变保证金的背景颜色。 **SC_MARK_UNDERLINE** 在文本中绘制下划线。的 **SC_MARK_EMPTY** 符号是不可见的，从而允许客户机代码来跟踪线的移动。如果您更改了折叠样式并希望一个或多个 **SC_FOLDERNUM_*** 标记没有关联的符号，您也可以使用它。

应用程序可以使用标记符号 **SC_MARK_AVAILABLE** 来指示插件可以分配该标记号。

还有标记符号设计用于折叠边缘的扁平树样式。

```

SC_MARK_BOXMINUS, SC_MARK_BOXMINUSCONNECTED, SC_MARK_BOXPLUS,
SC_MARK_BOXPLUSCONNECTED, SC_MARK_CIRCLEMINUS,
SC_MARK_CIRCLEMINUSCONNECTED, SC_MARK_CIRCLEPLUS,
SC_MARK_CIRCLEPLUSCONNECTED, SC_MARK_LCORNER, SC_MARK_LCORNERCURVE,
SC_MARK_TCORNER, SC_MARK_TCORNERCURVE, 和 SC_MARK_VLINE。

```

通过将字符的 ASCII 值添加到 **SC_MARK_CHARACTER** (10000)，可以将字符用作标记。例如，要使用 'A' (ASCII 代码 65) 作为标记号 1，请使用：

```
SCI_MARKERDEFINE(1, SC_MARK_CHARACTER+65)。
```

标记数字 **SC_MARKNUM_FOLDER** 和 **SC_MARKNUM_FOLDEROPEN** 用于表示一个折叠存在并且打开或关闭。尽管 (**SC_MARK_PLUS**, **SC_MARK_MINUS**) 对或 (**SC_MARK_ARROW**, **SC_MARK_ARROWDOWN**) 对是好的选择，但可以为此分配任何符号。除了这两点，都需要扁平树风格比较分配：**SC_MARKNUM_FOLDEREND**, **SC_MARKNUM_FOLDERMIDTAIL**, **SC_MARKNUM_FOLDEROPENMID**, **SC_MARKNUM_FOLDERSUB**, 和 **SC_MARKNUM_FOLDERTAIL**。用于折叠的位由指定 **SC_MASK_FOLDERS**，通常用作 **SCI_SETMARGINMASKN** 定义用于折叠的边距的参数。

此表显示哪些 **SC_MARK_*** 符号应分配给哪些 **SC_MARKNUM_*** 标记号以获得四种折叠样式：箭头（模仿 Macintosh），加/减将折叠线显示为“+”，打开折叠为“-”，圆形树，框树。

SC_MARKNUM_*	箭头	正负	圆树	箱子树
--------------	----	----	----	-----

SC_MARKNUM_*	箭头	正负	圆树	箱子树
FOLDEROPEN	ARROWDOWN	MINUS	CIRCLEMINUS	BOXMINUS
FOLDER	ARROW	PLUS	CIRCLEPLUS	BOXPLUS
FOLDERSUB	EMPTY	EMPTY	VLINE	VLINE
FOLDERTAIL	EMPTY	EMPTY	LCORNERCURVE	LCORNER
FOLDEREND	EMPTY	EMPTY	CIRCLEPLUSCONNECTED	BOXPLUSCONNECTED
FOLDEROPENMID	EMPTY	EMPTY	CIRCLEMINUSCONNECTED	BOXMINUSCONNECTED
FOLDERMIDTAIL	EMPTY	EMPTY	TCORNERCURVE	TCORNER

	SC_MARK_CIRCLE,
	SC_MARK_ROUNDRECT,
	SC_MARK_SMALLRECT,
	SC_MARK_SHORTARROW,
	SC_MARK_BOOKMARK,
	SC_MARK_FULLRECT,
	SC_MARK_LEFTRECT,
	SC_MARK_BACKGROUND,
	SC_MARK_UNDERLINE,
...	SC_MARK_DOTDOTDOT,
	SC_MARK_ARROWS,
	SC_MARK_ARROW,
	SC_MARK_ARROWDOWN,
	SC_MARK_PLUS,
	SC_MARK_MINUS,
	SC_MARK_BOXPLUS,
	SC_MARK_BOXPLUSCONNECTED,
	SC_MARK_BOXMINUS,
	SC_MARK_BOXMINUSCONNECTED,
	SC_MARK_CIRCLEPLUS,
	SC_MARK_CIRCLEPLUSCONNECTED,
	SC_MARK_CIRCLEMINUS,
	SC_MARK_CIRCLEMINUSCONNECTED,
	SC_MARK_VLINE,
	SC_MARK_LCORNER,
	SC_MARK_TCORNER,
	SC_MARK_LCORNERCURVE,
	SC_MARK_TCORNERCURVE

SCI_MARKERDEFINEPIXMAP (int markerNumber, const char * pixmap)
 可以使用此消息将标记设置为 pixmaps。该 [XPM 格式](#)用于像素图。Pixmaps 使用 **SC_MARK_PIXMAP** 标记符号。

SCI_RGBAIMAGESETWIDTH (int width)

SCI_RGBAIMAGESETHEIGHT (int height)

SCI_RGBAIMAGESETSCALE (int scalePercent)

SCI_MARKERDEFINERGBAIMAGE (int markerNumber, const char * pixels)

可以使用此消息将标记设置为半透明像素图。该 [RGBA 格式](#)用于像素图。之前必须使用 **SCI_RGBAIMAGESETWIDTH** 和 **SCI_RGBAIMAGESETHEIGHT** 消息设置宽度和高度。

可以设置以百分比表示的比例因子 **SCI_RGBAIMAGESETSCALE**。这在具有视网膜显示的 OS X 上是有用的，其中每个显示单元是 2 个像素：使用因子 200，使得使用屏幕像素显示每个图像像素。默认比例 100 将拉伸每个图像像素以覆盖视网膜显示器上的 4 个屏幕像素。

Pixmap 使用 **SC_MARK_RGBAIMAGE** 标记符号。

SCI_MARKERSYMBOLDEFINED (int markerNumber) → int

返回与 markerNumber 定义的符号 **SCI_MARKERDEFINE** 或 **SC_MARK_PIXMAP** 如果与定义 **SCI_MARKERDEFINEPIXMAP** 或 **SC_MARK_RGBAIMAGE** 如果与定义 **SCI_MARKERDEFINERGBAIMAGE**。

SCI_MARKERSETFORE (int markerNumber, color fore)

SCI_MARKERSETBACK (int markerNumber, color back)

这两条消息设置标记号的前景色和背景色。

SCI_MARKERSETBACKSELECTED (int markerNumber, color back)

此消息设置选择折叠块时标记号的高亮背景颜色。默认颜色为 #FF0000。

SCI_MARKERENABLEHIGHLIGHT (启用 bool)

此消息允许在选中时启用/禁用高亮折叠块。（即包含插入符号的块）

SCI_MARKERSETALPHA (int markerNumber, alpha alpha)

当在内容区域中绘制标记时，或者因为它们没有边距或者它们是

SC_MARK_BACKGROUND 或 **SC_MARK_UNDERLINE** 类型，可以通过设置 alpha 值半透明地绘制它们。

SCI_MARKERADD (int line, int markerNumber) → int

此消息将标记号添加 *markerNumber* 到一行。如果失败（非法行号，内存不足），则消息返回 -1，或者返回标识添加的标记的标记句柄号。您可以使用此返回的句柄 **SCI_MARKERLINEFROMHANDLE** 来查找移动或组合线后标记的位置，以及 **SCI_MARKERDELETEHANDLE** 根据其句柄删除标记。该消息不检查 markerNumber 的值，也不检查该行是否已包含该标记。

SCI_MARKERADDDSET (int line, int markerSet)

此消息可以将一个或多个标记添加到具有单个调用的行，以相同的“每标记一位”32 位整数格式指定 **SCI_MARKERGET**（并由掩码使用）基于标记的搜索功能 **SCI_MARKERNEXT** 和 **SCI_MARKERPREVIOUS**）。与此同时 **SCI_MARKERADD**，不检查目标线上是否已存在任何标记。

SCI_MARKERDELETE (int line, int markerNumber)

这将在给定的行号中搜索给定的标记号，如果存在则删除它。如果您在该行中多次添加相同的标记，则每次使用时都会删除一个副本。如果传入标记号 -1，则从行中删除所有标记。

SCI_MARKERDELETEALL (int markerNumber)

这将从所有行中删除给定数字的标记。如果 `markerNumber` 为-1，则删除所有行中的所有标记。

SCI_MARKERGET (int line) →int

这将返回一个 32 位整数，指示该行上存在哪些标记。如果标记 0 存在，则设置位 0，标记 1 的位 1 设置，依此类推。

SCI_MARKERNEXT (int lineStart, int markerMask) →int

SCI_MARKERPREVIOUS (int lineStart, int markerMask) →int

这些消息有效地搜索包含给定标记集的行。搜索从行号开始，`lineStart` 继续向前到文件的末尾 (`SCI_MARKERNEXT`) 或向后到文件的开头

(`SCI_MARKERPREVIOUS`)。该 `markerMask` 参数应该有你想找到的每个标记一个位集。将位 0 设置为标记 0，将位 1 设置为标记 1，依此类推。该消息返回包含其中一个标记的第一行的行号，`markerMask` 如果未找到标记，则返回-1。

SCI_MARKERLINEFROMHANDLE (INT markerHandle) →INT

的 `markerHandle` 参数是用于通过返回标记的标识符 `SCI_MARKERADD`。此函数使用此句柄在文档中搜索标记，并返回包含它的行号，如果未找到，则返回-1。

SCI_MARKERDELETEHANDLE (INT markerHandle)

的 `markerHandle` 参数是用于通过返回标记的标识符 `SCI_MARKERADD`。此功能在文档中搜索带有此句柄的标记，并删除标记（如果找到）。

指标

指示符用于在样式顶部显示其他信息。例如，它们可以通过在文本或文本框周围绘制下划线来显示语法错误，不推荐使用的名称和不良缩进。

当鼠标悬停在指针上或插入符号中时，指示符可能具有不同的“悬停”颜色和样式。例如，这可以用于指示可以点击 URL。

指示器可以显示为简单的下划线，波浪形下划线，一条小“T”形状，一条斜线阴影线，一个敲击或文本周围的矩形。当用于跟踪应用程序的内容时，它们也可能是不可见的 `INDIC_HIDDEN`。

这些 `SCI_INDIC*`消息允许您获取和设置指标的视觉外观。它们都使用 `indicator0` 到 `INDIC_MAX` (35) 范围内的参数来将指标设置为样式。为了防止干扰，指标集被划分为一个范围供词法者使用 (0..7) 容器使用的范围 (8 = `INDIC_CONTAINER...` 31 = `INDIC_IME-1`) 和 `IME` 指标的范围 (32 = `INDIC_IME...` 35 = `INDIC_IME_MAX`)。

指示符以类似于行程长度编码的格式存储，这对于稀疏信息的速度和存储都是有效的。

指标可以为每个范围存储不同的值，但通常所有值都相同。所述 **SCI_INDICSETFLAGS** API 可被用于显示不同的值不同的颜色。

最初，Scintilla 使用了不同的指示器技术，但这已被[删除](#)，API 不执行[任何操作](#)。虽然这两种技术都得到了支持，但“现代指标”这一术语被用于较新的实施。

```
SCI_INDICSETSTYLE(int indicator, int indicatorStyle)
SCI_INDICGETSTYLE(int indicator) → int
SCI_INDICSETFORE(int indicator, colour fore)
SCI_INDICGETFORE(int indicator) → colour
SCI_INDICSETALPHA(int indicator, alpha alpha)
SCI_INDICGETALPHA(int indicator) → int
SCI_INDICSETOUTLINEALPHA(int indicator, alpha alpha)
SCI_INDICGETOUTLINEALPHA(int indicator) → int
SCI_INDICSETUNDER(int indicator, bool under)
SCI_INDICGETUNDER(int indicator) → bool
SCI_INDICSETHOVERSTYLE(int indicator, int indicatorStyle)
SCI_INDICGETHOVERSTYLE(int indicator) → int
SCI_INDICSETHOVERFORE(int indicator, colour fore)
SCI_INDICGETHOVERFORE(int indicator) → colour
SCI_INDICSETFLAGS(int indicator, int flags)
SCI_INDICGETFLAGS(int indicator) → int

SCI_SETINDICATORCURRENT(int indicator)
SCI_GETINDICATORCURRENT → int
SCI_SETINDICATORVALUE(int value)
SCI_GETINDICATORVALUE → int
SCI_INDICATORFILLRANGE(int start, int lengthFill)
SCI_INDICATORCLEARRANGE(int start, int lengthClear)
SCI_INDICATORALLONFOR(int pos) → int
SCI_INDICATORVALUEAT(int indicator, int pos) → int
SCI_INDICATORSTART(int indicator, int pos) → int
SCI_INDICATOREND(int indicator, int pos) → int
SCI_FINDINDICATORSHOW(int start, int end)
SCI_FINDINDICATORFLASH(int start, int end)
SCI_FINDINDICATORHIDE
```

SCI_INDICSETSTYLE (int indicator, int indicatorStyle)
SCI_INDICGETSTYLE (int indicator) →int

这两条消息设置并获取特定指标的样式。目前可用的指标样式是：

INDIC_PLAIN
INDIC_SQUIGGLE
INDIC_TT
INDIC_DIAGONAL
INDIC_STRIKE
INDIC_HIDDEN
INDIC_BOX
INDIC_ROUNDBOX
INDIC_STRAIGHTBOX
INDIC_FULLBOX
INDIC_DASH
INDIC_DOTS
INDIC_SQUIGGLELOW
INDIC_DOTBOX
INDIC_SQUIGGLEPIXMAP
INDIC_COMPOSITIONTHICK
INDIC_COMPOSITIONTHIN
INDIC_TEXTFORE
INDIC_POINT
INDIC_POINTCHARACTER

符号	值	视觉效果
INDIC_PLAIN	0	用一条直线加下划线。
INDIC_SQUIGGLE	1	一个波浪形的下划线。需要 3 个像素的下行空间。
INDIC_TT	2	一条小 T 形状的线条。
INDIC_DIAGONAL	3	对角线孵化。
INDIC_STRIKE	4	罢工。
INDIC_HIDDEN	五	没有视觉效果的指标。
INDIC_BOX	6	文本周围的矩形。
INDIC_ROUNDBOX	7	使用半透明绘图在文本周围使用圆角的矩形，内部通常比边框更透明。您可以使用 SCI_INDICSETALPHA 和 SCI_INDICSETOUTLINEALPHA 来控制 Alpha 透明度值。填充颜色的默认 alpha 值为 30，轮廓颜色为 50。
INDIC_STRAIGHTBOX	8	文本周围的矩形使用半透明绘图，内部通常比边框更透明。您可以使用 SCI_INDICSETALPHA 和 SCI_INDICSETOUTLINEALPHA 来控制 Alpha 透明度值。填充颜色的默认 alpha 值为 30，轮廓颜色为 50。此指示器不会对线条的顶部像素着色，以使连续线条上的指示符在视觉上不同并断开连接。
INDIC_FULLBOX	16	文本周围的矩形使用半透明绘图，类似

		INDIC_STRAIGHTBOX 但覆盖整个角色区域。
INDIC_DASH	9	虚线下划线。
INDIC_DOTS	10	虚线下划线。
INDIC_SQUIGGLELOW	11	类似 INDIC_SQUIGGLE 但仅使用 2 个垂直像素，因此适合小字体。 使用半透明绘图围绕文本的虚线矩形。半透明使用 alpha 设置在左上角像素和字母 alpha 设置之间切换。 SCI_INDICSETALPHA 和
INDIC_DOTBOX	12	SCI_INDICSETOUTLINEALPHA 控制 Alpha 透明度值。 alpha 的默认值为 30，大纲 alpha 的默认值为 50。为避免过多的内存分配，虚线框的最大宽度为 4000 像素。
INDIC_GRADIENT	20	顶部颜色和 alpha 之间的垂直渐变，底部完全透明。
INDIC_GRADIENTCENTRE	21	具有指定颜色的垂直渐变和中间的 alpha 渐变，在顶部和底部完全透明。
INDIC_SQUIGGLEPIXMAP	13	它的一个版本 INDIC_SQUIGGLE 使用像素图而不是一系列线段来绘制性能。测得比 INDIC_SQUIGGLEGTK + 快 3 到 6 倍。外观不如 INDIC_SQUIGGLEHiDPI 模式下的 OS X 好。
INDIC_COMPOSITIONTHICK	14	位于线条底部的 2 像素厚的下划线，以避免接触字符基础。每一侧都插入 1 个像素，以便覆盖范围的这种风格的不同指示符显得孤立。这类似于亚洲语言输入组合中用于目标的外观。
INDIC_COMPOSITIONTHIN	15	1 像素厚的下划线位于线的底部之前。每一侧都插入 1 个像素，以便覆盖范围的这种风格的不同指示符显得孤立。这类似于亚洲语言输入组合中用于非目标范围的外观。
INDIC_TEXTFORE	17	将文本的颜色更改为指示符的前色。
INDIC_POINT	18	在指标范围的开头下方画一个三角形。
INDIC_POINTCHARACTER	19	在指标范围的第一个字符的中心下方绘制一个三角形。

默认指标样式相当于：

```
SCI_INDICSETSTYLE(0, INDIC_SQUIGGLE);
SCI_INDICSETSTYLE(1, INDIC_TT);
SCI_INDICSETSTYLE(2, INDIC_PLAIN);
```

SCI_INDICSETFORE (int indicator, color
fore) **SCI_INDICGETFORE** (int indicator) →color

这两条消息设置并获取用于绘制指标的颜色。默认指示灯颜色相当于:(

```
SCI_INDICSETFORE(0, 0x007f00);深绿色)
```

SCI_INDICSETFORE(1, 0xff0000); (浅蓝色)

SCI_INDICSETFORE(2, 0x0000ff); (浅红色)

SCI_INDICSETALPHA (int indicator, alpha alpha)

SCI_INDICGETALPHA (int indicator) →int

这两条消息设置并获取用于绘制 INDIC_ROUNDBOX 和 INDIC_STRAIGHTBOX 矩形的填充颜色的 alpha 透明度。alpha 值的范围可以从 0 (完全透明) 到 255 (无透明度)。

SCI_INDICSETOUTLINEALPHA (int indicator, alpha alpha)

SCI_INDICGETOUTLINEALPHA (int indicator) →int

这两条消息设置并获取用于绘制 INDIC_ROUNDBOX 和 INDIC_STRAIGHTBOX 矩形的轮廓颜色的 alpha 透明度。alpha 值的范围可以从 0 (完全透明) 到 255 (无透明度)。

SCI_INDICSETUNDER (int indicator, bool under)

SCI_INDICGETUNDER (int indicator) →bool

这两条消息设置并获取指标是在文本下绘制还是在 (默认) 下绘制。在文本下绘图不适用于不推荐的 [单相绘图](#) 模式。

SCI_INDICSETHOVERSTYLE (int indicator, int indicatorStyle)

