XLSX Plugin 1.0

Erstellen und Bearbeiten von XLSX-Dokumenten mit Hollywood

Andreas Falkenhahn

Inhaltsverzeichnis

1 Allgemeine Informationen

1.1 Einführung

Mit dem XLSX-Plugin können Sie bequem XLSX-Dokumente aus Hollywood-Skripten lesen und schreiben. Es bietet eine Vielzahl von Befehlen zum Festlegen und Abrufen von Zellwerten, Zelltypen, Zellformeln, Dokument-/Arbeitsblatteigenschaften und mehreren anderen Attributen. Es bietet auch eine Iterator-Funktion für eine leistungsstarke Iteration einer großen Anzahl von Zellen.

Darüber hinaus unterstützt das XLSX-Plugin auch die Serialisierungsschnittstelle von Hollywood, was bedeutet, dass Sie Hollywood-Tabellen bequem in XLSX-Dokumente serialisieren können, indem Sie nur einen einzigen Aufruf des Befehls `SerializeTable()` von Hollywood ausführen. Auf die gleiche Weise können Sie auch ganze XLSX-Dokumente in Hollywood-Tabellen deserialisieren, indem Sie einfach den Befehl `DeserializeTable()` von Hollywood aufrufen. Einfacher geht es nicht mehr!

Beachten Sie, dass Reiter, Registerkarten, Registerblatt, Tabellenblätter, Arbeitsblätter oder Worksheets in diesem Handbuch einheitlich Arbeitsblätter genannt werden.

1.2 Lizenz

DRITTE DAS PROGRAMM "SO WIE ES IST" ZUR VERFÜGUNG, OHNE IRGENDEINE GARANTIE, WEDER DIREKT NOCH INDIREKT. DIES BEINHALTET, IST ABER NICHT DARAUF BESCHRÄNKT, VERKÄUFLICHKEIT UND EIGNUNG FÜR EINEN BESTIMMTEN VERWENDUNGSZWECK. DAS VOLLSTÄNDIGE RISIKO DER QUALITÄT UND AUSFÜHRBARKEIT DES PROGRAMMS LIEGT BEIM ANWENDER. SOLLTE SICH DAS PROGRAMM ALS DEFEKT HERAUSSTELLEN, LIEGEN ALLE KOSTEN FÜR SERVICE, INSTANDSETZUNG ODER NACHBESSERUNG BEIM ANWENDER.

KEIN COPYRIGHT-INHABER ODER DRITTER, DER DAS PROGRAMM WIE OBEN ERLAUBT WEITERVERKAUFT, KANN FÜR SCHÄDEN IRGENDWELCHER ART HAFTBAR GEMACHT WERDEN (DIES BEINHALTET, IST ABER NICHT BESCHRÄNKT AUF, DATENVERLUST INFOLGE UNFÄHIGKEIT DES PROGRAMMS, MIT ANDEREN PROGRAMMEN ZUSAMMENZUARBEITEN), SELBST WENN EIN SOLCHER INHABER ODER DRITTER AUF DIE MÖGLICHKEIT EINES SOLCHEN SCHADENS HINGEWIESEN WURDE, AUSSER ES BESTEHT EINE SCHRIFTLICHE EINWILLIGUNG ODER WIRD VOM GESETZ VERLANGT.

1.3 Anforderungen

- Hollywood 9.0 oder besser
- macOS-Version erfordert macOS 10.14 oder besser

1.4 Installation

Die Installation von xlsx.hwp ist unkompliziert und einfach: Kopieren Sie einfach die Datei `xlsx.hwp` für die Plattform Ihrer Wahl in Hollywoods Plugin-Verzeichnis. Auf allen Systemen außer auf AmigaOS und kompatiblen müssen Plugins in einem Verzeichnis mit dem Namen `Plugins` gespeichert werden, das sich im selben Verzeichnis wie das Hauptprogramm von Hollywood befindet. Auf AmigaOS und kompatiblen Systemen müssen Plugins stattdessen in `LIBS:Hollywood` installiert werden. Unter macOS X muss sich das Verzeichnis `Plugins` im Verzeichnis `Resources` des Programmpakets befinden, d.h. im Verzeichnis `HollywoodInterpreter.app/Contents/Resources`. Beachten Sie, dass `HollywoodInterpreter.app` im Programmpaket `Hollywood.app` selbst gespeichert ist, nämlich in `Hollywood.app/Contents/Resources`.