SCI_INDICGETHOVERSTYLE (int indicator) →int

SCI_INDICSETHOVERFORE (int indicator, color fore) **SCI_INDICGETHOVERFORE** (int indicator) →color

这些消息设置并获取鼠标悬停时用于绘制指示符的颜色和样式插入符号插入其中。在悬停样式中绘制指标时，鼠标光标也会发生变化。默认情况下，悬停外观与正常外观相同，并且调用 [SCI_INDICSETFORE](#) 或 [SCI_INDICSETSTYLE](#) 也将重置悬停属性。

SCI_INDICSETFLAGS (int indicator, int flags)

SCI_INDICGETFLAGS (int indicator) →int

这些消息设置并获取与指标相关的标志。目前定义了一个标志

SC_INDICFLAG_VALUEFORE: 当设置此标志时，指示器使用的颜色不是来自指示器的前置设置，而是来自文件中该点的指示器值。这允许为单个指示器显示许多颜色。该值是 [RGB 整数颜色](#)，**SC_INDICVALUEBIT** 在调用

[SCI_SETINDICATORVALUE](#) 时已经使用 (0x1000000) 进行了 [存储](#)。要从值中查找颜色，并使用 **SC_INDICVALUEMASK** (0xFFFFFFFF) 查找值。

SCI_SETINDICATORCURRENT (int indicator)

SCI_GETINDICATORCURRENT →int

这两条消息设置并获取将受 [SCI_INDICATORFILLRANGE](#) (int start, int lengthFill) 和 [SCI_INDICATORCLEARRANGE](#) (int start, int lengthClear) 调用影响的指标。

SCI_SETINDICATORVALUE (int value)

SCI_GETINDICATORVALUE→int

这两条消息设置并获取将通过调用 **SCI_INDICATORFILLRANGE** 设置的值。

SCI_INDICATORFILLRANGE (int start, int lengthFill)

SCI_INDICATORCLEARRANGE (int start, int lengthClear)

这两条消息填充或清除当前指标的范围。 **SCI_INDICATORFILLRANGE** 填写当前值。

SCI_INDICATORALLONFOR (int pos) →int

检索位图值，表示某个位置的哪些指标非零。结果中仅显示前 32 个指标，因此不包括 IME 指标。

SCI_INDICATORVALUEAT (int indicator, int pos) →int

检索某个位置的特定指标的值。

SCI_INDICATORSTART (int indicator, int pos) →int

SCI_INDICATOREND (int indicator, int pos) →int

使用范围内位置的一个值查找范围的开始或结束。可用于迭代文档以发现所有指标位置。

OS X 查找指标

在 OS X 上，搜索匹配以动画金色圆角矩形突出显示。指示器显示，然后短暂增长 25% 并缩小到原始大小以吸引用户的注意力。虽然此功能目前仅在 OS X 上实现，但它可能在将来在其他平台上实现。

SCI_FINDINDICATORSHOW (int start, int end)

SCI_FINDINDICATORFLASH (int start, int end)

这两条消息显示并为 find 指标设置动画。 **SCI_FINDINDICATORSHOW** 在显示半秒后，指示器仍然可见并淡出 **SCI_FINDINDICATORFLASH**。 **SCI_FINDINDICATORSHOW** 其行为与 OS X TextEdit 和 Safari 应用程序类似，最适合编辑搜索目标通常是单词的文档。 **SCI_FINDINDICATORFLASH** 类似于 Xcode，适用于编辑源代码，其中匹配通常位于运算符旁边，否则将隐藏在指标的填充下。

SCI_FINDINDICATORHIDE

此消息隐藏查找指示符。

早期版本的 Scintilla 允许在样式编号和指示符之间[划分样式字节](#)，并提供用于设置和查询它的 API。

自动完成

自动填充显示一个列表框，根据用户的输入显示可能的标识符。用户通过按 Tab 键字符或作为定义的填充字符集成员的另一个字符来选择当前选择的项目 **SCI_AUTOSETFILLUPS**。您的应用程序会触发自动完成功能。例如，在 C 中，如

果您检测到用户刚刚键入了 **fred**。您可以查找 **fred**，并且如果它具有已知的成员列表，则可以在自动完成列表中提供它们。或者，您可以监控用户的输入，并在他们的输入将选择范围缩小到合理列表后提供可能的项目列表。作为另一种选择，您可以定义一个密钥代码来激活列表。

当用户从列表中进行选择时，向容器发送 **SCI_AUTOCELECTION** 通知消息。从通知返回时，Scintilla 将插入所选文本，并向容器发送 **SCI_AUTOCCOMPLETED** 通知消息，除非已取消自动完成列表，例如通过容器发送 **SCI_AUTOCCANCEL**。

要使用自动完成功能，您必须监视添加到文档中的每个字符。有关 **SciTEBase::CharAdded()** 自动完成的示例，请参阅 **SciTEBase.cxx**。

```
SCI_AUTOCSHOW(int lengthEntered, const char *itemList)
SCI_AUTOCCANCEL
SCI_AUTOCACTIVE → bool
SCI_AUTOCPSTART → position
SCI_AUTOCCOMplete
SCI_AUTOCESTOP(<unused>, const char *characterSet)
SCI_AUTOCESETSEPARATOR(int separatorCharacter)
SCI_AUTOCEGETSEPARATOR → int
SCI_AUTOCESELECT(<unused>, const char *select)
SCI_AUTOCEGETCURRENT → int
SCI_AUTOCEGETCURRENTTEXT(<unused>, char *text) → int
SCI_AUTOCESETCANCELATSTART(bool cancel)
SCI_AUTOCEGETCANCELATSTART → bool
SCI_AUTOCESETFILLUPS(<unused>, const char *characterSet)
SCI_AUTOCESETCHOOSESINGLE(bool chooseSingle)
SCI_AUTOCEGETCHOOSESINGLE → bool
SCI_AUTOCESETIGNORECASE(bool ignoreCase)
SCI_AUTOCEGETIGNORECASE → bool
SCI_AUTOCESETCASEINSENSITIVEBEHAVIOUR(int behaviour)
SCI_AUTOCEGETCASEINSENSITIVEBEHAVIOUR → int
SCI_AUTOCESETMULTI(int multi)
SCI_AUTOCEGETMULTI → int
SCI_AUTOCESETORDER(int order)
SCI_AUTOCEGETORDER → int
SCI_AUTOCESETAUTOHIDE(bool autoHide)
SCI_AUTOCEGETAUTOHIDE → bool
SCI_AUTOCESETDROPRESTOFWORD(bool dropRestOfWord)
SCI_AUTOCEGETDROPRESTOFWORD → bool
SCI_REGISTERIMAGE(int type, const char *xpmData)
SCI_REGISTERRGBAIMAGE(int type, const char *pixels)
SCI_CLEARREGISTEREDIMAGES
SCI_AUTOCESETTYPESEPARATOR(int separatorCharacter)
SCI_AUTOCEGETTYPESEPARATOR → int
SCI_AUTOCESETMAXHEIGHT(int rowCount)
SCI_AUTOCEGETMAXHEIGHT → int
SCI_AUTOCESETMAXWIDTH(int characterCount)
SCI_AUTOCEGETMAXWIDTH → int
```

SCI_AUTOCSHOW (int lengthEntered, const char * itemList)

此消息导致显示列表。*lengthEntered* 是已输入单词的字符数，*itemList* 是由分隔符分隔的单词列表。初始分隔符是一个空格，但可以设置或使用 **SCI_AUTOCESETSEPARATOR** 和 **SCI_AUTOCEGETSEPARATOR**。

使用默认设置时，单词列表应按排序顺序排列。如果设置为忽略大小写模式 [SCI_AUTOCSETIGNORECASE](#)，则在转换为大写后匹配字符串。这样做的一个结果是列表应该用标点字符'[', '\n', ']', '^', '_'和'"'排序，并在字母后排序。可以使用 [SCI_AUTOCSETORDER](#) 指定列表顺序的替代处理

SCI_AUTOCCANCEL

此消息取消任何显示的自动完成列表。当处于自动完成模式时，当用户键入不能成为自动完成的一部分的字符时，列表应该消失，例如'.'，'('或'"'在键入标识符时。一组将取消自动完成的字符可以指定 [SCI_AUTOCSTOPS](#)。

SCI_AUTOCACTIVE→bool

如果存在活动的自动完成列表，则此消息返回非零，如果不存在，则返回零。

SCI_AUTOCPOSSTART→position

当 [SCI_AUTOCSHOW](#) 开始显示列表时，返回当前位置的值。

SCI_AUTOCCOMplete

此消息触发自动完成。这与 tab 键具有相同的效果。

SCI_AUTOCSTOPS (<未使用>, 常量字符* CHARACTERSET)

的 *characterSet* 参数是包含将自动取消自动完成列表的字符列表的字符串。启动编辑器时，此列表为空。

SCI_AUTOCSETSEPARATOR (int separatorCharacter)

SCI_AUTOCGETSEPARATOR→int

这两条消息设置并获取用于分隔 [SCI_AUTOCSHOW](#) 列表中单词的分隔符。默认值为空格字符。

SCI_AUTOCSELECT (<unused>, const char * select)

SCI_AUTOCGETCURRENT→int

此消息选择自动完成列表中的项目。它搜索匹配的第一个单词列表 *select*。默认情况下，比较区分大小写，但您可以使用更改 [SCI_AUTOCSETIGNORECASE](#)。匹配是 *select* 字符串长度的字符。也就是说，如果 *select* 是“Fred”，它将匹配“Frederick”，如果这是列表中以“Fred”开头的第一个项目。如果找到某个项目，则会选中该项目。如果找不到该项，则自动填充列表会在 *auto-hide* 为 *true* 时关闭（请参阅参考资料 [SCI_AUTOCSETAUTOHIDE](#)）。

可以使用检索当前选择索引 [SCI_AUTOCGETCURRENT](#)。

SCI_AUTOCGETCURRENTTEXT (<unused>, char * text NUL-terminated)→int

此消息检索自动完成列表中的当前所选文本。通常使用 [SCI_AUTOCSELECTION](#) 通知。

该值被复制到 *text* 缓冲区，返回长度（不包括终止 0）。如果未找到，则将空字符串复制到缓冲区并返回 0。

如果 `value` 参数为 0，则返回应分配用于存储该值的长度；再次，不包括终止 0。

SCI_AUTOCSETCANCELATSTART (bool 取消)

SCI_AUTOCGETCANCELATSTART→bool

如果插入符号移动到显示列表时的位置，则默认行为是取消列表。通过使用 `false` 参数调用此消息，直到插入符号在完成单词之前移动至少一个字符时才会取消列表。

SCI_AUTOCSETFILLUPS (<unused>, const char * characterSet)

如果在激活自动填充列表的情况下键入填充字符，则会将列表中当前选定的项目添加到文档中，然后添加填充字符。常见的填充字符是 '(', '[' 和 '.'，但其他字符可能取决于语言。默认情况下，不设置填充字符。

SCI_AUTOCSETCHOOSESINGLE (bool chooseSingle)

SCI_AUTOCGETCHOOSESINGLE→bool

如果使用 **SCI_AUTOCSETCHOOSESINGLE(1)** 且列表只有一个项目，则会自动添加，不显示任何列表。即使只有一个项目，默认也是显示列表。

SCI_AUTOCSETIGNORECASE (bool ignoreCase)

SCI_AUTOCGETIGNORECASE→bool

默认情况下，字符与列表成员的匹配区分大小写。这些消息可让您设置并获得区分大小写。

SCI_AUTOCSETCASEINSENSITIVEBEHAVIOUR (int 行为)

SCI_AUTOCGETCASEINSENSITIVEBEHAVIOUR→int

当自动完成设置为忽略 case (**SCI_AUTOCSETIGNORECASE**) 时，默认情况下它仍会选择匹配的列表成员，以区分大小写的方式输入字符。这对应于 **SC_CASEINSENSITIVEBEHAVIOUR_RESPECTCASE (0)** 的行为属性。如果您希望自动完成忽略大小写，请选择 **SC_CASEINSENSITIVEBEHAVIOUR_IGNORECASE (1)**。

SCI_AUTOCSETMULTI (int multi)

SCI_AUTOCGETMULTI→int

当存在多个选择的自动完成时，自动完成的文本可以仅使用 **SC_MULTIAUTOC_ONCE (0)** 进入主选择，或使用 **SC_MULTIAUTOC_EACH (1)** 进入每个选择。默认是 **SC_MULTIAUTOC_ONCE**。

SCI_AUTOCSETORDER (int order)

SCI_AUTOCGETORDER→int

默认设置 **SC_ORDER_PRESORTED (0)** 要求按字母顺序提供列表。

对列表进行排序可以通过 Scintilla 而不是带有 **SC_ORDER_PERFORMSORT (1)** 的应用程序来完成。这将需要额外的时间。

希望优先考虑某些值并按优先级顺序而不是按字母顺序显示列表的应用程序可以使用 **SC_ORDER_CUSTOM (2)**。这需要在 **SCI_AUTOCSHOW** 中进行额外处理以创建排序索引。

应在调用 **SCI_AUTOCSHOW** 之前完成设置顺序。

SCI_AUTOCSETAUTOHIDE (bool autoHide)

SCI_AUTOCGETAUTOHIDE→bool

默认情况下，如果没有可行匹配（用户键入的字符不再与列表条目匹配），则会取消列表。如果要继续显示原始列表，请设置 *autoHide* 为 **false**。这也有影响 **SCI_AUTOCSELECT**。

SCI_AUTOCSETDROPRESTOFWORD (bool dropRestOfWord)

SCI_AUTOCGETDROPRESTOFWORD→bool

选择项目后，如果 *dropRestOfWord* 设置了插入符号后面的任何单词字符将首先被删除 **true**。默认是 **false**。

SCI_REGISTERIMAGE (int type, const char * xpmData)

SCI_REGISTERRGBAIMAGE (int type, const char * pixels)

SCI_CLEARREGISTEREDIMAGES

SCI_AUTOCSETTYPESEPARATOR (int separatorCharacter)

SCI_AUTOCGETTYPESEPARATOR→int

自动完成列表项可以显示图像和文本。每个图像首先以整数类型注册。然后这个整数包含在由 '?' 分隔的列表文本中 从文本。例如，“fclose? 2 fopen”在字符串“fclose”之前显示图像 2，在“fopen”之前没有图像。图像采用 **XPM 格式**（**SCI_REGISTERIMAGE**）或 **RGBA 格式**（**SCI_REGISTERRGBAIMAGE**）。

SCI_REGISTERRGBAIMAGES**SCI_RGBAIMAGESETWIDTH** 和 **SCI_RGBAIMAGESETHEIGHT** 消息。

注册图像集可以用 **SCI_CLEARREGISTEREDIMAGES** '?' 清除 分隔符改变了

SCI_AUTOCSETTYPESEPARATOR。

SCI_AUTOCSETMAXHEIGHT (int rowCount)

SCI_AUTOCGETMAXHEIGHT→int

获取或设置自动完成列表中可见的最大行数。如果列表中有更多行，则会显示垂直滚动条。默认值为 5。

SCI_AUTOCSETMAXWIDTH (int characterCount)

SCI_AUTOCGETMAXWIDTH→int

获取或设置自动完成列表的最大宽度，表示为将在完全可见的最长项目中的字符数。如果为零（默认值），则计算列表的宽度以适合具有最多字符的项目。任何无法在可用宽度内完全显示的项目都由省略号表示。

用户列表

用户列表使用与自动完成列表相同的内部机制，并且为自动完成列出的所有调用都对它们起作用；您无法在自动填充列表处于活动状态的同时显示用户列表。它们在以下方面有所不同：

SCI_USERLISTSHOW(int listType, const char *itemList)

- o 该 `SCI_AUTOCSETCHOOSESINGLE` 消息无效。
- o 当用户进行选择时，会向您发送 `SCN_USERLISTSELECTION` 通知消息而不是 `SCN_AUTOCELECTION`。

请注意：如果您设置了填充字符或停止字符，则这些字符仍将在用户列表中处于活动状态，并且可能会导致选择项目或用户列表因用户键入编辑器而被取消。

SCI_USERLISTSHOW (int listType, const char * itemList)

该 *listType* 参数作为结构的 `wParam` 字段返回给容器 `SCNotification`。它必须大于 0，因为这就是 Scintilla 告诉自动完成列表和用户列表之间的区别。如果您有不同类型的列表，例如缓冲区列表和宏列表，您可以使用它 *listType* 来判断哪个列表已返回选择。

致电提示

调用提示是显示函数参数的小窗口，在用户键入函数名称后显示。它们通常使用由其定义的字体名称，大小和字符集来显示字符 `STYLE_DEFAULT`。您可以选择使用 `STYLE_CALLTIP` 来定义面名，大小，前景和背景颜色以及字符集 `SCI_CALLTIPUSESTYLE`。这也支持 Tab 字符。呼叫提示和自动完成列表之间存在一些交互，显示呼叫提示取消任何活动的自动完成列表，反之亦然。

Qt 上没有实现通话提示。

通话提示可以突出显示其中的部分文字。您可以使用此方法通过计算逗号（或您的语言使用的任何分隔符）的数量来突出显示函数的当前参数。看到 `SciTEBase::CharAdded()` 在 `SciTEBase.cxx` 用于呼叫提示使用的一个例子。

可以在呼叫提示上单击鼠标，这会导致将 `SCN_CALLTIPCLICK` 通知发送到容器。小的向上和向下箭头可以分别在呼叫提示中显示，包括字符 `\ 001'` 或 `\ 002'`。这对于显示一个函数名称的重载变体很有用，并且用户可以单击箭头来循环重载。

或者，当您将鼠标指针停留在单词上一段时间以响应 `SCN_DWELLSTART` 通知时，可以显示呼叫提示，并在响应时取消 `SCN_DWELLEND`。此方法可以在调试器中用于给出变量的值，或者在编辑期间提供有关指针下的单词的信息。

```
SCI_CALLTIPSHOW(int pos, const char *definition)
SCI_CALLTIPCANCEL
SCI_CALLTIPACTIVE → bool
SCI_CALLTIPPOSSTART → position
SCI_CALLTIPSETPOSSTART(int posStart)
SCI_CALLTIPSETHLT(int highlightStart, int highlightEnd)
SCI_CALLTIPSETBACK(colour back)
SCI_CALLTIPSETFORE(colour fore)
SCI_CALLTIPSETFOREHLT(colour fore)
SCI_CALLTIPUSESTYLE(int tabSize)
SCI_CALLTIPSETPOSITION(bool above)
```

SCI_CALLTIPSHOW (int pos, const char * definition)

此消息通过显示调用提示窗口来启动该过程。如果呼叫提示已处于活动状态，则无效。

pos 是文档中与调用提示对齐的位置。调用提示文本对齐以在此字符下方开始 1 行，除非您在调用提示文本中包含向上和/或向下箭头，在这种情况下，提示与最右侧箭头的右侧边缘对齐。假设您将使用“\ 001 1 of 3 \ 002”之类的内容启动文本。

definition 是通话提示文字。这可以包含由“\ n”（换行符，ASCII 代码 10）字符分隔的多行。不要包含“\ r”（回车符，ASCII 代码 13），因为这很可能打印为空框。如果您设置了标签大小，则支持“\ t”（Tab，ASCII 代码 9）