Unter Windows sollten Sie auch die Datei `xlsx.chm` in das Verzeichnis `Docs` Ihrer Hollywood-Installation kopieren. Wenn sich dann der Cursor über einem xlsx.hwp-Befehl in der Hollywood-IDE befindet, können Sie die Online-Hilfe aufrufen, indem Sie F1 drücken.

Unter Linux und macOS kopieren Sie das Verzeichnis `xlsx`, das sich im Verzeichnis `Docs` des xlsx.hwp-Distributionsarchivs befindet, in das Verzeichnis `Docs` Ihrer Hollywood-Installation. Beachten Sie, dass sich unter macOS das Verzeichnis `Docs` innerhalb des Programmpakets `Hollywood.app` befindet, d.h. in `Hollywood.app/Contents/Resources/Docs`.

2 Über das XLSX-Plugin

2.1 Danksagungen

xlsx.hwp wurde von Andreas Falkenhahn geschrieben. Dieses Plugin wurde zunächst als Konzeptioneller Nachweis für die neue Serialisierungsschnittstelle von Hollywood 9 entwickelt und später zu einer vollständigen Bibliothek für den Umgang mit XLSX-Dokumenten erweitert. Vielen Dank an Kenneth Troldal Balslev für sein wunderbares OpenXLSX, auf dem dieses Plugin basiert.

Ein besonderer Dank geht an Dominic Widmer und Helmut Haake für die Übersetzung des Handbuchs ins Deutsche. Fehler oder Verbesserungsvorschläge bzgl. des deutschen Handbuchs bitte an das Übersetzungsteam richten, welches unter handbuch@gmx.ch oder https://amiga-resistance.info erreicht werden kann.

Wenn Sie mich kontaktieren möchten, senden Sie bitte eine E-Mail an andreas@airsoftsoftwair . de oder nutzen Sie das Kontaktformular unter http://www.hollywood-mal.com.

2.2 Häufig gestellte Fragen

In diesem Abschnitt werden einige häufig gestellte Fragen behandelt. Bitte lesen Sie sie zuerst, bevor Sie im Forum nachfragen, da Ihr Problem hier möglicherweise behandelt wurde.

F: Wie heissen Reiter, Registerkarten, Registerblatt, Tabellenblätter, Arbeitsblätter oder Worksheets?

A: Bei der Übersetzung des Handbuches ins Deutsche wurde einheitlich Arbeitsblätter verwendet.

F: Gibt es ein Hollywood-Forum, in dem ich mit anderen Benutzern in Kontakt treten kann?

A: Ja, bitte besuchen Sie die "Community" oder "Forum"-Sektion des offiziellen Hollywood-Portals unter http://www.hollywood-mal.com.

F: Wo kann ich um Hilfe bitten?

A: Es gibt ein lebhaftes englischsprachiges Forum auf http://forums.hollywood-mal.com. Sie können gerne Ihre Frage dort stellen. Ausserdem ist ein deutschsprachiges Forum vorhanden, welches Sie unter https://www.amiga-resistance.info/ erreichen können.

F: Ich habe einen Fehler gefunden.

A: Bitte informieren Sie mich darüber in den speziellen Bereichen des Forums.

2.3 Bekannte Probleme

Hier ist eine Liste von Punkten, die xlsx.hwp noch nicht unterstützt oder die auf irgendeine Weise verwirrend sein könnten:

- Ist noch offen (tpd)

2.4 Zukunft

Hier sind einige Punkte, die auf meiner Aufgabenliste stehen:

- Unterstützung für die 68k-Plattform hinzufügen (derzeit gibt es keinen C++17-Compiler für Amiga 68k, daher ist es derzeit nicht möglich, die Plattform zu unterstützen)
- Unterstützung für Zellformatierung hinzufügen
- Unterstützung für das Einbetten von Bildern hinzufügen

Zögern Sie nicht, mich zu kontaktieren, wenn xlsx.hwp eine bestimmte Funktion fehlt, die für Ihr Projekt wichtig ist.

2.5 Geschichte

Bitte schauen Sie in die auf englisch verfasste Datei `history.txt`. Hier finden Sie ein vollständiges Änderungsprotokoll von Polybios.

3 Anwendung

3.1 Schnittstellenübersicht

Es gibt zwei Möglichkeiten, dieses Plugin zu verwenden: Entweder über die Bibliotheks- oder über die Serialisierungsschnittstelle. Die Verwendung des Plugins über die Serialisierungsschnittstelle ist einfacher und sehr bequem, geht aber auf Kosten der Flexibilität. Die Verwendung des Plugins über die Bibliotheksschnittstelle ist etwas schwieriger, bietet aber volle Flexibilität. In den nächsten beiden Kapiteln finden Sie einen kurzen Überblick über die beiden unterschiedlichen Schnittstellen.