SCI_CALLTIPUSESTYLE。

这里记住插入符号的位置，以便如果后续删除在该位置之前移动插入符号，则可以自动取消呼叫提示。

SCI_CALLTIPCANCEL

此消息取消任何显示的呼叫提示。如果您使用与编辑函数的参数列表不兼容的任何键盘命令，Scintilla 也将取消您的调用提示。如果您在触发提示时删除了超过插入符号的位置，则会取消呼叫提示。

SCI_CALLTIPACTIVE→bool

如果呼叫提示处于活动状态，则返回 1，如果未激活，则返回 0。

SCI_CALLTIPPOSSTART→位置

SCI_CALLTIPSETPOSSTART (int posStart)

此消息在 **SCI_CALLTIPSHOW** 开始显示提示时返回或设置当前位置的值。

SCI_CALLTIPSETHLT (int highlightStart, int highlightEnd)

这将设置调用提示文本的区域以突出显示的样式显示。**highlightStart** 是要突出显示的第一个字符的字符串的从零开始的索引，并且 **highlightEnd** 是突出显示后第一个字符的索引。**highlightEnd** 必须大于 **highlightStart**；**highlightEnd-highlightStart** 是要突出显示的字符数。如果需要，亮点可以延伸到线端。

未突出显示的文本以中灰色绘制。选定的文字以深蓝色绘制。背景是白色的。这些可以被改变 **SCI_CALLTIPSETBACK**， **SCI_CALLTIPSETFORE** 和 **SCI_CALLTIPSETFOREHLT**。

SCI_CALLTIPSETBACK (回颜色)

可以使用此消息设置呼叫提示的背景颜色；默认颜色为白色。将深色设置为背景并不是一个好主意，因为正常的提示文本的默认颜色是中灰色，高亮文本的默认颜色是深蓝色。这也设置了背景颜色 **STYLE_CALLTIP**。

SCI_CALLTIPSETFORE (颜色前)

可以使用此消息设置呼叫提示文本的颜色；默认颜色为中灰色。这也设置了前景色 **STYLE_CALLTIP**。

SCI_CALLTIPSETFOREHLT（颜色前）

可以使用此消息设置突出显示的呼叫提示文本的颜色; 默认颜色为深蓝色。

SCI_CALLTIPUSESTYLE（int tabSize）

此消息将用于调用提示的样式更改为 `STYLE_DEFAULT`，`STYLE_CALLTIP` 并以屏幕像素为单位设置选项卡大小。如果 `tabSize` 小于 1，则不会特别处理制表符。使用此调用后，呼叫提示前景色和背景色也将从样式中获取。

SCI_CALLTIPSETPOSITION（上面的 bool）

默认情况下，`calltip` 显示在文本下方，上面的设置 `true`（1）将显示在文本上方。

键盘命令

为了允许容器应用程序使用键盘执行用户可用的任何操作，所有键盘操作都是消息。他们不接受任何参数。在使用 `SCI_ASSIGNCMDKEY` 消息重新定义键绑定时也会使用这些命令。

SCI_LINEDOWN	SCI_LINEDOWNNEXTEND	SCI_LINEDOWNRECTEXTEND	SCI_LINESCROLLDOWN
SCI_LINEUP	SCI_LINEUPNEXTEND	SCI_LINEUPRECTEXTEND	SCI_LINESCROLLUP
SCI_PARADOWN	SCI_PARADOWNNEXTEND	SCI_PARAUP	SCI_PARAUPNEXTEND
SCI_CHARLEFT	SCI_CHARLEFTTEXTEND	SCI_CHARLEFTRECTEXTEND	
SCI_CHARRIGHT	SCI_CHARRIGHTTEXTEND	SCI_CHARRIGHTRECTEXTEND	
SCI_WORDLEFT	SCI_WORDLEFTTEXTEND	SCI_WORDRIGHT	SCI_WORDRIGHTTEXTEND
SCI_WORDLEFTEND	SCI_WORDLEFTTEXTEND	SCI_WORDRIGHTEND	SCI_WORDRIGHTTEXTEND
SCI_WORDPARTLEFT	SCI_WORDPARTLEFTTEXTEND	SCI_WORDPARTRIGHT	SCI_WORDPARTRIGHTTEXTEND
SCI_HOME	SCI_HOMEEXTEND	SCI_HOMERECTEXTEND	
SCI_HOMEDISPLAY	SCI_HOMEDISPLAYEXTEND	SCI_HOMEWRAPEXTEND	SCI_HOMEDISPLAYEXTEND
SCI_VCHOME	SCI_VCHOMEEXTEND	SCI_VCHOMERECTEXTEND	
SCI_VCHOMEWRAP	SCI_VCHOMEWRAPNEXTEND	SCI_VCHOMEDISPLAY	SCI_VCHOMEDISPLAYEXTEND
SCI_LINEEND	SCI_LINEENDNEXTEND	SCI_LINEENDRECTEXTEND	
SCI_LINEENDDISPLAY	SCI_LINEENDDISPLAYEXTEND	SCI_LINEENDWRAP	SCI_LINEENDWRAPNEXTEND
SCI_DOCUMENTSTART	SCI_DOCUMENTSTARTTEXTEND	SCI_DOCUMENTEND	SCI_DOCUMENTENDTEXTEND
SCI_PAGEUP	SCI_PAGEUPNEXTEND	SCI_PAGEUPRECTEXTEND	
SCI_PAGEDOWN	SCI_PAGEDOWNNEXTEND	SCI_PAGEDOWNRECTEXTEND	
SCI_STUTTEREDPAGEUP	SCI_STUTTEREDPAGEUPNEXTEND		
SCI_STUTTEREDPAGEDOWN	SCI_STUTTEREDPAGEDOWNNEXTEND		
SCI_DELETEBACK	SCI_DELETEBACKNOTLINE		
SCI_DELWORDLEFT	SCI_DELWORDRIGHT	SCI_DELWORDRIGHTEND	
SCI_DELLINELEFT	SCI_DELLINERIGHT	SCI_LINEDELETE	SCI_LINECUT
SCI_LINECOPY	SCI_LINETRANSPOSE	SCI_LINEREVERSE	SCI_LINEDUPLICATE
SCI_LOWERCASE	SCI_UPPERCASE	SCI_CANCEL	SCI_EDITTOGGLEOVER
SCI_NEWLINE	SCI_FORMFEED	SCI_TAB	SCI_BACKTAB
SCI_SELECTIONDUPLICATE	SCI_VERTICALCENTRECARET		
SCI_MOVESELECTEDLINESUP	SCI_MOVESELECTEDLINESDOWN		
SCI_SCROLLTOSTART	SCI_SCROLLTOEND		

的 `SCI_*EXTEND` 消息扩展选择。

的 `SCI_*RECTEXTEND` 消息扩展矩形选择（和转换规则选择到长方形一个，如果有的话）。

这些 `SCI_WORDPART*` 命令用于在大写标记（`aCamelCaseIdentifier`）或下划线（`an_under_bar_ident`）标记的单词段之间移动。

这些 `SCI_WORD[LEFT|RIGHT]END*` 命令类似于 `SCI_WORD[LEFT|RIGHT]*` 但是在单词结束而不是单词开始之间移动。

该 `SCI_HOME*` 命令将移动插入符行的开始，而 `SCI_VCHOME*` 命令移动插入符号到行（即刚好压痕之后），除非它已经处在那里的第一非空白字符；在这种情况下，它充当 `SCI_HOME*`。

这些 `SCI_[HOME|LINEEND]DISPLAY*` 命令在换行模式下使用，以允许移动到显示行的开头或结尾，而不是 `SCI_[HOME|LINEEND]` 移动到文档行的开头或结尾的常规命令。

这些 `SCI_[[VC]HOME|LINEEND]WRAP*` 命令就像它们的名字一样，`SCI_[[VC]HOME|LINEEND]*` 除非它们在启用自动换行时表现不同：它们首先进入显示行的开始/结束 `SCI_[HOME|LINEEND]DISPLAY*`，但是如果光标已经在该点，它将继续到开始或结束适用的文件行 `SCI_[[VC]HOME|LINEEND]*`。

该 `SCI_SCROLLTO[START|END]` 命令滚动文档的开始或结束，而不改变该选择。这些命令与 OS X 平台约定匹配 `home` 和 `end` 键的行为。通过绑定这些命令的 `home` 和 `end` 键，可以使 Scintilla 与 OS X 应用程序匹配。

该 `SCI_CANCEL` 命令取消自动完成和呼叫提示显示并删除任何其他选择。

键绑定

对于命令的键的默认绑定是 `KeyMap.cxx` 由常量在文件中的 Scintilla 源中定义的 `KeyMap::MapDefault[]`。此表将键定义映射到 `SCI_*` 没有参数的消息（主要是上面讨论的 **键盘命令**，但可以映射任何没有参数的 Scintilla 命令）。您可以更改映射以满足您自己的要求。

```
SCI_ASSIGNCMDKEY(int keyDefinition, int sciCommand)
SCI_CLEARCMDKEY(int keyDefinition)
SCI_CLEARALLCMDKEYS
SCI_NULL
```

keyDefinition

键定义包含低 16 位的密钥代码和高 16 位的密钥修饰符。组合 `keyCode` 和 `keyMod` 设置：

```
keyDefinition = keyCode + (keyMod << 16)
```

键控代码是可见光或控制字符或从一个键 `SCK_*` 的枚举，它包含：
`SCK_ADD`, `SCK_BACK`, `SCK_DELETE`, `SCK_DIVIDE`, `SCK_DOWN`, `SCK_END`, `SCK_ESCAPE`,
`SCK_HOME`, `SCK_INSERT`, `SCK_LEFT`, `SCK_MENU`, `SCK_NEXT` (下页), `SCK_PRIOR`
(页上), `SCK_RETURN`, `SCK_RIGHT`, `SCK_RWIN`, `SCK_SUBTRACT`, `SCK_TAB`,
`SCK_UP` 和 `SCK_WIN`。

所述改性剂是零个或多个的组合 `SCMOD_ALT`, `SCMOD_CTRL`, `SCMOD_SHIFT`,
`SCMOD_META`, 和 `SCMOD_SUPER`。在 OS X 上, `Command` 键映射到 `SCMOD_CTRL`,
`Control` 键映射到 `SCMOD_META`。 `SCMOD_SUPER` 仅适用于 GTK +, 它通常是
Windows 密键。如果要构建表, 则可能需要使用 `SCMOD_NORM` 值为 0 的表, 而不
是表示修饰符。

SCI_ASSIGNCMDKEY (int keyDefinition, int sciCommand)
这将给定的键定义分配给标识的 Scintilla 命令 `sciCommand`。 `sciCommand` 可以是任
何 `SCI_*` 没有参数的命令。

SCI_CLEARCMDKEY (int keyDefinition)
这使得给定的键定义不会通过 `SCI_NULL` 为其分配操作来执行任何操作。

SCI_CLEARALLCMDKEYS
此命令通过设置空映射表来删除所有键盘命令映射。

SCI_NULL
的 `SCI_NULL` 什么也不做, 是分配给执行任何操作键的值。 `SCI_NULL` 确保键不
会传播到父窗口, 因为这可能导致焦点移动。如果您想要标准平台行为, 请使用
常量 0。

弹出编辑菜单

SCI_USEPOPUP (int popUpMode)

SCI_USEPOPUP (int popUpMode)
单击鼠标上的错误按钮会弹出一个简短的默认编辑菜单。这可能会被关闭
`SCI_USEPOPUP(SC_POPUP_NEVER)`。如果将其关闭, `WM_CONTEXTMENU` Scintilla 将不会
处理上下文菜单命令 (在 Windows 中), 因此 Scintilla 窗口的父级将有机会处
理该消息。

符号	值	含义
<code>SC_POPUP_NEVER</code>	0	从不显示默认编辑菜单。
<code>SC_POPUP_ALL</code>	1	如果单击 scintilla, 则显示默认编辑菜单。
<code>SC_POPUP_TEXT</code>	2	仅在单击文本区域时显示默认编辑菜单。

宏录制

启动和停止宏录制模式。在宏录制模式下，通过 **SCI_MACRORECORD** 通知将操作报告给容器。然后由容器记录这些操作以供将来重播。

SCI_STARTRECORD
SCI_STOPRECORD

SCI_STARTRECORD
SCI_STOPRECORD

这两条消息可以打开和关闭宏录制。

印花

SCI_FORMATRANGE 可用于将文本绘制到显示表面上，该显示表面可包括打印机显示表面。打印输出在屏幕上显示文本样式，但它隐藏了除行号边距之外的所有边距。删除所有特殊标记效果，隐藏选择和插入符号。

不同平台使用不同的显示表面 ID 类型进行打印。在 Windows 上，这些是 **HDC** 在 GTK + 3.x 上 **cairo_t ***，并且在 Cocoa **CGContextRef** 上使用。

SCI_FORMATRANGE(bool draw, Sci_RangeToFormat *fr) → position
SCI_SETPRINTMAGNIFICATION(int magnification)
SCI_GETPRINTMAGNIFICATION → int
SCI_SETPRINTCOLOURMODE(int mode)
SCI_GETPRINTCOLOURMODE → int
SCI_SETPRINTWRAPMODE(int wrapMode)
SCI_GETPRINTWRAPMODE → int

SCI_FORMATRANGE (bool draw, Sci_RangeToFormat * fr) →position

此调用将一系列文本呈现到设备上下文中。如果您使用它进行打印，您可能需要安排页眉和页脚; Scintilla 不会为您做这件事。见 **SciTEWin::Print()** 中 **SciTEWinDlg.cxx** 的一个例子。每次使用此消息都会将一系列文本呈现为矩形区域，并返回要打印的下一个字符的文档中的位置。

draw 控制是否有任何输出。如果要进行分页，请将此项设置为 **false**（例如，如果将其与 **MFC** 一起使用，则需要在 **OnBeginPrinting()** 输出每个页面之前对其进行分页。

```
struct Sci_Rectangle {int left; int top; int right; int bottom; };

struct Sci_RangeToFormat {
    Sci_SurfaceID hdc; //我们打印到的 Surface ID
    Sci_SurfaceID hdcTarget; //我们用于测量的 Surface ID (可能与 hdc 相同)
    Sci_Rectangle rc; //要打印的矩形
    Sci_Rectangle rcPage; //物理可打印的页面大小
    Sci_CharacterRange chrg; //要打印的字符范围
};
```

在 Windows 上，**hdc** 并且 **hdcTarget** 都应该设置为输出设备（通常为打印机）的设备上下文句柄。如果您打印到元文件，这些将与 Windows 元文件（与扩展元

文件不同) 不同, 不会实现用于返回信息的完整 API。在这种情况下, 设置 `hdcTarget` 为屏幕 DC。
`rcPage` 是矩形{0, 0, `maxX`, `maxY`} , 其中 `maxX+1` 和 `maxY+1` 是 x 和 y 中可物理打印的像素数。
`rc` 是用于渲染文本的矩形 (当然, 它将适合由 `rcPage` 定义的矩形)。
`chrg.cpMin` 并 `chrg.cpMax` 定义要输出的字符的起始位置和最大位置。绘制该字符范围内的所有每一行。

在 Cocoa 上, 当调用视图的 `drawRect` 方法时, `printing (draw=1)` 的表面 ID 应该是当前 context ((`CGContextRef`) `[[NSGraphicsContext currentContext] graphicsPort]`) 的图形端口。Surface ID 实际上并不用于 `measurement (draw=0)`, 但可以设置为位图上下文 (使用 `CGBitmapContextCreate`) 创建, 以避免运行时警告。

在 GTK +上, 可以从打印上下文找到要使用的曲面 ID
`gtk_print_context_get_cairo_context(context)`。

`chrg.cpMin` 并 `chrg.cpMax` 定义要输出的字符的起始位置和最大位置。绘制该字符范围内的所有每一行。

打印时, 最繁琐的部分总是要确定边距应该是什么, 以允许纸张的不可打印区域和打印页眉和页脚。如果你看一下 ScITE 中的打印代码, 你会发现大部分内容都是用它来处理的。如果您去掉所有边距, 不可打印区域, 页眉和页脚代码, 导致 Scintilla 呈现文本的循环非常简单。

SCI_SETPRINTMAGNIFICATION (int 放大)
SCI_GETPRINTMAGNIFICATION→int

`SCI_GETPRINTMAGNIFICATION` 允许您以与屏幕字体不同的大小进行打印。
magnification 是要添加到每个屏幕字体大小的点数。值-3 或-4 给出相当小的打印。您可以使用此值 `SCI_GETPRINTMAGNIFICATION`。

SCI_SETPRINTCOLOURMODE (int 模式)
SCI_GETPRINTCOLOURMODE→int

这两条消息设置并获取用于在可能使用白皮书的打印机上呈现彩色文本的方法。如果您使用黑色或黑色屏幕背景, 考虑颜色处理尤为重要。在黑色上打印白色比使用碳粉和墨水快很多倍。您可以将模式设置为以下之一:

符号	值	目的
SC_PRINT_NORMAL	0	使用当前屏幕颜色打印, 但在白色背景上打印的行号边距除外。这是默认值。
SC_PRINT_INVERTLIGHT	1	如果使用深色背景, 则可以通过反转所有颜色的光值并在白色背景上打印来节省墨水。
SC_PRINT_BLACKONWHITE	2	在白色背景上打印所有文本为黑色。

SC_PRINT_COLOURONWHITE	3	一切都在白色背景上以自己的颜色打印。
SC_PRINT_COLOURONWHITEDEFAULTBG	4	所有内容都以自己的前景色打印，但所有样式（包括 STYLE_LINENUMBER ）都将打印在白色背景上。
SC_PRINT_SCREENCOLOURS	五	使用当前屏幕颜色打印前景和背景。这是唯一不将行号边距的背景颜色设置为白色的模式。

SCI_SETPRINTWRAPMODE (int wrapMode)

SCI_GETPRINTWRAPMODE→int

这两个函数获取并设置打印机包装模式。*wrapMode* 可以设置为 **SC_WRAP_NONE** (0)，**SC_WRAP_WORD** (1) 或 **SC_WRAP_CHAR** (2)。默认值是 **SC_WRAP_WORD**，它包装打印输出，以便所有字符都适合打印矩形。如果设置 **SC_WRAP_NONE**，则每行文本生成一行输出，如果行太长而无法放入打印区域，则会截断该行。

SC_WRAP_WORD 尝试仅在白色空格或样式更改所指示的单词之间进行换行，但如果单词比行长，则它将在行结束之前换行。对于在单词之间没有空格的亚洲语言，**SC_WRAP_CHAR** 首选 **SC_WRAP_WORD**。

直接访问

SCI_GETDIRECTFUNCTION → int

SCI_GETDIRECTPOINTER → int

SCI_GETCHARACTERPOINTER → int

SCI_GETRANGEPOINTER(int start, int lengthRange) → int

SCI_GETGAPPOSITION → position

在 Windows 上，用于在容器和 Scintilla 之间进行通信的消息传递方案由操作系统 **SendMessage** 函数调解，并且在密集调用时可能导致性能不佳。为了避免这种开销，Scintilla 提供了允许您直接调用 Scintilla 消息函数的消息。在 C / C++ 中执行此操作的代码具有以下形式：

```
#include "Scintilla.h"
SciFnDirect pSciMsg = (SciFnDirect) SendMessage (hSciWnd,
SCI_GETDIRECTFUNCTION, 0, 0);
sptr_t pSciWndData = (sptr_t) SendMessage (hSciWnd,
SCI_GETDIRECTPOINTER, 0, 0);

//现在是一个直接调用 Scintilla 的包装器
sptr_t CallScintilla (unsigned int iMessage, uptr_t wParam, sptr_t
lParam) {
    return pSciMsg (pSciWndData, iMessage, wParam, lParam);
}
```

SciFnDirect，**sptr_t** 并 **uptr_t** 在宣布 **Scintilla.h**。 **hSciWnd** 是创建 Scintilla 窗口时返回的窗口句柄。

虽然速度更快，但是如果从不同的线程执行到 Scintilla 窗口的本机线程，这种直接调用将导致问题，在这种情况下 `SendMessage(hSciWnd, SCI_*, wParam, lParam)` 应该使用与窗口线程同步的情况。

此功能也适用于 GTK +，但对速度没有显着影响。

从 Windows 上的 1.47 版本开始，Scintilla 导出一个名为的函数 `Scintilla_DirectFunction`，该函数 可以与返回的函数使用相同 `SCI_GETDIRECTFUNCTION`。这可以节省您 `SCI_GETDIRECTFUNCTION` 通过函数指针间接调用 Scintilla 的调用 和需要。

SCI_GETDIRECTFUNCTION→int

此消息返回要调用以处理 Scintilla 消息的函数的地址，而不会通过 Windows 消息传递系统。无论您创建的 Scintilla 窗口的数量是多少，您只需要调用一次。

SCI_GETDIRECTPOINTER→int

这将返回一个指向数据的指针，该数据标识正在使用的 Scintilla 窗口。您必须为您创建的每个 Scintilla 窗口调用一次。调用直接函数时，必须传入与目标窗口关联的直接指针。

SCI_GETCHARACTERPOINTER→int

SCI_GETRANGEPOINTER (int start, int lengthRange) →int

SCI_GETGAPPOSITION→position

授予对 Scintilla 用于存储文档的内存的临时直接只读访问权限。

`SCI_GETCHARACTERPOINTER` 移动 Scintilla 中的间隙，以便连续存储文档的文本并确保文本后面有一个 NUL 字符，然后返回指向第一个字符的指针。然后，应用程序可以将其传递给接受字符指针的函数，例如正则表达式搜索或解析器。不应该写入指针，因为这可能使 Scintilla 的内部状态失去同步。

由于 Scintilla 中的任何操作都可能更改其内部状态，因此在任何调用之后或通过允许用户界面活动，此指针将变为无效。在对 Scintilla 进行任何调用或执行任何用户界面调用（如修改进度指示器）之后，应用程序应重新获取指针。

此调用在文档末尾插入字符需要相似的时间，这可能包括移动文档内容。具体而言，文档间隙之后的所有字符都移动到间隙之前。此压缩状态应该保持不会更改文档内容的调用和用户界面操作，因此之后重新获取指针非常快。如果此调用用于实现全局替换操作，则每次替换将移动间隙，因此如果

`SCI_GETCHARACTERPOINTER` 在每次替换后调用，则操作将变为 $O(n^2)$ 而不是 $O(n)$ 。相反，应找到并记住所有匹配，然后执行所有替换。

`SCI_GETRANGEPOINTER` 提供对所要求范围的直接访问。除非它在请求的范围内，否则不会移动间隙，因此此调用可以比快 `SCI_GETCHARACTERPOINTER`。这可以由能够作用于文本块或行范围的应用程序代码使用。

SCI_GETGAPPOSITION 返回当前的间隙位置。这是一个提示，应用程序可以使用以避免使用 **SCI_GETRANGEPOINTER** 包含差距的范围调用以及随之而来的移动差距的成本。

多个视图

Scintilla 窗口及其显示的文档是单独的实体。创建新窗口时，还会创建一个新的空文档。每个文档的引用计数最初设置为 1。该文档还有一个链接到它的 Scintilla 窗口列表，因此当任何窗口更改文档时，通知它的所有其他窗口都会通知它们使它们更新。系统以这种方式排列，以便您可以在单个 Scintilla 窗口中处理许多文档，因此您可以在多个窗口中显示单个文档（用于拆分窗口）。

虽然这些消息使用 **document *doc**，为了确保与未来 Scintilla 版本的兼容性，您应该将其 **doc** 视为不透明 **void***。也就是说，您可以按照本节中的说明使用和存储指针，但不应取消引用它。

```
SCI_GETDOCPOINTER → document *
SCI_SETDOCPOINTER(<unused>, document *doc)
SCI_CREATEDOCUMENT(int bytes, int documentOptions) → document *
SCI_ADDREFDOCUMENT(<unused>, document *doc)
SCI_RELEASEDOCUMENT(<unused>, document *doc)
SCI_GETDOCUMENTOPTIONS → int
```

SCI_GETDOCPOINTER → **document ***

这将返回指向窗口当前正在使用的文档的指针。它没有其他影响。

SCI_SETDOCPOINTER (**<unused>**, **document * doc**)

此消息执行以下操作：

1. 它从当前文档持有的列表中删除当前窗口。
2. 它将当前文档的引用计数减少 1。
3. 如果引用计数达到 0，则删除该文档。
4. **doc** 被设置为窗口的新文档。
5. 如果 **doc** 为 0，则创建一个新的空文档并将其附加到窗口。
6. 如果 **doc** 不为 0，则其引用计数增加 1。

SCI_CREATEDOCUMENT (**int bytes**, **int documentOptions**) → **document ***

此消息创建一个新的空文档并返回指向它的指针。此文档未在编辑器中选择，并以引用计数 1 开始。这意味着您拥有它，并且必须在使用后将其引用计数减少 1，**SCI_SETDOCPOINTER** 以便 Scintilla 窗口拥有它，或者您必须确保

SCI_RELEASEDOCUMENT 在关闭应用程序之前将引用计数减少 1 以避免内存泄漏。该 **bytes** 参数确定文档的初始内存分配，因为分配一次更有效，而不是在添加数据时依赖缓冲区增长。如果 **SCI_CREATEDOCUMENT** 失败则返回 0。

的 **documentOptions** 影响存储器分配和性能不同的文档能力之间的参数选择

SC_DOCUMENTOPTION_DEFAULT (0) 选择标准选项。 **SC_DOCUMENTOPTION_STYLES_NONE** (0x1) 停止为样式字符分配内存，这样可以节省大量内存，通常 40%，整个

文档被视为样式 0. Lexers 仍然可以使用指标生成视觉样式。

SC_DOCUMENTOPTION_TEXT_LARGE (0x100) 可容纳 64 位可执行文件中大于 2 GigaBytes 的文档。

有了 **SC_DOCUMENTOPTION_STYLES_NONE**，词法分析器仍处于活动状态，可能会显示指标。有些人可能会产生折叠信息，但大多数人都需要词汇风格来正确地确定折叠。设置 **null** 词法分析器通常更有效，**SCLEX_NULL** 因此不运行词法分析器。

该 **SC_DOCUMENTOPTION_TEXT_LARGE** 选项是实验性的，尚未经过全面测试。当 **lexing** 超过 2GB 或 4GB 时，Lexers 可能会失败或挂起。使用此选项的应用程序应进行测试，以确保该选项适用于其环境，并且包含的每个词法分析器也应使用大于 4GB 的文档进行测试。对于许多应用程序来说，大于 4GB 的 **lexing** 文件太迟钝了，因此可以使用 **SC_DOCUMENTOPTION_STYLES_NONE null lexer SCLEX_NULL**。另一种方法是打开空闲样式 **SCI_SETIDLESTYLING**。

符号	值	影响
SC_DOCUMENTOPTION_DEFAULT	0	标准行为
SC_DOCUMENTOPTION_STYLES_NONE	为 0x1	停止为样式分配内存并将所有文本视为样式 0。
SC_DOCUMENTOPTION_TEXT_LARGE	为 0x100	允许文档大于 2 GB。

SCI_ADDREFDOCUMENT (<unused>, document * doc)

这会将文档的引用计数增加 1。如果要在 Scintilla 窗口中替换当前文档并获取当前文档的所有权，例如，如果要编辑许多文档在一个窗口中，执行以下操作：

1. **SCI_GETDOCPOINTER** 用于获取指向文档的指针 *doc*。
2. **SCI_ADDREFDOCUMENT(0, doc)** 用于增加引用计数。
3. **SCI_SETDOCPOINTER(0, docNew)** 用于设置其他文档或 **SCI_SETDOCPOINTER(0, 0)** 设置新的空文档。

SCI_RELEASEDOCUMENT (<unused>, document * doc)

此消息减少了标识的文档的引用计数 *doc*。doc 必须是 **SCI_GETDOCPOINTER** 或者 **SCI_CREATEDOCUMENT** 必须指向仍然存在的文档的结果。如果您在引用计数为 1 且仍附加到 Scintilla 窗口的文档上调用此方法，则会发生错误。为了让世界在其轨道中旋转，您必须平衡每次呼叫 **SCI_CREATEDOCUMENT** 或 **SCI_ADDREFDOCUMENT** 呼叫 **SCI_RELEASEDOCUMENT**。

SCI_GETDOCUMENTOPTIONS→int

返回用于创建文档的选项。

背景加载和保存

为了确保响应式用户界面，应用程序可以决定使用来自用户界面的单独线程来加载和保存文档。

在后台加载

SCI_CREATELOADER(int bytes, int documentOptions) → int

应用程序可以将所有文件加载到它在后台线程上分配的缓冲区中，然后将该缓冲区中的数据添加到用户界面线程上的 Scintilla 文档中。该技术使用额外的内存来存储文件的完整副本，这也意味着 Scintilla 执行初始行结束发现所花费的时间会阻塞用户界面。

为了避免这些问题，可以创建加载器对象并用于加载文件。loader 对象支持 ILoader 接口。

SCI_CREATELOADER (int bytes, int documentOptions) →int

创建一个支持 ILoader 接口的对象，该接口可用于加载数据，然后转换为 Scintilla 文档对象以附加到视图对象。该 *bytes* 参数确定文档的初始内存分配，因为分配一次更有效，而不是在添加数据时依赖缓冲区增长。如果 SCI_CREATELOADER 失败则返回 0。

该 *documentOptions* 论点在本 SCI_CREATEDOCUMENT 节中描述。

ILoader

```
class ILoader { public : virtual int SCI_METHOD Release () = 0 ; //从 SC_STATUS_* virtual int 返回状态代码 SCI_METHOD AddData (const char * data , Sci_Position length ) = 0 ; virtual void * SCI_METHOD ConvertToDocument () = 0 ; };
```

应用程序应该 AddData 使用从文件中读取的每个数据块调用该方法。AddData 将返回 SC_STATUS_OK，除非发生内存耗尽等故障。如果 AddData 在文件读取调用中或在文件读取调用中发生故障，则可以放弃加载并且随着 Release 调用释放加载器。当整个文件被读取时，ConvertToDocument 应调用生成 Scintilla 文档指针。新创建的文档的引用计数为 1，与从 SCI_CREATEDOCUMENT 返回的文档指针的方式相同。有没有需要调用 Release 后 ConvertToDocument。

在后台保存

想要在后台保存的应用程序应锁定文档 SCI_SETREADONLY(1) 以防止修改并使用 SCI_GETCHARACTERPOINTER 检索指向统一文档内容的指针。锁定文档的缓冲区不会移动，因此指针在应用程序调用之前有效 SCI_SETREADONLY(0)。

如果用户在文档被锁定时尝试执行修改，`SCI_MODIFYATTEMPTRO` 则会向应用程序发送通知。然后，应用程序可以决定忽略修改或终止后台保存线程并在从通知返回之前重新启用修改。

折页

折叠的基本操作是使线条不可见或可见。线条可见性是视图的属性而不是文档，因此每个视图可能显示不同的一组线条。从用户的角度来看，使用折叠点隐藏和显示线条。通常，文档的折叠点基于文档内容的层次结构。在 Python 中，层次结构由缩进确定，在 C++ 中由括号字符确定。通过将数字“折叠级别”附加到每一行，可以在 Scintilla 文档对象中表示此层次结构。词法分析器最容易设置折叠级别，但您也可以使用消息进行设置。

您的代码可以设置用户操作与折叠和展开之间的连接。查看如何完成此操作的最佳方法是在 ScITE 源代码中搜索本文档此部分中使用的消息，并查看它们的使用方法。您还需要使用标记和折叠边距来完成折叠实施。该“`fold`”属性应设置为“1”with `SCI_SETPROPERTY("fold", "1")`以启用折叠。

```
SCI_VISIBLEFROMDOCLINE(int docLine) → int
SCI_DOCLINEFROMVISIBLE(int displayLine) → int
SCI_SHOWLINES(int lineStart, int lineEnd)
SCI_HIDELINES(int lineStart, int lineEnd)
SCI_GETLINEVISIBLE(int line) → bool
SCI_GETALLLINESVISIBLE → bool
SCI_SETFOLDLEVEL(int line, int level)
SCI_GETFOLDLEVEL(int line) → int
SCI_SETAUTOMATICFOLD(int automaticFold)
SCI_GETAUTOMATICFOLD → int
SCI_SETFOLDFLAGS(int flags)
SCI_GETLASTCHILD(int line, int level) → int
SCI_GETFOLDPARENT(int line) → int
SCI_SETFOLDEXPANDED(int line, bool expanded)
SCI_GETFOLDEXPANDED(int line) → bool
SCI_CONTRACTEDFOLDNEXT(int lineStart) → int
SCI_TOGGLEFOLD(int line)
SCI_TOGGLEFOLDSHOWTEXT(int line, const char *text)
SCI_FOLDDISPLAYTEXTSETSTYLE(int style)
SCI_FOLDLINE(int line, int action)
SCI_FOLDCHILDREN(int line, int action)
SCI_FOLDALL(int action)
SCI_EXPANDCHILDREN(int line, int level)
SCI_ENSUREVISIBLE(int line)
SCI_ENSUREVISIBLEENFORCEPOLICY(int line)
```

SCI_VISIBLEFROMDOCLINE (int docLine) →int

当隐藏某些行和/或显示注释时，文档中的特定行可能会显示在与其文档位置不同的位置。如果没有隐藏任何行且没有注释，则返回此消息 `docLine`。否则，返回显示行（将第一个可见行计为 0）。不可见线的显示行与前一个可见线相同。文档中第一行的显示行号为 0。如果行被隐藏且 `docLine` 超出文档中的行范围，则返回值为 -1。如果换行，则可以占用多条显示行。

SCI_DOCLINEFROMVISIBLE (int displayLine) →int

当隐藏某些行和/或显示注释时，文档中的特定行可能会显示在与其文档位置不同的位置。此消息返回与显示行对应的文档行号（将文档中第一行的显示行计为 0）。如果 *displayLine* 小于或等于 0，则结果为 0。如果 *displayLine* 大于或等于显示的行数，则结果为文档中的行数。

SCI_SHOWLINES (int lineStart, int lineEnd)

SCI_HIDE_LINES (int lineStart, int lineEnd)

SCI_GETLINEVISIBLE (int line) →bool

SCI_GETALLLINESVISIBLE →bool

前两条消息将一系列行标记为可见或不可见，然后重绘显示。

SCI_GETLINEVISIBLE 报告一行的可见状态，如果可见则返回 1，如果不可见则返回 0。**SCI_GETALLLINESVISIBLE** 如果所有行都可见则返回 1，如果隐藏某些行则返回 0。这些消息对折叠级别或折叠标记没有影响。第一行无法隐藏。

SCI_SETFOLDLEVEL (int line, int level)

SCI_GETFOLDLEVEL (int line) →int

这两个消息设置并获得一个 32 位值，其中包含一行的折叠级别和一些与折叠相关的标志。折叠级别是 0 到 **SC_FOLDLEVELNUMBERMASK** (0x0FFF) 范围内的数字。但是，初始折叠级别设置为 **SC_FOLDLEVELBASE** (0x400) 以允许折叠级别的无符号算术。有两个加法标志位。**SC_FOLDLEVELWHITEFLAG** 表示该行为空白，并允许对其进行略微不同的处理，然后其级别可能表示。例如，空白行通常不应该是折叠点，并且即使它们可能具有较小的折叠水平，也将被视为前一部分的一部分。**SC_FOLDLEVELHEADERFLAG** 表示该行是标题（折叠点）。

使用 **SCI_GETFOLDLEVEL(line) & SC_FOLDLEVELNUMBERMASK** 取得一行的折叠级别。同样，用于 **SCI_GETFOLDLEVEL(line) & SC_FOLDLEVEL*FLAG** 获取标志的状态。要设置必须或在相关标志中的折叠级别。例如，要将级别设置为 **thisLevel** 并将一条线标记为折叠点，请使用：**SCI_SETFOLDLEVEL(line, thisLevel | SC_FOLDLEVELHEADERFLAG)**。

如果您使用词法分析器，则不需要使用，**SCI_SETFOLDLEVEL** 因为词法分析器可以更好地处理它。您需要使用它 **SCI_GETFOLDLEVEL** 来决定如何处理用户折叠请求。如果您确实更改了折叠级别，折叠边距将更新以匹配您的更改。

SCI_SETFOLD_FLAGS (int flags)

除了在折叠边距中显示标记外，您还可以通过在文本区域中绘制线条来向用户指示折叠。线条以前景色设置绘制 **STYLE_DEFAULT**。设置的位 *flags* 确定绘制折叠线的位置：

符号	值	影响
	1	已删除的实验功能。
SC_FOLDFLAG_LINEBEFORE_EXPANDED	2	如果展开则绘制在上面
SC_FOLDFLAG_LINEBEFORE_CONTRACTED	4	如果不扩展，请在上面绘制

SC_FOLDFLAG_LINEAFTER_EXPANDED	8	如果展开则绘制如下
SC_FOLDFLAG_LINEAFTER_CONTRACTED	16	如果不扩展，请在下面绘制在行边距中显示十六进制折叠级别以帮助调整折叠。此功能的外观可能在将来发生变化。
SC_FOLDFLAG_LEVELNUMBERS	64	在行边缘显示十六进制行状态，以帮助调整 <code>lexing</code> 和折叠。不得与之同时使用 <code>SC_FOLDFLAG_LEVELNUMBERS</code> 。
SC_FOLDFLAG_LINESTATE	128	

此消息导致显示重绘。

SCI_GETLASTCHILD (int line, int level) →int

此消息搜索之后的下一行 *line*，其折叠级别小于或等于 *level*，然后返回上一行号。如果设置 *level* 为-1，*level* 则设置为折叠级别的行 *line*。如果 *from* 是折叠点，`SCI_GETLASTCHILD(from, -1)`则通过切换折叠状态返回可见或隐藏的最后一行。