3.2 Bibliotheksschnittstelle

Die typische Art, dieses Plugin zu benutzen, ist der Umgang mit XLSX-Dokumenten über die Bibliotheksschnittstelle des Plugins. Die Bibliotheksschnittstelle besteht aus einer Vielzahl von Befehlen, mit denen Sie XLSX-Dokumente öffnen und speichern, Zellwerte und andere Dokument- und Arbeitsblatteigenschaften festlegen und abrufen können. Hier ist zum Beispiel ein Skript, das ein XLSX-Dokument mit 100 Zeilen und 30 Spalten erstellt. Die Zellenwerte werden auf eine Textzeichenkette gesetzt, die die Spalten- und Zeilennummer jeder Zelle enthält, und das XLSX-Dokument wird als `test.xlsx` gespeichert.

```
@REQUIRE "xlsx"
xlsx.Create(1, "test.xlsx")
For Local y = 1 To 100
   For Local x = 1 to 30
      xlsx.SetCellValue(1, x, y, "Cell " .. x .. "/" .. y)
   Next
Next
xlsx.Save(1)
xlsx.Close(1)
```

Alternativ können Sie auch die Serialisierungsschnittstelle des Plugins verwenden. Dies ist einfacher, da nur ein einziger Befehlsaufruf erforderlich ist, um Hollywood-Tabellen in XLSX-Dokumente und umgekehrt zu konvertieren, aber Sie haben keine fein abgestimmte Kontrolle über alles, wie Sie es bei der Verwendung der Bibliotheksschnittstelle haben.

Im nächsten Kapitel finden Sie weitere Details zur Serialisierungsschnittstelle des Plugins.

3.3 Serialisierungsschnittstelle

Wenn Sie die Bibliotheksschnittstelle von xlsx.hwp (siehe oben) aus irgendeinem Grund nicht verwenden möchten, können Sie auch die Serialisierungsschnittstelle des Plugins benutzen. Diese ist einfacher zu verwenden, da nur ein einziger Befehlsaufruf erforderlich ist, um Hollywood-Tabellen in XLSX-Dokumente und umgekehrt zu konvertieren, aber Sie haben nicht die fein abgestimmte Kontrolle über alles, wie Sie es bei der Verwendung der Bibliotheksschnittstelle haben.

Der Zugriff auf die Serialisierungsschnittstelle von xlsx.hwp erfolgt über den Befehl `SerializeTable()` und `DeserializeTable()` von Hollywood oder alternativ über die Befehle `ReadTable()` und `WriteTable()`. Wenn Sie die Serialisierungsschnittstelle

verwenden, können Sie ein XLSX-Dokument mit einem einzigen Befehlsaufruf in eine Hollywood-Tabelle umwandeln:

```
t = DeserializeTable(FileToString("test.xlsx"), "xlsx")
```

Der obige Code liest alle Zeilen und Spalten aus `test.xlsx` und speichert sie in der Hollywood-Tabelle `t`. Sie könnten dann alle Zeilen und Spalten in dieser Tabelle wie folgt ausgeben:

```
For Local y = 0 To ListItems(t) - 1
   For Local x = 0 To ListItems(t[y]) - 1
      DebugPrint(t[y][x])
   Next
Next
```

Sie könnten dann einfach die Zellenwerte ändern, indem Sie neue Werte in die Tabelle `t` schreiben. Der folgende Code ändert beispielsweise den Wert der Zelle in der 5. Spalte und der 10. Zeile in "Hallo":

```
t[9][4] = "Hello"
```

Wenn Sie mit allen Änderungen fertig sind, können Sie Ihre Hollywood-Tabelle einfach wieder in ein XLSX-Dokument umwandeln, und zwar in einer einzigen Zeile:

```
StringToFile(SerializeTable(t, "xlsx"), "test2.xlsx")
```

Der obige Code konvertiert die Tabelle `t` mithilfe des xlsx.hwp-Plugins in ein XLSX-Dokument und speichert das XLSX-Dokument als `test2.xlsx`.

Wie Sie sehen können, ist die Serialisierungsschnittstelle sehr einfach zu verwenden, bietet jedoch nicht so viel Flexibilität wie die Bibliotheksschnittstelle, die Ihnen eine fein abgestimmte Kontrolle über viele Funktionen von XLSX-Dokumenten bietet.