SCI_GETFOLDPARENT (int line) →int

此消息返回第一行的行号，然后 *line* 将其标记为折叠点，`SC_FOLDLEVELHEADERFLAG` 并且折叠级别小于 *line*。如果未找到任何行，或者标题标志和折叠级别不一致，则返回值为-1。

SCI_TOGGLEFOLD (int line)

SCI_TOGGLEFOLDSHOWTEXT (int line, const char * text)

每个折叠点可以展开，显示所有子行，也可以缩小，隐藏所有子行。只要具有该 `SC_FOLDLEVELHEADERFLAG` 集合，这些消息就切换给定行的折叠状态。这些消息负责折叠或扩展依赖于该行的所有行。此消息后显示更新。

可以使用 *text* 参数 `to` 在折叠文本的右侧显示可选的文本标记 `SCI_TOGGLEFOLDSHOWTEXT`。文本是用 `STYLE_FOLDDISPLAYTEXT` 样式绘制的 。

SCI_FOLDDISPLAYTEXTSETSTYLE (int style)

此消息更改折叠文本标记的外观。

符号	值	影响
SC_FOLDDISPLAYTEXT_HIDDEN	0	不显示文本标签。这是默认值。
SC_FOLDDISPLAYTEXT_STANDARD	1	显示文本标签。
SC_FOLDDISPLAYTEXT_BOXED	2	显示文本标签，并在其周围绘制一个框。

SCI_SETFOLDEXPANDED (int line, bool expanded)

SCI_GETFOLDEXPANDED (int line) →bool

这些消息设置并获得单行的展开状态。set 消息对行的可见状态或依赖于它的任何行没有影响。它确实改变了折叠边距中的标记。如果要求文档外部的行的扩展状态，则结果为 **false** (0)。

如果您只想切换一行的折叠状态并处理依赖于它的所有行，则使用起来要容易得多 **SCI_TOGGLEFOLD**。 **SCI_SETFOLDEXPANDED** 在完成之前，您将使用该 消息处理多个折叠而不更新显示。请参阅 **SciTEBase::FoldAll()** 以及 **SciTEBase::Expand()** 使用这些消息的示例。

SCI_FOLDLINE (int line, int action)

SCI_FOLDCHILDREN (int line, int action)

SCI_FOLDALL (int action)

这些消息提供了更高级别的折叠方法，而不是设置扩展标志并显示或隐藏单独的行。

可以收缩/扩展/切换单个折叠 **SCI_FOLDLINE**。要影响所有儿童褶皱以及打电话 **SCI_FOLDCHILDREN**。

影响整个文档调用 **SCI_FOLDALL**。与 **SC_FOLDACTION_TOGGLE** 文档中的第一折叠头被检查以决定是否扩大或收缩。

符号	值	影响
SC_FOLDACTION_CONTRACT	0	合同。
SC_FOLDACTION_EXPAND	1	扩大。
SC_FOLDACTION_TOGGLE	2	在签约和扩展之间切换。

SCI_EXPANDCHILDREN (int line, int level)

这用于响应对导致其折叠级别的行的更改，或者是否是要更改的标题，可能是在添加或删除“{”时。

当容器收到线路已更改的通知时，折叠级别已经设置，因此容器必须使用此调用中的上一级别，以便可以显示隐藏在此行下方的任何范围。

SCI_SETAUTOMATICFOLD (int automaticFold)

SCI_GETAUTOMATICFOLD →int

Scintilla 可以提供足以满足许多应用程序的行为，而不是实现处理容器中折叠的所有逻辑。该 *automaticFold* 参数是一个位集，用于定义应该启用 3 个折叠实现中的哪一个。大多数应用程序应该能够使用 **SC_AUTOMATICFOLD_SHOW** 和 **SC_AUTOMATICFOLD_CHANGE** 标志，除非他们希望实现完全不同的行为，例如定义自己的折叠结构。 **SC_AUTOMATICFOLD_CLICK** 当应用程序想要添加或更改单击行为时更有可能被设置，例如仅当 **Shift + Alt** 与单击结合使用时才显示方法标题。

符号	值	影响
SC_AUTOMATICFOLD_SHOW	1	根据需要自动显示行。这可以避免发送 SCN_NEEDSHOWN 通知。
SC_AUTOMATICFOLD_CLICK	2	自动处理折叠边距中的点击次数。这样可以避免发送 SCN_MARGINCLICK 折叠边距的通知。
SC_AUTOMATICFOLD_CHANGE	4	折叠结构更改时根据需要显示线条。该 SCN_MODIFIED 通知还发送，除非它是由容器禁用。

SCI_CONTRACTEDFOLDNEXT (int lineStart) →int

有效搜索缩小折叠标题的行。在切换文档或使用文件保存折叠时保存用户的折叠时，这非常有用。搜索从行号开始 *lineStart* 并继续向前到文件末尾。

lineStart 如果它是一个缩小的折叠标题，则返回，否则返回下一个缩小的折叠标题。如果没有更多的合同折叠标题，则返回-1。

SCI_ENSUREVISIBLE (int line)

SCI_ENSUREVISIBLEENFORCEPOLICY (int line)

可能隐藏一条线，因为其父线的多个收缩。这两条消息都向上传递到折叠层次结构，扩展任何缩小的折叠，直到达到顶层。然后该线将可见。如果使用

SCI_ENSUREVISIBLEENFORCEPOLICY，**SCI_SETVISIBLEPOLICY** 则应用设置的垂直插入符号策略。

换行

```

SCI_SETWRAPMODE(int wrapMode)
SCI_GETWRAPMODE → int
SCI_SETWRAPVISUALFLAGS(int wrapVisualFlags)
SCI_GETWRAPVISUALFLAGS → int
SCI_SETWRAPVISUALFLAGSLLOCATION(int wrapVisualFlagsLocation)
SCI_GETWRAPVISUALFLAGSLLOCATION → int
SCI_SETWRAPINDENTMODE(int wrapIndentMode)
SCI_GETWRAPINDENTMODE → int
SCI_SETWRAPSTARTINDENT(int indent)
SCI_GETWRAPSTARTINDENT → int
SCI_SETLAYOUTCACHE(int cacheMode)
SCI_GETLAYOUTCACHE → int
SCI_SETPOSITIONCACHE(int size)
SCI_GETPOSITIONCACHE → int
SCI_LINESSPLIT(int pixelWidth)
SCI_LINESJOIN
SCI_WRAPCOUNT(int docLine) → int

```

默认情况下，Scintilla 不会包装文本行。如果启用换行，则在以下行中继续比窗口宽度宽的行。在空格或制表符后或不同样式的行之间断行。如果这是不可能的，因为一种风格的单词比窗口宽，那么在完全适合该行的最后一个字符之后发生中断。打开包装模式时，不会出现水平滚动条。

对于包裹线，Scintilla 可以在包裹线的子线的末端和下一个子线的开始处绘制视觉标志（小箭头）。这些可以单独启用，但如果 Scintilla 在下一个子行的开头绘制可视标志，则此子行将缩进一个 char。独立于开始时绘制视觉标志，子线可以有缩进。

Scintilla 使用的大部分时间都用于布局和绘制文本。即使在这些计算中使用的数据没有改变时，也可以多次执行相同的文本布局计算。为了避免在某些情况下进行这些不必要的计算，行布局缓存可以存储计算结果。只要基础数据（例如文档的内容或样式）发生变化，缓存就会失效。缓存整个文档的布局效果最好，使动态换行速度提高了 20 倍，但这需要文档内容所需内存的 7 倍加上每行大约 80 个字节。

如果有更改，则不会立即执行换行，但会延迟换行，直到重绘显示为止。此延迟通过允许执行一组更改然后包装并显示一次来提高性能。因此，某些操作可能不会按预期发生。如果读取文件并且滚动位置移动到文本中的特定行，例如当容器尝试恢复先前的编辑会话时发生，则滚动位置将在包装之前确定，因此意外的文本范围将是显示。要正确滚动到该位置，请通过等待初始 **SCN_PAINTED** 通知来延迟滚动，直到执行换行。

SCI_SETWRAPMODE (int wrapMode)
SCI_GETWRAPMODE→int

将 wrapMode 设置为 **SC_WRAP_WORD** (1) 以启用包装在单词或样式边界上，**SC_WRAP_CHAR** (2) 启用任何字符之间的换行，**SC_WRAP_WHITESPACE** (3) 启用换行空白，以及 **SC_WRAP_NONE** (0) 禁用换行。**SC_WRAP_CHAR** 对于单词之间没有空格的亚洲语言是首选。

SCI_SETWRAPVISUALFLAGS (int wrapVisualFlags)
SCI_GETWRAPVISUALFLAGS→int

您可以启用可视标志的绘制以指示包装线。wrapVisualFlags 中设置的位确定绘制哪些可视标志。

符号	值	影响
SC_WRAPVISUALFLAG_NONE	0	没有视觉旗帜
SC_WRAPVISUALFLAG_END	1	视觉旗帜在包裹线的子线的末尾。
SC_WRAPVISUALFLAG_START	2	视觉旗帜在包装线的子线开始。 子线缩进至少 1，为旗帜腾出空间。
SC_WRAPVISUALFLAG_MARGIN	4	行号边距中的可视标志。

SCI_SETWRAPVISUALFLAGSLOCATION (int wrapVisualFlagsLocation)
SCI_GETWRAPVISUALFLAGSLOCATION→int

您可以设置是否在边框附近或文本附近绘制表示线条的可视标记。在 wrapVisualFlagsLocation 中设置的位将位置设置为相应可视标志的文本附近。

符号	值	影响
----	---	----

<code>SC_WRAPVISUALFLAGLOC_DEFAULT</code>	0	边界附近绘制的视觉标志
<code>SC_WRAPVISUALFLAGLOC_END_BY_TEXT</code>	1	在文本附近画的子线的末端的视觉旗子
<code>SC_WRAPVISUALFLAGLOC_START_BY_TEXT</code>	2	在文本附近画的子线的开头的视觉旗子

SCI_SETWRAPINDENTMODE (int wrapIndentMode)

SCI_GETWRAPINDENTMODE→int

包装的子行可以缩进到第一个子行的位置或一个缩进级别。默认是

`SC_WRAPINDENT_FIXED`。模式是：

符号	值	影响
<code>SC_WRAPINDENT_FIXED</code>	0	与窗口左侧对齐的包装子线加上 <code>SCI_SETWRAPSTARTINDENT</code> 设置的 金额
<code>SC_WRAPINDENT_SAME</code>	1	包装的子行与第一个子行缩进对齐
<code>SC_WRAPINDENT_INDENT</code>	2	包装的子行与第一个子行缩进对齐，再加上一个缩进级别
<code>SC_WRAPINDENT_DEEPINDENT</code>	3	包装的子行与第一个子行缩进对齐，再加上两个缩进级别

SCI_SETWRAPSTARTINDENT (int indent)

SCI_GETWRAPSTARTINDENT→int

`SCI_SETWRAPSTARTINDENT` 根据平均字符宽度设置包装行的子行缩进的大小 `STYLE_DEFAULT`。缩进大小没有限制，但小于 0 或大值的值可能会产生不良影响。子线的缩进与视觉标志无关，但如果 `SC_WRAPVISUALFLAG_START` 设置，则使用至少为 1 的缩进。

SCI_SETLAYOUTCACHE (int cacheMode)

SCI_GETLAYOUTCACHE→int

您可以设置 *cacheMode* 为表中的一个符号：

符号	值	为这些行缓存的布局
<code>SC_CACHE_NONE</code>	0	没有缓存行。
<code>SC_CACHE_CARET</code>	1	包含文本插入符的行。这是默认值。
<code>SC_CACHE_PAGE</code>	2	可见线条加上包含插入符号的线条。
<code>SC_CACHE_DOCUMENT</code>	3	文档中的所有行。

SCI_SETPOSITIONCACHE (int size)

SCI_GETPOSITIONCACHE→int

位置缓存存储短文本运行的位置信息，以便在运行重复时更快地确定其布局。

可以使用设置此缓存的条目大小 `SCI_SETPOSITIONCACHE`。

SCI_LINESSPLIT (int pixelWidth)

将目标指示的行范围拆分为最多像素宽度的行。尽可能以与换行相似的方式在字边界上进行拆分。当 *pixelWidth* 为 0 时，则使用窗口的宽度。

SCI_LINESJOIN

通过删除行结束字符，将目标指示的行范围加入一行。如果这会导致单词之间没有空格，则会插入额外的空格。

SCI_WRAPCOUNT (int docLine) → int 如果

文档行换行，则它们可以占用多个显示行，这将返回包装文档行所需的显示行数。

缩放

Scintilla 采用了“缩放系数”，可让您以一个点的步长使文档中的所有文本变大或变小。无论您设置的缩放系数如何，显示的磅值都不会低于 2。您可以在 -10 到 +20 点范围内设置缩放系数。

```
SCI_ZOOMIN
SCI_ZOOMOUT
SCI_SETZOOM(int zoomInPoints)
SCI_GETZOOM → int
```

如果当前缩放系数小于 20 点，**SCI_ZOOMIN** **SCI_ZOOMOUT** 会将

SCI_ZOOMIN 缩放系数增加一个点。**SCI_ZOOMOUT** 如果当前缩放系数大于 -10 点，则将缩放系数减小一个点。

SCI_SETZOOM (int zoomInPoints)

SCI_GETZOOM → int

这些消息允许您直接设置和获取缩放系数。您可以设置的因素没有限制设置，因此将自己限制在 -10 到 +20 以匹配增量缩放功能是一个好主意。

排长龙

您可以选择通过绘制垂直线或通过着色超出设置长度的字符背景来标记超过给定长度的线条。

```
SCI_SETEDGEMODE(int edgeMode)
SCI_GETEDGEMODE → int
SCI_SETEDGECOLUMN(int column)
SCI_GETEDGECOLUMN → int
SCI_SETEDGECOLOUR(colour edgeColour)
SCI_GETEDGECOLOUR → colour

SCI_MULTIEDGEADDLINE(int column, colour edgeColour)
SCI_MULTIEDGECLEARALL
```

SCI_SETEDGEMODE (int edgeMode)

SCI_GETEDGEMODE→int

这两条消息设置并获取用于显示长行的模式。您可以在表中设置其中一个值：

符号	值	长线显示模式
EDGE_NONE	0	长线没有标记。这是默认状态。 在由列号设置的垂直线上绘制 SCI_SETEDGECOLUMN 。这适用于等宽字体。该行是根据空格字符的宽度绘制的
EDGE_LINE	1	STYLE_DEFAULT ，因此如果您的样式使用比例字体或者您的样式具有不同的字体大小或者使用粗体，斜体和普通文本的混合，它可能无法正常工作。
EDGE_BACKGROUND	2	列限制后字符的背景颜色更改为由设置的颜色 SCI_SETEDGE COLOUR 。建议用于比例字体。 这类似于 EDGE_LINE 但是与仅显示一条单线相反，可以同时显示一组可配置的垂直线。这 edgeMode 使用完全独立的数据集，只能使用 SCI_MULTIEDGE* 消息进行配置。
EDGE_MULTILINE	3	

SCI_SETEDGECOLUMN (int column)

SCI_GETEDGECOLUMN→int

这些消息设置并获取显示长行标记的列号。绘制线条时，列以空格字符宽度为单位设置位置 **STYLE_DEFAULT**。设置背景颜色时，该列是字符数（允许选项卡）到行中。

SCI_SETEDGE COLOUR (color edgeColour)

SCI_GETEDGE COLOUR→color

这些消息设置并获取用于显示行超出设置长度的标记的颜色 **SCI_SETEDGECOLUMN**。

SCI_MULTIEDGEADDLINE (int column, color edgeColour)

SCI_MULTIEDGECLEARALL

SCI_MULTIEDGEADDLINE 为视图添加新的垂直边。边缘将显示在给定的列号处。生成的边缘位置取决于空间字符的度量标准 **STYLE_DEFAULT**。可以清除所有边缘 **SCI_MULTIEDGECLEARALL**。

无障碍

Scintilla 支持一些平台辅助功能。这种支持因平台而异。在 **GTK +**和 **Cocoa** 上，平台可访问性 **API**的实现足以使屏幕阅读器工作。在 **Win32** 上，系统插入符被操纵以帮助屏幕阅读器。

SCI_SETACCESSIBILITY(int accessibility)

SCI_GETACCESSIBILITY → int

SCI_SETACCESSIBILITY (int 辅助功能)

SCI_GETACCESSIBILITY→int

这些消息可以启用或禁用辅助功能并报告其当前状态。

在大多数平台上，可访问要么执行或不执行，这可以用被发现 **SCI_GETACCESSIBILITY** 有 **SCI_SETACCESSIBILITY** 不执行任何动作。在 GTK + 上，可访问性存在存储和性能成本，因此可以通过调用禁用它 **SCI_SETACCESSIBILITY**。

符号	值	辅助功能状态
SC_ACCESSIBILITY_DISABLED	0	可访问性已禁用。
SC_ACCESSIBILITY_ENABLED	1	辅助功能已启用。

词法分析器

如果 **SCI_LEXER** 在构建 Scintilla 时定义符号（有时称为 Scintilla 的 SciLexer 版本），则会包含对各种编程语言的支持，并支持本节中的消息。如果要为不支持的语言设置样式和折叠点，您可以在容器中执行此操作，或者更好的是，按照其中一个现有模式编写自己的词法分析器。

Scintilla 还支持外部词法分析器。这些 DLL 文件（在 Windows 上）或导出三个函数的 .so 模块（在 GTK + / Linux 的）：**GetLexerCount**，**GetLexerName** 和 **GetLexerFactory**。了解 **externalLexer.cxx** 更多。

```
SCI_SETLEXER (int lexer)  
SCI_GETLEXER→int  
SCI_SETLEXERLANGUAGE (<unused>, const char * language)  
SCI_GETLEXERLANGUAGE (<unused>, char * language)→int  
SCI_LOADLEXERLIBRARY (<unused>, const char * path)  
SCI_COLOURISE (int start, int end)  
SCI_CHANGELEXERSTATE (int start, int end)→int  
SCI_PROPERTYNAMES (<unused>, char * names)→int  
SCI_PROPERTYTYPE (const char * name)→int  
SCI_DESCRIBEPROPERTY (const char * name, char * description)  
→int  
SCI_SETPROPERTY (const char * key , const char * value)  
SCI_GETPROPERTY (const char * key, char * value)→int  
SCI_GETPROPERTYEXPANDED (const char * key, char * value)→int  
SCI_GETPROPERTYINT (const char * key, int defaultValue)→int  
SCI_DESCRIBEKEYWORDSETS (<unused>, char * description)→int  
SCI_SETKEYWORDS (int keyWordSet, const char * keyWords)  
SCI_GETSUBSTYLEBASES (<unused>, char * styles)→int  
SCI_DISTANCETOSECONDARYSTYLES→int  
SCI_ALLOCATESUBSTYLES (int styleBase, int numberStyles)→int  
SCI_FREESUBSTYLES  
SCI_GETSUBSTYLESSTART ( int styleBase)→int  
SCI_GETSUBSTYLESLENGTH (int styleBase)→int  
SCI_GETSTYLEFROMSUBSTYLE (int subStyle)→int  
SCI_GETPRIMARYSTYLEFROMSTYLE (int style)→int
```

SCI_SETIDENTIFIERS (int style, const char * identifiers)
SCI_PRIVATELEXERCALL (int operation, int pointer) →int
SCI_GETNAMEDSTYLES→int
SCI_NAMEOFSTYLE (int style, char * name) →int
SCI_TAGSOFSSTYLE (int style, char * tags) →int
SCI_DESCRIPTIONOFSTYLE (int style, char * description) →int

SCI_SETLEXER (int lexer)
SCI_GETLEXER→int

您可以选择要使用 **SCLEX_***枚举中的整数代码的词法分析器 **Scintilla.h**。此序列中有两个不使用词法分析器的代码：**SCLEX_NULL** 选择无 **lexing** 动作，**SCLEX_CONTAINER** 并在 **SCN_STYLENEEDED** 需要设置样式范围时将通知发送到容器。你不能使用这个 **SCLEX_AUTOMATIC** 值；这标识了 Scintilla 分配未使用的词法分析器编号的其他外部词法分析器。

SCI_SETLEXERLANGUAGE (<unused>, const char *语言)
SCI_GETLEXERLANGUAGE (<unused>, char *语言 NUL 终止) →int
SCI_SETLEXERLANGUAGE 允许您按名称选择词法分析器，并且是唯一的方法，如果您使用外部词法分析器或者如果您有为您自己的语言编写词法分析器模块，并且不希望为其分配明确的词法分析器编号。要选择现有词法分析器，请设置 **Language** 为匹配给予模块的（区分大小写）名称，例如“ada”或“python”，而不是“Ada”或“Python”。要找到内置词法分析器的名称，请打开相关 **Lex*.cxx** 文件并搜索 **LexerModule**。**LexerModule** 构造函数中的第三个参数 是要使用的名称。

要测试词法分配器是否有效，请 **SCI_GETLEXER** 在设置新词法分析器之前和之后使用，以查看词法分析器编号是否已更改。

SCI_GETLEXERLANGUAGE 检索词法分析器的名称。

SCI_LOADLEXERLIBRARY (<unused>, const char * path)
加载在共享库中实现的词法分析器。这是 GTK + / Linux 上的.so 文件或 Windows 上的.DLL 文件。

SCI_COLOURISE (int start, int end)
它请求当前词法分析器或容器（如果词法分析器设置为 **SCLEX_CONTAINER**）以在 **start** 和 之间设置 文档的样式 **end**。如果 **end** 为-1，则文档的样式从 **start** 最后开始。如果“**fold**”属性设置为 “1”并且词法分析器或容器支持折叠，则还会设置折叠级别。此消息导致重绘。

SCI_CHANGELEXERSTATE (int start, int end) →int
表示词法分析器的内部状态在一个范围内发生了变化，因此可能需要重绘。

SCI_PROPERTYNAMES (<unused>, char *名称 NUL 终止) →int
SCI_PROPERTYTYPE (const char * name) →int
SCI_DESCRIBEPROPERTY (const char * name, char * description NUL-

terminated) →int

可以检索有关可以属性的属性的信息为当前词法分析器设置。此信息仅适用于较新的词法分析器。**SCI_PROPERTYNAMES** 返回一个字符串，其中所有有效属性都以“\n”分隔。如果词法分析器不支持此调用，则返回空字符串。属性可以是 **boolean (SC_TYPE_BOOLEAN)**，**integer (SC_TYPE_INTEGER)** 或 **string (SC_TYPE_STRING)**，可以找到它 **SCI_PROPERTYTYPE**。英语属性的描述由返回 **SCI_DESCRIBEPROPERTY**。

SCI_SETPROPERTY (const char * key, const char * value)

您可以使用关键字：值字符串对将设置传递给词法分析器。除可用内存外，您可以设置的关键字对数量没有限制。**key** 是区分大小写的关键字，**value** 是与关键字关联的字符串。如果已存在与该关键字关联的值字符串，则将其替换。如果传递零长度字符串，则消息不执行任何操作。既 **key** 和 **value** 被不加修改地使用；开头或结尾的额外空间 **key** 很重要。

该 **value** 字符串可以引用其他关键字。例如，**SCI_SETPROPERTY("foldTimes10", "\$ (fold)0")** 存储字符串 **"\$ (fold)0"**，但是当访问它时，将 **"\$ (fold)"** 替换为 **"fold"** 关键字的值（如果此关键字不存在，则替换为空）。

目前，如果设置为“1”，则为大多数词法分析器定义“折叠”属性以设置折叠结构。**SCLEX_PYTHON** 理解 **"tab.timmy.whinge.level"** 为确定如何指示不良缩进的设置。大多数关键字都具有被解释为整数的值。搜索词法分析器源 **GetPropertyInt** 以查看属性的使用方式。

有一个用于命名词法分析器使用的属性的约定，以便脚本可以找到属性集。属性名称应以“**lexer.<lexer>**”开头。或者“**折叠.<lexer>**。”当他们申请一个词法分子或以“词法分子”开头时。或“折叠”。如果它们适用于多个词法分析器。

应用程序可以通过搜索词法分析器的源代码来查找包含 **GetProperty** 和双引号字符串的行所使用的属性集，并提取双引号字符串的值作为属性名称。该 **scintilla/scripts/LexGen.py** 脚本执行此操作并可用作示例。该物业的文档可能位于通话上方，作为以。开头的多行注释

```
// property <property-name>
```

SCI_GETPROPERTY (const char * key, char * value NUL-terminated) →int

查找关键字：使用指定键的值对；如果找到，将值复制到用户提供的缓冲区并返回长度（不包括终止 0）。如果未找到，请将空字符串复制到缓冲区并返回 0。

请注意，将不执行如上所述的“关键字替换”。 **SCI_SETPROPERTY**

如果 **value** 参数为 0，则返回应分配用于存储该值的长度；再次，不包括终止 0。

SCI_GETPROPERTYEXPANDED (const char * key, char * value) →int

使用指定的键查找关键字：值对；如果找到，将值复制到用户提供的缓冲区并返回长度（不包括终止 0）。如果未找到，请将空字符串复制到缓冲区并返回 0。

注意，将执行如上所述的“关键字替换”。 [SCI_SETPROPERTY](#)

如果 `value` 参数为 0，则返回应分配用于存储值的长度（包括任何指示的关键字替换）；再次，不包括终止 0。

SCI_GETPROPERTYINT (`const char * key, int defaultValue`) → `int`

使用指定的键查找关键字：值对；如果找到，将该值解释为整数并返回它。如果未找到（或值为空字符串），则返回提供的默认值。如果找到关键字：值对但不是数字，则返回 0。

请注意，将在任何数字解释之前执行所描述的“关键字替换”。 [SCI_SETPROPERTY](#)

SCI_SETKEYWORDS (`int keyWordSet, const char * keyWords`)

您最多可以设置 9 个关键字列表供当前词法分析器使用。`keyWordSet` 可以是 0 到 8（实际上是 0 到 `KEYWORDSET_MAX`）并选择要替换的关键字列表。`keyWords` 是由空格，制表符“\n”或“\r”这些关键字分隔的关键字列表。预计关键字将由标准的 ASCII 打印字符组成，但没有什么可以阻止您使用 1 到 255 之间的任何非分隔符字符代码（常识除外）。

如何使用这些关键字完全取决于词法分析器。某些语言（如 HTML）可能包含嵌入式语言，VBScript 和 JavaScript 在 HTML 中很常见。对于 HTML，关键字集 0 用于 HTML，1 用于 JavaScript，2 用于 VBScript，3 用于 Python，4 用于 PHP，5 用于 SGML 和 DTD 关键字。查看词法分析器代码以查看关键字列表的示例。完全符合的词法分析器将 `LexerModule` 构造函数的第四个参数设置为描述关键字列表用法的字符串列表。

或者，您可以将 set 0 用于常规关键字，将 1 设置为导致缩进的关键字，将 set 2 设置为导致未压缩的关键字。再一次，你可能有一个简单的词法分析器，它可以为关键字着色，你可以通过改变第 0 组中的关键词来改变语言。没有什么可以阻止你在词法分析器中建立自己的关键词列表，但这意味着必须重建词法分析器。添加了更多关键字。

SCI_DESCRIBEKEYWORDSETS (`<unused>, char *描述 NUL 终止`) → `int`
返回由“\n”分隔的所有关键字集的描述 `SCI_DESCRIBEKEYWORDSETS`。

个子风格

词典可能支持几种不同的子语言，每个子语言可能希望唯一地设置一定数量的标识符（或类似的词汇，如文档关键词）。为每个目的预分配大量数字会快速耗尽允许的样式数量。这可以通过子类来缓解，这些子类允许应用程序确定为每个目的分配多少组标识符。Lexers 必须通过实现特定方法明确支持此功能。

SCI_GETSUBSTYLEBASES (`<unused>, char * styles NUL-terminated`)

→ `int` 为每个可以拆分为子类的样式

填充 `styles` 一个字节。

SCI_DISTANCETOSECONDARYSTYLES→int

返回主样式与其对应的辅助样式之间的距离。

SCI_ALLOCATESUBSTYLES (int styleBase, int numberStyles) →int

为特定基本样式分配一定数量的子类，返回分配的第一个子代数。子样式是连续分配的。

SCI_FREESUBSTYLES

释放所有分配的子代码。

SCI_GETSUBSTYLESSTART (int styleBase) →int**SCI_GETSUBSTYLESLENGTH (int styleBase) →int**

返回为基本样式分配的子类的开始和长度。

SCI_GETSTYLEFROMSUBSTYLE (int subStyle) →int

对于子样式，返回基本样式，否则返回参数。

SCI_GETPRIMARYSTYLEFROMSTYLE (int style) →int

对于辅助样式，返回主样式，否则返回参数。

SCI_SETIDENTIFIERS (int style, const char * identifiers)

类似于 **SCI_SETKEYWORDS** 但是对于 substyles。可用的前缀功能 **SCI_SETKEYWORDS** 未实现 **SCI_SETIDENTIFIERS**。

SCI_PRIVATELEXERCALL (int operation, int pointer) →int

以 Scintilla 不理解的方式调用词法分析器。

风格元数据

Lexers 可能会提供他们使用的样式的信息。Lexers 必须通过实现特定方法明确支持此功能。

SCI_GETNAMEDSTYLES→int

检索词法分析器的命名样式数。

SCI_NAMEOFSTYLE (int style, char * name) →int

检索样式的名称。这是一个 C 预处理器符号，如“SCE_C_COMMENTDOC”。

SCI_TAGSOFFSTYLE (int style, char * tags) →int

检索样式的标记。这是一组以空格分隔的单词，如“评论文档”。

SCI_DESCRIPTIONOFFSTYLE (int style, char * description) →int

检索可能适合在用户界面中显示的样式的英语描述。这看起来像“文档评论：阻止以/**或/ *! 开头的评论”。

Lexer 对象

Lexer 被编程为实现 `ILexer4` 接口的对象，并通过 `IDocument` 接口与它们正在进行的文档交互。以前的词法分析器是通过提供 `lexing` 和折叠函数来定义的，但是创建一个对象来处理词法分析器与文档的交互允许词法分析器存储可以在 `lexing` 期间使用的状态信息。例如，C++ 词法分析器可以存储一组预处理器定义或变量声明，并根据它们的角色设置样式。

一组辅助类允许在 Scintilla 中使用由函数定义的较旧词法分析器。

ILexer4

```
class ILexer4 { public : virtual int SCI_METHOD Version ()  
const = 0 ; virtual void SCI_METHOD Release () = 0 ; virtual  
const char * SCI_METHOD PropertyNames () = 0 ; virtual int  
SCI_METHOD PropertyType (const char * name ) = 0 ; 虚拟  
const char *  
  
SCI_METHOD DescribeProperty (const char * name ) = 0 ;  
虚拟 Sci_Position SCI_METHOD PropertySet (const char * key ,  
const char * val ) = 0 ; virtual const char * SCI_METHOD  
DescribeWordListSets () = 0 ; 虚拟 Sci_Position SCI_METHOD  
WordListSet (int n , const  
  
char * wl ) = 0 ; virtual void SCI_METHOD Lex  
( Sci_PositionU startPos , Sci_Position lengthDoc , int initStyle ,  
IDocument * pAccess ) = 0 ; virtual void SCI_METHOD Fold  
( Sci_PositionU startPos , Sci_Position lengthDoc , int initStyle ,  
IDocument * pAccess ) = 0 ;  
  
virtual void * SCI_METHOD PrivateCall (int operation ,  
void * pointer ) = 0 ; virtual int SCI_METHOD  
LineEndTypesSupported () = 0 ; virtual int SCI_METHOD  
AllocateSubStyles (int styleBase , int numberStyles ) = 0 ;  
virtual int SCI_METHOD SubStylesStart (int styleBase ) = 0  
  
; virtual int SCI_METHOD SubStylesLength (int styleBase )  
= 0 ; virtual int SCI_METHOD StyleFromSubStyle (int subStyle )  
= 0 ; virtual int SCI_METHOD PrimaryStyleFromStyle (int style )  
= 0 ; virtual void SCI_METHOD FreeSubStyles () = 0 ; virtual  
void SCI_METHOD SetIdentifiers (
```

```

    int style , const char * identifiers ) = 0 ; virtual int
    SCI_METHOD DistanceToSecondaryStyles ( ) = 0 ; virtual const
    char * SCI_METHOD GetSubStyleBases ( ) = 0 ; virtual int
    SCI_METHOD NamedStyles ( ) = 0 ; virtual const char *
    SCI_METHOD NameOfStyle ( int style ) = 0

    ; virtual const char * SCI_METHOD TagsOfStyle ( int style )
    = 0 ; virtual const char * SCI_METHOD DescriptionOfStyle ( int
    style ) = 0 ; };

```

类型 **Sci_Position** 和 **Sci_PositionU** 用于文档中的位置和行号。使用 Scintilla 4.64 位版本将这些版本定义为 64 位类型，以便将来实现大于 2 GB 的文档。

返回字符串的方法 **const char *** 不需要无限期地维护单独的分配：词法分析器实现可能拥有一个为每次调用重用的缓冲区。调用者应该立即复制返回的字符串。

PropertySet 和 **WordListSet** 的返回值用于指示更改是否需要任何文档执行 **lexing** 或折叠。它是重新启动 **lexing** 和折叠的位置，如果更改不需要对文档进行任何额外的工作，则为 -1。一个简单的方法是返回 0，如果有可能更改需要再次文档，而优化可以记住设置首先影响文档并返回该位置。

Version 返回枚举值，指定实现接口的版本：**lvRelease4for ILexer4**。在 Scintilla 4.0 之前，可能有不同的值。

Release 被调用来摧毁词法分析器对象。


PrivateCall 允许应用程序和词法分析器之间的直接通信。一个示例是应用程序维护包含有关系统标头（如 **Windows.h**）的符号信息的单个大型数据结构，并将其提供给可以应用于每个文档的词法分析器。这避免了为每个文档构造系统头信息的成本。这是使用 **SCI_PRIVATELEXERCALLAPI** 调用的。

Fold 调用具有需要折叠的确切范围。以前，词法分析器的调用范围是在需要折叠的范围之前开始一行，因为这允许修复前一次折叠的最后一行。新方法允许词法分析者决定是回溯还是更有效地处理这个问题。

NamedStyles, **NameOfStyle**, **TagsOfStyle**, 和 **DescriptionOfStyle** 用于提供关于设定的这个词法分析器使用样式的信息。**NameOfStyle** 是像“**SCE_LUA_COMMENT**”这样的 C 语言标识符。**TagsOfStyle** 是一组用标准化方式描述样式的标签，如“**literal string multiline raw**”。[这里描述](#)了一组用于组合它们的通用标签和约定。**DescriptionOfStyle** 是一种英文描述的风格，如“**Function or method name definition**”。

的 IDocument

```
class IDocument { public : virtual int SCI_METHOD Version ()  
const = 0 ; virtual void SCI_METHOD SetErrorStatus ( int status )  
= 0 ; 虚拟 Sci_Position SCI_METHOD 长度 () const = 0 ; virtual  
void SCI_METHOD GetCharRange ( char * buffer , Sci_Position  
position , Sci_Position  
  
lengthRetrieve ) const = 0 ; virtual char SCI_METHOD  
StyleAt ( Sci_Position position ) const = 0 ; 虚拟 Sci_Position  
SCI_METHOD LineFromPosition ( Sci_Position position ) const =  
0 ; 虚拟 Sci_Position SCI_METHOD LineStart ( Sci_Position 行 )  
const = 0 ; virtual int SCI_METHOD GetLevel ( Sci_Position 行 ) const = 0 ; virtual int SCI_METHOD  
SetLevel ( Sci_Position line , int level ) = 0 ; virtual int  
SCI_METHOD GetLineState ( Sci_Position line ) const = 0 ;  
virtual int SCI_METHOD SetLineState ( Sci_Position line , int  
state ) = 0 ; 虚拟空虚  
  
SCI_METHOD StartStyling ( Sci_Position position ) = 0 ;  
virtual bool SCI_METHOD SetStyleFor ( Sci_Position length ,  
char style ) = 0 ; virtual bool SCI_METHOD SetStyles  
( Sci_Position length , const char * styles ) = 0 ; virtual void  
SCI_METHOD DecorationSetCurrentIndicator ( int indicator )  
  
= 0 ; 虚拟 void SCI_METHOD DecorationFillRange  
( Sci_Position position , int value , Sci_Position fillLength ) = 0 ;  
virtual void SCI_METHOD ChangeLexerState ( Sci_Position start ,  
Sci_Position end ) = 0 ; 虚拟 INT SCI_METHOD 代码页 () const 的  
= 0 ; virtual bool SCI_METHOD IsDBCSLeadByte  
  
( char ch ) const = 0 ; virtual const char * SCI_METHOD  
BufferPointer () = 0 ; virtual int SCI_METHOD  
GetLineIndentation ( Sci_Position line ) = 0 ; 虚拟 Sci_Position  
SCI_METHOD LineEnd ( Sci_Position line ) const = 0 ; 虚拟  
Sci_Position SCI_METHOD GetRelativePosition ( Sci_Position  
  
positionStart , Sci_Position characterOffset ) const = 0 ;  
virtual int SCI_METHOD GetCharacterAndWidth ( Sci_Position  
position , Sci_Position * pWidth ) const = 0 ; };
```

Scintilla 尝试尽可能减少修改文本的后果，以便尽可能地改变文本并重新绘制更改行。Lexer 对象包含它们自己的私有额外状态，这可能会影响后面的行。例如，如果 C++ 词法分析器使非活动代码段变灰，那么将语句更改 `#define BEOS 0` 为 `#define BEOS 1` 可能需要重新设置并重新显示文档的后续部分。词法分析器可以调用 `ChangeLexerState` 向文档发出信号，表明它应该反射并显示更多信息。