4 Befehlsreferenz

4.1 xlsx.AddSheet

BEZEICHNUNG

xlsx.AddSheet – fügt ein neues Arbeitsblatt hinzu

ÜBERSICHT

```
xlsx.AddSheet(id, name$[, pos])
```

BESCHREIBUNG

Dieser Befehl fügt dem durch `id` angegebenen XLSX-Dokument ein neues Arbeitsblatt hinzu. Das Arbeitsblatt erhält den durch `name$` angegebenen Namen. Mit dem optionalen Argument `pos` können Sie angeben, wo das Arbeitsblatt im XLSX-Dokument eingefügt werden soll (Arbeitsblattpositionen beginnen bei 1). Wenn `pos` weggelassen oder auf eine Position außerhalb des zulässigen Bereichs gesetzt wird, wird das neue Arbeitsblatt als letztes Arbeitsblatt hinzugefügt.

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments
`name$`	Name für das neue Arbeitsblatt
`pos`	optional: gewünschte Einfügeposition, beginnend bei 1 für das erste Arbeitsblatt (voreingestellt ist 0, was bedeutet, dass es als letztes Arbeitsblatt eingefügt wird)

4.2 xlsx.CellRange

BEZEICHNUNG

xlsx.CellRange – durchläuft einen Zellbereich

ÜBERSICHT

```
ref = xlsx.CellRange(id, startx, starty, endx, endy[, sheet])
ref = xlsx.CellRange(id, startcell$, endcell$[, sheet])
```

BESCHREIBUNG

Dieser Befehl kann verwendet werden, um einen Bereich von Zellen zu durchlaufen. Sie müssen die Zelle übergeben, in dem der Durchlauf beginnen soll, und die Zelle, in der sie enden soll. `xlsx.CellRange()` gibt dann eine Iteratorfunktion zurück, die zusammen mit Hollywoods generischer For-Schleife verwendet werden kann. Die Iteratorfunktion gibt eine Referenz auf eine Zelle zurück, die an alle Befehle übergeben werden kann, die mit Zellen wie `xlsx.SetCellValue()` oder `xlsx.GetCellValue()` arbeiten.

Die Übergabe einer Zellreferenz, die von `xlsx.CellRange()` zurückgegeben wird, an Befehlen wie `xlsx.SetCellValue()` oder `xlsx.GetCellValue()` ist viel schneller als das Adressieren der Zelle über ihre Spalten- und Zeilenposition oder ihre alphanumerische Position (z.B. `"A1"`). Aus diesem Grund wird empfohlen, `xlsx.CellRange()` immer dann zu verwenden, wenn Sie viele Zellen durchlaufen müssen, insbesondere in großen XLSX-Dokumenten mit Tausenden von Spalten und Zeilen.

`xlsx.CellRange()` unterstützt zwei Möglichkeiten, den Start und die Zellen anzugeben: Sie können entweder die zu verwendenden Zellen angeben, indem Sie ihre Spalten- (x) und Zeilenpositionen (y) in den Argumenten `startx/starty` und `endx/endy` übergeben. Diese Positionen beginnen bei 1 für die erste Spalte und Zeile. Alternativ können Sie die Zellen auch angeben, indem Sie ihre alphanumerischen Position in den Parametern `startcell$` und `endcell$` übergeben, z.B. `"A10"` für die erste Zelle in der 10. Zeile. Optional können Sie auch den Index des zu verwendenden Arbeitsblatts im optionalen Parameter `sheet` übergeben (beginnend mit 1 für das erste Arbeitsblatt). Wenn der Parameter `sheet` weggelassen wird, wird das mit `xlsx.SetDefaultSheet()` gesetzte Arbeitsblatt verwendet.

EINGABEN

`id` ID des zu verwendenden XLSX-Dokuments

`startx` Spaltenindex der Startzelle

`starty` Zeilenindex der Startzelle

`endx` Spaltenindex der Endzelle

`endy` Zeilenindex der Endzelle

`startcell$`
alphanumerischer Startzellenposition (z.B. `"A1"`), wird nur verwendet, wenn `startx` und `starty` weggelassen werden

`endcell$` alphanumerischer Zellenendposition (z.B. `"Z100"`), wird nur verwendet, wenn `endx` und `endy` weggelassen werden

`sheet` optional: Index des zu verwendenden Arbeitsblatts (voreingestellt ist der Index des Standardarbeitsblatts)

RÜCKGABEWERTE

`ref` eine Zellenposition

BEISPIEL

```
xlsx.Open(1, "test.xlsx")
cols = xlsx.GetColumnCount(1)
rows = xlsx.GetRowCount(1)
For ref In xlsx.CellRange(1, 1, 1, cols, rows)
   DebugPrint((xlsx.GetCellValue(1, ref)))
Next
xlsx.Close(1)
```

Der obige Code öffnet die Datei `test.xlsx` und gibt die Werte aller Zellen aus.