对于 `StartStylingmask` 参数没有任何影响。它在 3.4.2 及更早版本中使用。

`SetErrorStatus` 用于通知文档的例外情况。不应该在构建边界上抛出异常，因为双方可能使用不同的编译器或不兼容的异常选项构建。

允许词法分析器确定一行的结束位置，从而更容易支持 Unicode 行结束，`IDocument` 包括 `LineEnd` 哪些应该使用而不是测试特定的行结束符。

`GetRelativePosition` 按整个字符导航文档，返回 `INVALID_POSITION` 超出文档开头和结尾的移动。

`GetCharacterAndWidth` 提供从 UTF-8 字节到 UTF-32 字符或从 DBCS 到 16 位值的标准转换。无效 UTF-8 中的字节将单独报告，值为 `0xDC80 + byteValue`，这些值不是有效的 Unicode 代码点。*width* 如果调用者不需要知道字符中的字节数，则该参数可以为 `NULL`。

该 `ILexer4` 和 `IDocument` 接口可能会在未来扩展版本（扩展 `ILexer5...`）。该 `Version` 方法指示实现哪个接口，从而可以调用哪些方法。

Scintilla 4 `IDocument` 通过合并先前单独的 `IDocumentWithLineEnd` 接口并删除 `mask` 参数来更改定义 `StartStyling`。

通知

当发生可能对容器感兴趣的事件时，通知从 Scintilla 控件发送（触发）到其容器。

使用 `WM_NOTIFYWindows` 上的消息发送通知。

在 GTK+ 上，发送“sci-notify”信号，信号处理程序应具有签名 `handler(GtkWidget *, gint, SCNotification *notification, gpointer userData)`。

在 Cocoa 上，实现 `the` 的委托 `ScintillaNotificationProtocol` 可以设置为接收通知，或者 `ScintillaView` 类可以被子类化并且 `notification:` 方法被覆盖。覆盖 `notification:` 允许子类控制是否执行默认处理。

容器传递一个 **SCNotification** 包含有关事件信息的结构。

```
struct Sci_NotifyHeader { //这与 Win32 NMHDR 结构匹配
    void * hwndFrom; //特定于环境的窗口句柄/指针
    uptr_t idFrom; //发出通知的窗口的 CtrlID
    unsigned int 代码; // SCN_ *通知代码
};

struct SCNotification {
    struct Sci_NotifyHeader nmhdr;
    Sci_Position 位置;
    / * SCN_STYLENEEDED, SCN_DOUBLECLICK, SCN_MODIFIED,
SCN_MARGINCLICK, * /
    / * SCN_MARGINRIGHTCLICK, SCN_NEEDSHOWN, SCN_DWELLSTART,
SCN_DWELLEND, * /
    / * SCN_CALLTIPCLICK, SCN_HOTSPOTCLICK, SCN_HOTSPOTDOUBLECLICK,
* /
    / * SCN_HOTSPOTRELEASECLICK, SCN_INDICATORCLICK,
SCN_INDICATORRELEASE, * /
    / * SCN_USERLISTSELECTION, SCN_AUTOCSELECTION,
SCN_AUTOCSELECTIONCHANGE * /

    int ch;
    / * SCN_CHARADDED, SCN_KEY, SCN_AUTOCCOMplete,
SCN_AUTOCSELECTION, * /
    / * SCN_USERLISTSELECTION * /
    int 修饰符;
    / * SCN_KEY, SCN_DOUBLECLICK, SCN_HOTSPOTCLICK,
SCN_HOTSPOTDOUBLECLICK, * /
    / * SCN_HOTSPOTRELEASECLICK, SCN_INDICATORCLICK,
SCN_INDICATORRELEASE, * /

    int modificationType; / * SCN_MODIFIED * /
    const char * text;
    / * SCN_MODIFIED, SCN_USERLISTSELECTION, SCN_AUTOCSELECTION,
SCN_URIDROPPED, * /
    / * SCN_AUTOCSELECTIONCHANGE * /

    Sci_Position 长度; / * SCN_MODIFIED * /
    Sci_Position linesAdded; / * SCN_MODIFIED * /
    int message; / * SCN_MACRORECORD * /
    uptr_t wParam; / * SCN_MACRORECORD * /
    sptr_t lParam; / * SCN_MACRORECORD * /
    Sci_Position 线; / * SCN_MODIFIED * /
    int foldLevelNow; / * SCN_MODIFIED * /
    int foldLevelPrev; / * SCN_MODIFIED * /
    int margin; / * SCN_MARGINCLICK, SCN_MARGINRIGHTCLICK * /
    int listType; / * SCN_USERLISTSELECTION,
SCN_AUTOCSELECTIONCHANGE * /
    int x; / * SCN_DWELLSTART, SCN_DWELLEND * /
    int y; / * SCN_DWELLSTART, SCN_DWELLEND * /
    int 标记; / * SCN_MODIFIED 与 SC_MOD_CONTAINER * /
    int annotationLinesAdded; / * SCN_MODIFIED 与
SC_MOD_CHANGEANNOTATION * /
    int 更新; / * SCN_UPDATEUI * /
    int listCompletionMethod;
```

```

        / * SCN_AUTOCSELECTION, SCN_AUTOCCOMPLETED,
        SCN_USERLISTSELECTION * /

};

```

容器可以选择处理的通知消息以及与它们关联的消息是：

```

SCN_STYLENEEDED
SCN_CHARADDED
SCN_SAVEPOINTREACHED
SCN_SAVEPOINTLEFT
SCN_MODIFYATTEMPTRO
SCN_KEY
SCN_DOUBLECLICK
SCN_UPDATEUI
SCN_MODIFIED
SCN_MACRORECORD
SCN_MARGINCLICK
SCN_NEEDSHOWN
SCN_PAINTED
SCN_USERLISTSELECTION
SCN_URIDROPPED
SCN_DWELLSTART
SCN_DWELLEND
SCN_ZOOM
SCN_HOTSPOTCLICK
SCN_HOTSPOTDOUBLECLICK
SCN_HOTSPOTRELEASECLICK
SCN_INDICATORCLICK
SCN_INDICATORRELEASE
SCN_CALLTIPCLICK
SCN_AUTOCSELECTION
SCN_AUTOCCANCELLED
SCN_AUTOCCCHARDELETED
SCN_FOCUSIN
SCN_FOCUSOUT
SCN_AUTOCCOMPLETED
SCN_MARGINRIGHTCLICK
SCN_AUTOCSELECTIONCHANGE

```

以下 **SCI_***消息与这些通知相关联：

```

SCI_SETMODEEVENTMASK(int eventMask)
SCI_GETMODEEVENTMASK → int
SCI_SETCOMMANDEVENTS(bool commandEvents)
SCI_GETCOMMANDEVENTS → bool
SCI_SETMOUSEDWELLTIME(int periodMilliseconds)
SCI_GETMOUSEDWELLTIME → int
SCI_SETIDENTIFIER(int identifier)
SCI_GETIDENTIFIER → int

```

使用辅助“命令”方法发送以下附加通知，并且应该在新代码中避免使用，因为主要“通知”方法提供具有更丰富信息的所有相同事件。该 **WM_COMMAND** 消息在 Windows 上使用。这将模拟 Windows 编辑控件。在这些通知中仅传递控件 ID 的低 16 位。

在 GTK +上，发送“命令”信号，信号处理程序应具有签名 `handler(GtkWidget *, gint wParam, gpointer lParam, gpointer userData)`。

[SCEN_CHANGE](#)
[SCEN_SETFOCUS](#)
[SCEN_KILLFOCUS](#)

SCI_SETIDENTIFIER (int identifier)

SCI_GETIDENTIFIER→int

这两个消息设置并获取作为 `idFrom` 字段包含在通知中的 Scintilla 实例的标识符。当应用程序创建多个 Scintilla 小部件时，这允许找到每个通知的来源。在 Windows 上，此值在 `CreateWindow` 调用中初始化并存储为 `GWLP_ID` 窗口的属性。该值应该很小，最好小于 16 位，而不是指针，因为某些函数只能传输 16 或 32 位。

SCN_STYLENEEDED

如果您曾经 [SCI_SETLEXER\(SCLEX_CONTAINER\)](#) 使容器充当词法分析器，当 Scintilla 即将显示或打印需要样式化的文本时，您将收到此通知。您需要设置包含返回位置的行的文本样式，[SCI_GETENDSTYLED](#) 直到传入的位置

`SCNotification.position`。象征性地，您需要以下形式的代码：

```
startPos = SCI\_GETENDSTYLED ()  
lineNumber = SCI\_LINEFROMPOSITION (startPos) ;  
startPos = SCI\_POSITIONFROMLINE (lineNumber) ;  
MyStyleRoutine (startPos, SCNotification.position) ;
```

SCN_CHARADDED

当用户键入输入到文本中的普通文本字符（而不是命令字符）时发送。容器可以使用它来决定显示[呼叫提示](#)或[自动完成列表](#)。角色在 `SCNotification.ch`。此通知是在角色设置样式之前发送的，因此应该在 `SCN_UPDATEUI` 通知中执行取决于样式的处理。

SCN_SAVEPOINTREACHED

SCN_SAVEPOINTLEFT

在输入或保留点时发送到容器，允许容器显示“文档脏”指示符并更改其菜单。

另见：[SCI_SETSAVEPOINT](#)，[SCI_GETMODIFY](#)

SCN_MODIFYATTEMPTRO

当处于只读模式时，如果用户尝试更改文本，则会将此通知发送到容器。这可用于从版本控制系统中检查文档。您可以使用设置文档的只读状态

[SCI_SETREADONLY](#)。

SCN_KEY

报告按下但未被 Scintilla 消耗的所有按键。由于键盘焦点存在问题而在 GTK +上使用，并且不是由 Windows 版本发送的。`SCNotification.ch` 保存关键代码并 `SCNotification.modifiers` 保存修饰符。如果修饰符包含 `SCMOD_ALT` 或 `SCMOD_CTRL` 且密钥代码小于 256，则发送此通知。

SCN_DOUBLECLICK

在编辑器中双击鼠标按钮。该 **position** 字段设置为双击的文本位置，该 **line** 字段设置为双击的行，并且该 **modifiers** 字段设置为以与 **SCN_KEY** 类似的方式按下的键修饰符。

SCN_UPDATEUI

文档的文本或样式已更改，或者选择范围或滚动位置可能已更改。现在是更新依赖于文档或视图状态的任何容器 UI 元素的好时机。由于有时很难确定是否发生了变化，因此当没有实际变化时，这些事件也可能会触发。该 **updated** 字段设置为自上次通知以来更改的位集。

符号	值	含义
SC_UPDATE_CONTENT	0x01	内容，样式或标记可能已更改。
SC_UPDATE_SELECTION	0x02	选择可能已经改变。
SC_UPDATE_V_SCROLL	0x04	可能已垂直滚动。
SC_UPDATE_H_SCROLL	0x08 的	可能已水平滚动。

SCN_MODIFIED

当文档的文本或样式更改或即将更改时，将发送此通知。您可以为发送到容器的通知设置掩码 **SCI_SETMODEVENTMASK**。通知结构包含有关更改内容，更改方式以及是否更改了文档中的行数的信息。在 **SCN_MODIFIED** 事件中不能进行任何修改。使用的 **SCNotification** 字段是：

领域	用法
modificationType	一组标志，用于标识所做的更改。见下表。
position	开始文本或样式更改的位置。如果不使用，设置为 0。
length	文本或样式更改时的字节更改长度。如果不使用，设置为 0。
linesAdded	添加的行数。如果为负数，则删除行数。如果未使用或未添加或删除行，则设置为 0。
text	适用于文本更改，而不适用于样式更改。如果我们正在收集撤消信息，则会保存一个指向 Undo 系统的文本的指针，否则它为零。对于用户执行的 SC_MOD_BEFOREDELETE ，文本字段为 0。
line	发生折叠级别或标记更改的行号。如果未使用，则为 0，如果更改了多行，则为 -1。
foldLevelNow	应用于该行的新折叠级别，如果未使用此字段，则为 0。
foldLevelPrev	行的先前折叠级别，如果未使用此字段，则为 0。

该 **SCNotification.modificationType** 字段的位设置为告诉您已执行的操作。这些 **SC_MOD_*** 位对应于动作。这些 **SC_PERFORMED_*** 位告诉您操作是由用户完成的，还是先前操作的撤消或重做的结果。

符号	值	含义	SCNotification 字段
SC_MOD_INSERTTEXT	0x01	文本已插入文档中。	position, length, text, linesAdded
SC_MOD_DELETETEXT	0x02	文本已从文档中删除。	position, length, text, linesAdded
SC_MOD_CHANGESTYLE	0x04	风格发生了变化。	position, length
SC_MOD_CHANGEFOLD	0x08 的	发生了折叠变化。	line, foldLevelNow, foldLevelPrev
SC_PERFORMED_USER	为 0x10	信息：操作由用户完成。	没有
SC_PERFORMED_UNDO	为 0x20	信息：这是撤消的结果。	没有
SC_PERFORMED_REDO	0x40 的	信息：这是重做的结果。	没有
SC_MULTISTEPUNDOREDO	0x80 的	这是多步撤消或重做事务的一部分。	没有
SC_LASTSTEPINUNDOREDO	为 0x100	这是撤消或重做事务的最后一步。	没有
SC_MOD_CHANGEMARKER	在 0x200	一个或多个标记已在一行中发生变化。	line
SC_MOD_BEFOREINSERT	0x400 的	文本即将插入到文档中。	position, if performed by user then text in bytes, length in bytes
SC_MOD_BEFOREDELETE	为 0x800	文本即将从文档中删除。	position, length
SC_MOD_CHANGEINDICATOR	0x4000 的	已在一系列文本中添加或删除指标。	position, length
SC_MOD_CHANGELINESTATE	为 0x8000	线状态已更改，因为已调用 SCI_SETLINESTATE 。	line
SC_MOD_CHANGETABSTOPS	0x200000	由于 SCI_CLEARABSTOPS 或 SCI_ADDTABSTOP 被调用，因此一行中的显式制表位已更改。	line
SC_MOD_LEXERSTATE	0x80000	词法分析器的内部状态在一定范围内发生了变化。	position, length
SC_MOD_CHANGEMARGIN	为 0x10000	文本边距已更改。	line
SC_MOD_CHANGEANNOTATION	地址 0x20000	注释已更改。	line

SC_MOD_INSERTCHECK	0x100000	即将插入文本。处理程序可以通过调用 SCI_CHANGEINSERTION 来更改正在插入的文本。此处理程序中不能进行任何其他修改。	position, length, text
SC_MULTILINEUNDOREDO	为 0x1000	这是具有多行更改的撤销或重做的一部分。	没有
SC_STARTACTION	为 0x2000	当它是撤销事务中的第一步或唯一步骤时，它在 SC_PERFORMED_USER 操作上设置。这可以用于将 Scintilla 撤销堆栈与容器应用程序中的撤销堆栈集成，方法是将 Scintilla 操作添加到当前打开的容器事务的容器堆栈中，或者如果没有打开的容器事务，则打开新的容器事务。	没有
SC_MOD_CONTAINER	0x40000	这是为容器存储到撤销堆栈中的操作设置的 SCI_ADDUNDOACTION 。	代币
SC_MODEVENTMASKALL	0x1FFFFFFF	这是所有有效标志的掩码。这是设置的默认掩码状态 SCI_SETMODEVENTMASK 。	没有

SCEN_CHANGE 当文档的文本（而不是样式）发生变化时，将触发 **SCEN_CHANGE**（768）。此通知使用 **WM_COMMAND** Windows 上的消息和 **GTK +** 上的“命令”信号发送，因为这是标准 **Edit** 控件的行为（**SCEN_CHANGE** 与 Windows **Edit** 控件具有相同的值 **EN_CHANGE**）。没有其他信息发送。如果您需要更详细的信息使用 **SCN_MODIFIED**。您可以过滤使用 **SCI_SETMODEVENTMASK** 和 通知的更改类型 **SCI_SETCOMMANDEVENTS**。

SCI_SETMODEVENTMASK (int eventMask)

SCI_GETMODEVENTMASK→int

这些消息设置并获取一个事件掩码，用于确定使用 **SCN_MODIFIED** 和向容器通知哪些文档更改事件 **SCEN_CHANGE**。例如，容器可能决定只查看有关文本更改的通知，而不是通过调用来查看样式更改

SCI_SETMODEVENTMASK(SC_MOD_INSERTTEXT|SC_MOD_DELETETEXT)。

可能的通知类型是一样的 `modificationType` 通过使用位标志 `SCN_MODIFIED`：
`SC_MOD_INSERTTEXT`, `SC_MOD_DELETETEXT`, `SC_MOD_CHANGESTYLE`,
`SC_MOD_CHANGEFOLD`, `SC_PERFORMED_USER`, `SC_PERFORMED_UNDO`, `SC_PERFORMED_REDO`,
`SC_MULTISTEPUNDOREDO`, `SC_LASTSTEPINUNDOREDO`, `SC_MOD_CHANGEMARKER`,
`SC_MOD_BEFOREINSERT`, `SC_MOD_BEFOREDELETE`, `SC_MULTILINEUNDOREDO`, 和
`SC_MODEVENTMASKALL`。

SCI_SETCOMMANDEVENTS (bool commandEvents)

SCI_GETCOMMANDEVENTS → bool

这些消息设置并获取是否将 `SCEN_*` 命令事件发送到容器。对于 `SCEN_CHANGE` 这个作为额外的过滤器 `SCI_SETMODEVENTMASK`。大多数应用程序应该将其设置为避免开销并且仅使用 `SCN_MODIFIED`。

SCEN_SETFOCUS

SCEN_KILLFOCUS

`SCEN_SETFOCUS` (512) 在 Scintilla 获得焦点时被触发，而 `SCEN_KILLFOCUS` (256) 在失去焦点时触发。这些通知使用 `WM_COMMAND` Windows 上的消息和 `GTK+` 上的“命令”信号发送，因为这是标准 Edit 控件的行为。不幸的是，这些代码与 Windows 编辑通知代码 `EN_SETFOCUS` (256) 和 `EN_KILLFOCUS` (512) 不匹配。现在改变 Scintilla 代码为时已晚，因为客户端依赖于当前值。

SCN_MACRORECORD

的 `SCI_STARTRECORD` 和 `SCI_STOPRECORD` 消息启用和禁用宏录制。启用后，每次发生可记录的更改时，都会将 `SCN_MACRORECORD` 通知发送到容器。由容器记录动作。要查看完整列表 `SCI_*` 是记录的信息，搜索 Scintilla 的来源 `Editor.cxx` 的 `Editor::NotifyMacroRecord`。 `SCNotification` 此通知中设置的字段为：

领域	用法
<code>message</code>	<code>SCI_*</code> 导致通知的消息。
<code>wParam</code>	的值 <code>wParam</code> 中 <code>SCI_*</code> 的消息。
<code>lParam</code>	的值 <code>lParam</code> 中 <code>SCI_*</code> 的消息。

SCN_MARGINCLICK

SCN_MARGINRIGHTCLICK

这些通知告诉容器在标记为敏感的边距内单击鼠标或右键单击鼠标（请参阅参考资料 `SCI_SETMARGINSSENSITIVEN`）。这可用于执行折叠或放置断点。使用以下 `SCNotification` 字段：