4.3 xlsx.ClearCellFormula

BEZEICHNUNG

xlsx.ClearCellFormula – löscht die Zellformel

ÜBERSICHT

```
xlsx.ClearCellFormula(id, x, y, f$[, sheet])
xlsx.ClearCellFormula(id, ref, f$[, sheet])
```

BESCHREIBUNG

Dieser Befehl löscht die Formel der angegebenen Zelle. Nach Aufruf von diesem Befehl gibt `xlsx.HaveCellFormula()` FLASE zurück. Es gibt zwei Möglichkeiten, die Zelle anzugeben, deren Formel gelöscht werden soll: Sie können entweder die zu verwendende Zelle angeben, indem Sie die Spalten- (x) und Zeilenposition (y) der Zelle in den Argumenten `x` und `y` übergeben. Diese Positionen beginnen bei 1 für die erste Spalte und Zeile. Alternativ können Sie die Zelle auch angeben, indem Sie ihre Referenz im Parameter `ref` übergeben. Dies kann entweder eine Zeichenkette sein, z.B. `"A10"` für die erste Zelle in der 10. Zeile oder eine Iteratorreferenz, die vom Befehl der `xlsx.CellRange()` zurückgegeben wird. Optional können Sie auch den Index des zu verwendenden Arbeitsblatts im optionalen `sheet-Parameter` übergeben (beginnend mit 1 für das erste Arbeitsblatt). Wenn der Parameter `sheet` weggelassen wird, wird das mit `xlsx.SetDefaultSheet()` eingestellte Arbeitsblatt verwendet.

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments
`x`	Spaltenindex der zu verwendenden Zelle (beginnend bei 1)
`y`	Zeilenindex der zu verwendenden Zelle (beginnend bei 1)
`ref`	Zellreferenz (z.B. `"A1"` oder ein Iteratorreferenz), wird nur verwendet, wenn `x` und `y` weggelassen werden
`sheet`	optional: Index des zu verwendenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

4.4 xlsx.ClearCellValue

BEZEICHNUNG

xlsx.ClearCellValue – löscht den Zellwert

ÜBERSICHT

```
xlsx.ClearCellValue(id, x, y[, sheet])
xlsx.ClearCellValue(id, ref[, sheet])
```

BESCHREIBUNG

Dieser Befehl löscht den Wert der angegebenen Zelle. Es gibt zwei Möglichkeiten, die Zelle anzugeben, deren Wert gelöscht werden soll: Sie können entweder die zu verwendende Zelle angeben, indem Sie die Spalten- (x) und Zeilenposition (y) der Zelle in den Argumenten `x` und `y` übergeben. Diese Positionen beginnen bei 1 für die erste Spalte und Zeile. Alternativ können Sie die Zelle auch angeben, indem Sie ihre Referenz im Parameter `ref` übergeben. Dies kann entweder eine Zeichenkette sein, z.B. `"A10"` für die erste Zelle in der 10. Zeile oder eine Iteratorreferenz, die vom Befehl der `xlsx.CellRange()` zurückgegeben wird. Optional können Sie auch den Index des zu verwendenden Arbeitsblatts im optionalen `sheet-Parameter` übergeben (beginnend mit 1 für das erste Arbeitsblatt). Wenn der

Parameter `sheet` weggelassen wird, wird das mit `xlsx.SetDefaultSheet()` eingestellte Arbeitsblatt verwendet.

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments
`x`	Spaltenindex der zu verwendenden Zelle (beginnend bei 1)
`y`	Zeilenindex der zu verwendenden Zelle (beginnend bei 1)
`ref`	Zellreferenz (z.B. `"A1"` oder ein Iteratorreferenz), wird nur verwendet, wenn `x` und `y` weggelassen werden
`sheet`	optional: Index des zu verwendenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

4.5 xlsx.Close

BEZEICHNUNG

xlsx.Close – schliesst ein XLSX-Dokument

ÜBERSICHT

```
xlsx.Close(id)
```

BESCHREIBUNG

Dieser Befehl schließt das angegebene XLSX-Dokument, das entweder durch `xlsx.Open()` oder `xlsx.Create()` erstellt wurde. Beachten Sie, dass dieser Befehl keine Änderungen speichert, die Sie am XLSX-Dokument vorgenommen haben. Wenn Sie möchten, dass Änderungen im XLSX-Dokument gespeichert werden, müssen Sie zuerst `xlsx.Save()` oder `xlsx.SaveAs()` aufrufen.

EINGABEN

`id`	ID des XLSX-Dokuments, welches geschlossen wird

4.6 xlsx.Create

BEZEICHNUNG

xlsx.Create – erstellt ein leeres XLSX-Dokument

ÜBERSICHT

```
[id] = xlsx.Create(id, filename$)
```

BESCHREIBUNG

Dieser Befehl erstellt ein leeres XLSX-Dokument, das ein einzelnes Arbeitsblatt namens `"Sheet1"` enthält. Beachten Sie, dass das XLSX-Dokument nicht in `filename$` gespeichert wird, bis Sie entweder `xlsx.Save()` oder `xlsx.SaveAs()` darauf aufrufen.

EINGABEN

`id`	ID des XLSX-Dokuments oder `Nil` für die automatische Id-Auswahl

`filename$`
: gewünschter Pfad und Dateiname für das neue Dokument

RÜCKGABEWERTE

`id`
: optional: ID des Dokuments; wird nur zurückgegeben, wenn Sie `Nil` als Argument 1 übergeben (siehe oben)

BEISPIEL

```
xlsx.Create(1, "test.xlsx")
For Local y = 1 To 100
   For Local x = 1 to 30
      xlsx.SetCellValue(1, x, y, "Cell " .. x .. "/" .. y)
   Next
Next
xlsx.Save(1)
xlsx.Close(1)
```

Der obige Code erstellt ein neues XLSX-Dokument und fügt ihm 30 Spalten und 100 Zeilen hinzu. Das Dokument wird als `test.xlsx` gespeichert.

4.7 xlsx.DeleteProperty

BEZEICHNUNG

xlsx.DeleteProperty – löscht eine Dokumenteigenschaft

ÜBERSICHT

```
xlsx.DeleteProperty(id, prop)
```

BESCHREIBUNG

Mit diesem Befehl können Sie die durch `prop` angegebene Dokumenteigenschaft löschen. Der Parameter `prop` muss eine der folgenden speziellen Konstanten sein:

```
#XLSX_PROPERTY_TITLE
#XLSX_PROPERTY_SUBJECT
#XLSX_PROPERTY_CREATOR
#XLSX_PROPERTY_KEYWORDS
#XLSX_PROPERTY_DESCRIPTION
#XLSX_PROPERTY_LASTMODIFIEDBY
#XLSX_PROPERTY_LASTPRINTED
#XLSX_PROPERTY_CREATIONDATE
#XLSX_PROPERTY_MODIFICATIONDATE
#XLSX_PROPERTY_CATEGORY
#XLSX_PROPERTY_APPLICATION
#XLSX_PROPERTY_DOCSECURITY
#XLSX_PROPERTY_SCALECROP
#XLSX_PROPERTY_MANAGER
#XLSX_PROPERTY_COMPANY
#XLSX_PROPERTY_LINKSUPTODATE
#XLSX_PROPERTY_SHAREDDOC
```

```
#XLSX_PROPERTY_HYPERLINKBASE
#XLSX_PROPERTY_HYPERLINKSCHANGED
#XLSX_PROPERTY_APPVERSION
```

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments
`prop`	zu löschende Eigenschaft (siehe oben für mögliche Werte)

4.8 xlsx.DeleteSheet

BEZEICHNUNG

xlsx.DeleteSheet – löscht ein Arbeitsblatt

ÜBERSICHT

```
xlsx.DeleteSheet(id, idx)
```

BESCHREIBUNG

Mit diesem Befehl kann das Arbeitsblatt an der durch `idx` angegebenen Position aus dem durch `id` angegebenen XLSX-Dokument gelöscht werden. Arbeitsblattpositionen werden von 1 an gezählt. Beachten Sie, dass Sie nicht alle Arbeitsblätter aus einem XLSX-Dokument löschen können; es muss mindestens ein Arbeitsblatt im XLSX-Dokument vorhanden sein.