领域	用法
<code>modifiers</code>	适当组合 <code>SCI_SHIFT</code> , <code>SCI_CTRL</code> 并 <code>SCI_ALT</code> 指示在保证金点击时按下的键。
<code>position</code>	文档中行的开头位置，对应于边距点击。
<code>margin</code>	已点击的保证金编号。

SCN_NEEDSHOWN

Scintilla 已经确定应该可以看到当前不可见的一系列线条。可能需要这样做的一个例子是如果删除了缩小折叠点的行尾。此消息将发送到容器，以防它以某种不寻常的方式使线条可见，例如使整个文档可见。大多数容器只会通过调用确保范围内的每一行都可见 [SCI_ENSUREVISIBLE](#)。在 **position** 和 **length** 领域 **SCNotification** 指示应可见文档的范围。容器代码类似于以下代码框架：

```
firstLine = SCI_LINEFROMPOSITION (scn.position)
lastLine = SCI_LINEFROMPOSITION (scn.position + scn.length-1)
for line = lineStart to lineEnd next SCI_ENSUREVISIBLE (line) next
```

SCN_PAINTED

绘画刚刚完成。当您想要根据 Scintilla 中的更改更新其他一些小部件时很有用，但是想要先使用 **paint** 来显示响应更快。没有其他信息 **SCNotification**。

SCN_USERLISTSELECTION

用户已选择[用户列表](#)中的项目。使用的 **SCNotification** 字段是：

领域	用法
listType	这将设置为启动列表 listType 的 SCI_USERLISTSHOW 消息中的参数。
text	选择的文本。
position	列表显示的位置。
ch	如果填充字符是选择方法，则使用字符，否则为 0。
listCompletionMethod	表示完成发生方式的值。见下表。

请参阅有关 [SCN_AUTOCOMPLETED](#) 可能值的通知 **listCompletionMethod**。

SCN_URIDROPPED

仅适用于 GTK +版本。表示用户已将诸如文件名或 Web 地址之类的 URI 拖到 Scintilla 上。容器可以将其解释为打开文件的请求。该 **text** 场 **SCNotification** 在 URI 文本点。

SCN_DWELLSTART

SCN_DWELLEND

SCN_DWELLSTART 是在用户将鼠标停留在停留时间的一个位置时生成的（请参阅[参考资料 SCI_SETMOUSEDWELLTIME](#)）。**SCN_DWELLEND** 在 a 之后生成 **SCN_DWELLSTART** 并且移动鼠标或其他活动（例如按键）表示停留时间结束。两个通知都设置相同的字段 **SCNotification**：

领域	用法
position	这是文档中最靠近鼠标指针停留位置的位置。
x, y	指针徘徊在哪里。该 position 字段设置为 SCI_POSITIONFROMPOINTCLOSE(x, y) 。

SCI_SETMOUSEDWELLTIME (int periodMilliseconds)

SCI_GETMOUSEDWELLTIME→int

这两条消息设置并获取鼠标必须静止的时间（以毫秒为单位）以生成 **SCN_DWELLSTART** 通知。如果设置为 **SC_TIME_FOREVER** 默认值，则不会生成任何驻留事件。

SCN_ZOOM

当用户使用键盘缩放显示或 **SCI_SETZOOM** 调用方法时，将生成此通知。此通知可用于重新计算位置，例如行号边距的宽度，以按字符而不是像素维持大小。

SCNotification 没有其他信息。

SCN_HOTSPOTCLICK

SCN_HOTSPOTDOUBLECLICK

SCN_HOTSPOTRELEASECLICK

当用户单击或双击具有热点属性集的样式中的文本时，将生成这些通知。此通知可用于链接到变量定义或网页。在通知处理程序中，您应该避免调用任何修改当前选择或插入位置的函数。该 **position** 字段设置为单击或双击的文本位置，并且 **modifiers** 字段设置为按照与 **SCN_KEY** 类似的方式按下的键修饰符。仅报告 Ctrl 键的状态 **SCN_HOTSPOTRELEASECLICK**。

SCN_INDICATORCLICK

SCN_INDICATORRELEASE

当用户在具有指示符的文本上单击或释放鼠标时，将生成这些通知。该 **position** 字段设置为单击或双击的文本位置，并且 **modifiers** 字段设置为按照与 **SCN_KEY** 类似的方式按下的键修饰符。

SCN_CALLTIPCLICK

当用户单击呼叫提示时，将生成此通知。当函数名称使用不同的参数重载时，此通知可用于显示下一个函数原型。该 **position** 字段被设置为 1，如果该点击是在一个向上的箭头，2 如果在一个向下箭头，和 0，如果别处。

SCN_AUTOCELECTION

用户已在 **自动完成列表** 中选择一个项目。在插入选择之前发送通知。可以通过 **SCI_AUTOCCANCEL** 在从通知返回之前发送消息来取消自动插入。使用的 **SCNotification** 字段是：

领域	用法
position	完成单词的开始位置。
text	选择的文本。
ch	如果填充字符是选择方法，则使用字符，否则为 0。
listCompletionMethod	表示完成发生方式的值。见下表。

符号	值	含义
SC_AC_FILLUP	1	填充字符触发完成。使用的字符是 ch 。

- SC_AC_DOUBLECLICK** 2 双击触发完成。ch 为 0。
- SC_AC_TAB** 3 Tab 键或 SCI_TAB 触发完成。ch 为 0。
- SC_AC_NEWLINE** 4 新行或 SCI_NEWLINE 触发完成。ch 为 0。
- SC_AC_COMMAND** 五 该 消息触发了完成。ch 为 0。 **SCI_AUTOCSELECT**

SCN_AUTOCCANCELLED

用户已取消 **自动完成列表**。SCNotification 中没有其他信息。

SCN_AUTOCCCHARDELETED

用户在自动填充列表处于活动状态时删除了一个字符。SCNotification 中没有其他信息。

SCN_AUTOCCOMPLETED

在自动填充插入文本后生成此通知。这些字段与 通知相同 。
SCN_AUTOCSELECTION

SCN_AUTOCSELECTIONCHANGE

在自动完成或用户列表中突出显示项目时发送此通知。使用的 **SCNotification** 字段是：

领域	用法
listType	这将设置为消息中的 listType 参数， SCI_USERLISTSHOW 或者设置为 0 以进行自动完成。
text	选择的文本。
position	列表显示的位置。

SCN_FOCUSIN

SCN_FOCUSOUT

SCN_FOCUSIN（2028）在 Scintilla 获得焦点时被触发，而 **SCN_FOCUSOUT**（2029）在失去焦点时触发 。

图片

边距标记和自动完成列表（RGBA 和 XPM）中使用的图像支持两种格式。

RGBA

RGBA 格式允许 每个像素具有 **alpha** 值的半透明度。它比简单 **XPM** 而且更有能力。

数据是 4 字节像素值的序列，以顶行的像素开始，最左边的像素首先，然后继续后续行的像素。由于对齐原因，线之间没有间隙。

每个像素依次由红色字节，绿色字节，蓝色字节和字母字节组成。颜色字节不会被 alpha 值预乘。也就是说，25%不透明的全红色像素将是[FF, 00,00,3F]

由于 RGBA 像素数据不包含任何大小信息，因此必须先使用 `SCI_RGBAIMAGESETWIDTH` 和 `SCI_RGBAIMAGESETHEIGHT` 消息设置宽度和高度。

GUI 平台通常包括用于将图像文件格式（如 PNG）以 RGBA 形式或类似形式读入存储器的功能。如果没有合适的平台支持，[LodePNG](#) 和 [picoPNG](#) 库是用于加载和解码 BSD 样式许可下可用的 PNG 文件的小型库。

Windows，GTK + 和 OS X Cocoa 支持 RGBA 格式。

XPM

[这里描述了](#) XPM 格式。Scintilla 只能处理每个像素使用一个字符且没有命名颜色的 XPM 像素图。可能有一种名为“无”的完全透明的颜色。

有两种形式的数据结构用于 XPM 图像，第一种“线形式”格式非常适合在 C 源代码中嵌入图像，“文本形式”适合于从文件中读取。在行形式中，使用字符串数组，第一个字符串表示所使用的颜色的尺寸和数量。接下来是每种颜色的字符串，该部分后面是每行一个字符串的图像。文本格式包含与一个空终止块相同的数据，格式化为 C 源代码，以“`/ * XPM * /`”注释开头，以标记格式。

这两种格式都可以与 Scintilla API 一起使用，并且指向要检查的位置处的字节确定哪种格式：如果字节以“`/ * XPM * /`”开头，则将其视为文本形式，否则将其视为行形式。

所有平台都支持 XPM 格式。

GTK +

在 GTK + 上，以下函数创建了一个 Scintilla 小部件，与之通信并允许在所有 Scintilla 小部件被销毁后释放资源。

```
GtkWidget *scintilla_new()
void scintilla_set_id(ScintillaObject *sci, uptr_t id)
sptr_t scintilla_send_message(ScintillaObject *sci,unsigned int iMessage,
uptr_t wParam, sptr_t lParam)
void scintilla_release_resources()
```

GtkWidget * scintilla_new ()

创建一个新的 Scintilla 小部件。返回的指针可以添加到容器中，并以与其他窗口小部件相同的方式显示。

void scintilla_set_id (ScintillaObject * sci, uptr_t id)

设置将在此实例的所有通知的 Sci_NotifyHeader 结构的 idFrom 字段中使用的控件 ID。这相当于 `SCI_SETIDENTIFIER`。

sptr_t scintilla_send_message (ScintillaObject * sci, unsigned int iMessage, uptr_t wParam, sptr_t lParam)

主入口点允许发送本文档中描述的任何消息。

void scintilla_release_resources ()

在销毁所有 Scintilla 小部件后，调用此方法释放所有剩余资源。

临时信息

复杂的新功能可能会添加为“临时”，以允许进一步更改 API。如果经验表明它们是错误的，甚至可以删除临时功能。

临时功能在本文档中显示，具有独特的背景颜色。

一些开发人员可能希望仅使用稳定且从临时状态渐变的功能。为避免使用临时消息，请使用 **SCI_DISABLE_PROVISIONAL** 定义的符号进行编译。

不推荐使用的消息和通知

当前支持以下消息来模拟现有的 Windows 控件，但在将来的 Scintilla 版本中将删除它们。如果您使用这些消息，则应将其替换为 Scintilla 等效消息。

```
WM_GETTEXT (int length, char * text)
WM_SETTEXT (<unused>, const char * text)
EM_GETLINE (int line, char * text)
EM_REPLACESEL (<unused>, const char * text)
EM_SETREADONLY
EM_GETTEXTRANGE (<unused>, TEXTRANGE * tr)
WM_CUT
WM_COPY
WM_PASTE
WM_CLEAR
WM_UNDO
EM_CANUNDO
EM_EMPTYUNDOBUFFER
WM_GETTEXTLENGTH
EM_GETFIRSTVISIBLELINE
EM_GETLINECOUNT
EM_GETMODIFY
EM_SETMODIFY (bool isModified)
EM_GETRECT (RECT * rect)
EM_GETSEL (int * start, int * end)
EM_EXGETSEL (<unused>, CHARRANGE * cr)
EM_SETSEL (int start, int end)
EM_EXSETSEL (<unused>, CHARRANGE * cr)
EM_GETSELTEXT (<unused>, char * text)
EM_LINEFROMCHAR (int position)
EM_EXLINEFROMCHAR (int position)
EM_LINEINDEX (int line)
EM_LINELENGTH (int position)
```

```

EM_SCROLL (int line)
EM_LINESCROLL (int column, int line)
EM_SCROLLCARET ()
EM_CANPASTE
EM_CHARFROMPOS (<unused>, POINT *位置)
EM_POSFROMCHAR (int position, POINT * location)
EM_SELECTIONTYPE
EM_HIDESELECTION (布尔隐藏)
EM_FINDTEXT (int flags, FINDTEXT * ft)
EM_FINDTEXT (int 标志, FINDTEXT * ft)
EM_GETMARGINS
EM_SETMARGINS (EC_LEFTMARGIN 或 EC_RIGHTMARGIN 或 EC_USEFONTINFO, int
val)
EM_FORMATRANGE

```

以下是仅在您定义 **INCLUDE_DEPRECATED_FEATURES** 时包含的功能 **Scintilla.h**。为确保将来的兼容性，您应该按照指示更改它们。

```

SCI_SETKEYSUNICODE(bool keysUnicode)
SCI_GETKEYSUNICODE → bool
SCI_SETTWOPHASEDRAW(bool twoPhase)
SCI_GETTWOPHASEDRAW → bool
SCI_SETSTYLEBITS(int bits)
SCI_GETSTYLEBITS → int
SCI_GETSTYLEBITSNEEDED → int

```

SCI_SETKEYSUNICODE (bool keysUnicode) 不推荐使用 **SCI_GETKEYSUNICODE**→bool 不推荐使用
在 Windows 上，Scintilla 不再支持窄字符窗口，因此输入始终被视为 Unicode。

SCI_SETTWOPHASEDRAW (bool twoPhase)
SCI_GETTWOPHASEDRAW→bool
此属性已被前面的 PHASESDRAW 属性所取代，该属性更为通用，允许多相绘制以及单相和两相绘制。

单相图纸 **SC_PHASES_ONE** 已弃用，应替换为 2 相 **SC_PHASES_TWO** 或多相 **SC_PHASES_MULTIPLE** 图纸。

以下是应从调用代码中删除但仍定义为避免破坏调用者的功能。

SCI_SETSTYLEBITS (INT 位) 已过时
SCI_GETSTYLEBITS→INT 已过时
SCI_GETSTYLEBITSNEEDED→INT 已过时
INDIC0_MASK, INDIC1_MASK, INDIC2_MASK, INDICS_MASK 已过时
的 Scintilla 不再支持风格字节指标。支持样式字节指示符的最后一个版本是 3.4.2。应删除这些符号的任何使用并替换为[标准指示符](#)。**SCI_GETSTYLEBITS** 并且 **SCI_GETSTYLEBITSNEEDED** 总是返回 8，表示 8 位用于样式，并且有 256 种样式。

编辑 Scintilla 从不支持的消息

```
EM_GETWORDBREAKPROC EM_GETWORDBREAKPROCEX  
EM_SETWORDBREAKPROC EM_SETWORDBREAKPROCEX  
EM_GETWORDWRAPMODE EM_SETWORDWRAPMODE  
EM_LIMITTEXT EM_EXLIMITTEXT  
EM_SETRECT EM_SETRECTNP  
EM_FMTLINES  
EM_GETHANDLE EM_SETHANDLE  
EM_GETPASSWORDCHAR EM_SETPASSWORDCHAR  
EM_SETTABSTOPS  
EM_FINDWORDBREAK  
EM_GETCHARFORMAT EM_SETCHARFORMAT  
EM_GETOLEINTERFACE EM_SETOLEINTERFACE  
EM_SETOLECALLBACK  
EM_GETPARAFORMAT EM_SETPARAFORMAT  
EM_PASTESPECIAL  
EM_REQUESTRESIZE  
EM_GETBKGDNDCOLOR EM_SETBKGDNDCOLOR  
EM_STREAMIN EM_STREAMOUT  
EM_GETIMECOLOR EM_SETIMECOLOR  
EM_GETIMEOPTIONS EM_SETIMEOPTIONS  
EM_GETOPTIONS EM_SETOPTIONS  
EM_GETPUNCTUATION EM_SETPUNCTUATION  
EM_GETTHUMB  
EM_GETEVENTMASK  
EM_SETEVENTMASK  
EM_DISPLAYBAND  
EM_SETTARGETDEVICE
```

Scintilla 试图成为标准 Windows Edit 和 RichEdit 控件的超集，只要它有意义。由于它不打算用在文字处理器中，因此无法合理地处理某些编辑消息。不支持的消息无效。

删除功能

这些功能现已完全删除。

SC_CP_DBCS 在 2016 年删除，版本 3.7.1

这用于在 GTK + 上设置 DBCS（双字节字符集）模式。调用 **SCI_SETCODEPAGE** 时应使用显式 DBCS 代码页

SCI_SETUSEPALETTE (bool usePalette) 于 2016 年删除，版本为 3.7.1

SCI_GETUSEPALETTE → bool 2016 年删除版本 3.7.1

Scintilla 不再支持调色板模式。支持调色板的最后一个版本是 2.29。必须删除对这些方法的任何调用。

以前版本的 Scintilla 允许将指标存储在每个样式字节的位中。这在 2007 年被弃用，并在 2014 年以 3.4.3 版删除。样式字节指示符的所有用法都应替换为[标准指标](#)。

建造 Scintilla

要构建 Scintilla 或 SciTE，请参阅 Scintilla 和 SciTE 目录中的 README 文件。编译器必须支持 C++ 17。对于 Windows，可以使用 GCC 7.1 或 Microsoft Visual C++ 2017.5 进行构建。对于 GTK +，应使用 GCC 7.1 或更高版本。glib 2.22+支持 GTK + 2.24 和 3.x. 应自动检测安装的 GTK +版本。当 GTK + 2 和 GTK + 3 都存在时，为 GTK + 3.x 构建需要在命令行上定义 GTK3。

静态链接

在 Windows 上，Scintilla 通常用作动态库作为.DLL 文件。如果要将 Scintilla 直接链接到应用程序.EXE 或.DLL 文件，则可以链接到静态库 `bin / libscintilla.lib`（或.a，如果使用 GCC）并调用 `Scintilla_RegisterClasses`。

`Scintilla_RegisterClasses` 获取 `HINSTANCE` 您的应用程序并确保注册“Scintilla”窗口类。

在生成独立的 Scintilla DLL 时，应该编译和链接 `ScintillaDLL.cxx` 文件以提供 `DllMain` 和 `Scintilla_RegisterClasses`。

确保词法分析与 Scintilla 相关联

根据所使用的编译器和链接器，可能会删除词法分析器。这通常是在构建静态库时引起的。为了确保词法分析器链接，`Scintilla_LinkLexers()` 可以调用该函数。

更改词法分析器集

要更改 Scintilla 中的词法分析器集，请从目录中添加和删除词法分析器源文件（`Lex*.cxx`）`scintilla/lexers directory` 并 `scripts/LexGen.py` 从 `scripts` 目录运行脚本以更新 `make` 文件和 `Catalogue.cxx`。`LexGen.py` 需要 Python 2.5 或更高版本。如果您无法访问 Python，可以 `Catalogue.cxx` 按照其他词法分析器的模式以简单的方式手动编辑。重要的是要包含 `LINK_LEXER(lmMyLexer);` 与 `LexerModule lmMyLexer(...);` 源代码中的对应。

使用替代正则表达式实现构建

一个简单的界面支持在编译时切换正则表达式引擎。您必须 `RegexSearchBase` 为所选引擎实现，查看内置实现 `BuiltinRegex` 以了解如何完成此操作。然后，您需要实现工厂方法 `CreateRegexSearch` 来创建类的实例。您必须通过定义禁用内置实现 `SCI_OWNREGEX`。