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments
`idx`	Index des zu löschenden Arbeitsblatts (das erste Arbeitsblatt hat den Index 1)

4.9 xlsx.GetCellFormula

BEZEICHNUNG

xlsx.GetCellFormula – ermittelt die Zellformel

ÜBERSICHT

```
f$ = xlsx.GetCellFormula(id, x, y[, sheet])
f$ = xlsx.GetCellFormula(id, ref[, sheet])
```

BESCHREIBUNG

Dieser Befehl gibt die Formel einer bestimmten Zelle zurück. Es gibt zwei Möglichkeiten, die Zelle anzugeben, deren Formel zurückgegeben werden soll: Sie können entweder die zu verwendende Zelle angeben, indem Sie die Spalten- (x) und Zeilenposition (y) der Zelle in den Argumenten `x` und `y` übergeben. Diese Positionen beginnen bei 1 für die erste Spalte und Zeile. Alternativ können Sie die Zelle auch angeben, indem Sie ihre Referenz im Parameter `ref` übergeben. Dies kann entweder eine Zeichenkette sein, z.B. `"A10"` für die erste Zelle in der 10. Zeile oder eine Iteratorreferenz, die vom Befehl `xlsx.CellRange()` zurückgegeben wird. Optional können Sie auch den Index des zu verwendenden Arbeitsblatts im optionalen Parameter `sheet` übergeben (beginnend mit

1 für das erste Arbeitsblatt). Wenn der Parameter `sheet` weggelassen wird, wird das mit `xlsx.SetDefaultSheet()` eingestellte Arbeitsblatt verwendet.

Beachten Sie, dass dieser Befehl fehlschlägt, falls die Zelle keine Formel enthält. Sie können den Befehl `xlsx.HaveCellFormula()` verwenden, um zu prüfen, ob die Zelle eine Formel enthält.

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments
`x`	Spaltenindex der zu verwendenden Zelle (beginnend bei 1)
`y`	Zeilenindex der zu verwendenden Zelle (beginnend bei 1)
`ref`	Zellreferenz (z.B. `"A1"` oder ein Iteratorreferenz), wird nur verwendet, wenn `x` und `y` weggelassen werden
`sheet`	optional: Index des zu verwendenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

RÜCKGABEWERTE

`f$`	die Formel der Zelle

4.10 xlsx.GetCellReference

BEZEICHNUNG

xlsx.GetCellReference – gibt die Zellposition/Zelladresse zurück

ÜBERSICHT

```
ref = xlsx.GetCellReference(id, x, y[, xyref, sheet])
ref = xlsx.GetCellReference(id, ref[, xyref, sheet])
```

BESCHREIBUNG

Dieser Befehl gibt einen Verweis auf die angegebene Zelle zurück, entweder als Spalten-/Zeilenposition oder als alphanumerische Zellen-ID. Wenn der Parameter `xyref` auf `True` gesetzt ist, wird die Zellreferenz als Paar von Spalten- (x) und Zeilenkoordinaten (y) an die Zelle zurückgegeben. Wenn `xyref` auf `False` gesetzt ist (Standardeinstellung), wird die Zellposition/Zelladresse als alphanumerische Zeichenkette zurückgegeben, der die Spalten- und Zeilen-ID der Zelle enthält, z.B. `"A1"`.

Es gibt zwei Möglichkeiten, die Zelle anzugeben, deren Position zurückgegeben werden soll: Sie können entweder die zu verwendende Zelle angeben, indem Sie die Spalten- (x) und Zeilenposition (y) der Zelle in den Argumenten `x` und `y` übergeben. Diese Positionen beginnen bei 1 für die erste Spalte und Zeile. Alternativ können Sie die Zelle auch angeben, indem Sie ihre Position im Parameter `ref` übergeben. Dies kann entweder eine Zeichenkette sein, z.B. `"A10"` für die erste Zelle in der 10. Zeile oder eine Iteratorreferenz, der von dem Befehl `xlsx.CellRange()` zurückgegeben wird. Optional können Sie auch den Index des zu verwendenden Arbeitsblatts im optionalen Parameter `sheet` übergeben (beginnend mit 1 für das erste Arbeitsblatt). Wenn der Parameter `sheet` weggelassen wird, wird das mit `xlsx.SetDefaultSheet()` eingestellte Arbeitsblatt verwendet.

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments

`x` Spaltenindex der zu verwendenden Zelle (beginnend bei 1)

`y` Zeilenindex der zu verwendenden Zelle (beginnend bei 1)

`ref` Zellposition (z.B. `"A1"` oder eine Iteratorreferenz), wird nur verwendet, wenn `x` und `y` weggelassen werden

`xyref` `True`, wenn Sie die Position als Paar von Spalten-/Zeilenkoordinaten wünschen, oder `False`, wenn Sie den Verweis als alphanumerische Zeichenkette wünschen (Standardwert: `False`)

`sheet` optional: Index des zu verwendenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

RÜCKGABEWERTE

`ref` die Zellposition/Zelladresse

4.11 xlsx.GetCellValue

BEZEICHNUNG

xlsx.GetCellValue – gibt den Wert der Zelle zurück

ÜBERSICHT

```
v, t = xlsx.GetCellValue(id, x, y[, sheet])
v, t = xlsx.GetCellValue(id, ref[, sheet])
```

BESCHREIBUNG

Dieser Befehl gibt den Wert einer bestimmten Zelle zurück. Es gibt zwei Möglichkeiten, die Zelle anzugeben, deren Wert zurückgegeben werden soll: Sie können entweder die zu verwendende Zelle angeben, indem Sie die Spalten- (x) und Zeilenposition (y) der Zelle in den Argumenten `x` und `y` übergeben. Diese Positionen beginnen bei 1 für die erste Spalte und Zeile. Alternativ können Sie die Zelle auch angeben, indem Sie ihre Position im Parameter `ref` übergeben. Dies kann entweder eine Zeichenkette sein, z.B. `"A10"` für die erste Zelle in der 10. Zeile oder eine Iteratorreferenz, die vom Befehl `xlsx.CellRange()` zurückgegeben wird. Optional können Sie auch den Index des zu verwendenden Arbeitsblatts im optionalen Parameter `sheet` übergeben (beginnend mit 1 für das erste Arbeitsblatt). Wenn der Parameter `sheet` weggelassen wird, wird das vom Befehl `xlsx.SetDefaultSheet()` eingestellte Arbeitsblatt verwendet.

`xlsx.GetCellValue()` gibt zwei Werte zurück: Den eigentlichen Zellenwert im ersten Rückgabewert `v` und den Zellenwerttyp im zweiten Rückgabewert `t`. Der Rückgabewerttyp ist eine der folgenden speziellen Konstanten:

`#INTEGER` Eine Ganzzahl.

`#DOUBLE` Ein Fließkommawert.

`#STRING` Ein Zeichenkettenwert.

`#BOOLEAN` Ein boolescher Wert (entweder `True` oder `False`).

`#NIL` Die Zelle ist leer.

`#VOID` Zeigt einen ungültigen Wert an, z.B. `NaN` oder einen logischen Fehler wie Division durch Null.

Beachten Sie, dass es beim Versuch, die Werte vieler Zellen zu erhalten, normalerweise viel schneller ist, den Befehl `xlsx.CellRange()` zusammen mit einer generischen For-Schleife zu verwenden, um die gewünschten Zellen zu durchlaufen. Dies wird besonders empfohlen, wenn es sich um große XLSX-Dokumente mit Tausenden von Zellen handelt.

EINGABEN

`id` ID des zu verwendenden XLSX-Dokuments

`x` Spaltenindex der zu verwendenden Zelle (beginnend bei 1)

`y` Zeilenindex der zu verwendenden Zelle (beginnend bei 1)

`ref` Zellposition (z.B. `"A1"` oder eine Iteratorreferenz), wird nur verwendet, wenn `x` und `y` weggelassen werden

`sheet` optional: Index des zu verwendenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

RÜCKGABEWERTE

`v` Zellenwert

`t` Typ des Zellwerts (siehe oben für mögliche Typen)

BEISPIEL

```
xlsx.Open(1, "test.xlsx")
cols = xlsx.GetColumnCount(1)
rows = xlsx.GetRowCount(1)
For Local y = 1 To rows
   For Local x = 1 to cols
      DebugPrint((xlsx.GetCellValue(1, x, y)))
   Next
   DebugPrint("************************")
Next
xlsx.Close(1)
```

Der obige Code öffnet die Datei `test.xlsx` und gibt die Werte aller Zellen aus.

4.12 xlsx.GetColumnCount

BEZEICHNUNG

xlsx.GetColumnCount – gibt die Anzahl der Arbeitsblattspalten zurück

ÜBERSICHT

```
cols = xlsx.GetColumnCount(id[, idx])
```

BESCHREIBUNG

Dieser Befehl gibt die Anzahl der Spalten im Arbeitsblatt zurück, das sich an dem durch `idx` angegebenen Index im XLSX-Dokument befindet. Wenn das Argument `idx` weggelassen wird, wird das mit dem Befehl `xlsx.SetDefaultSheet()` eingestellte Standardarbeitsblatt verwendet. Arbeitsblattindizes beginnen bei 1 für das erste Arbeitsblatt.

EINGABEN

`id` ID des zu verwendenden XLSX-Dokuments

`idx` optional: Index des abzufragenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

RÜCKGABEWERTE

`cols` Anzahl der Spalten im angegebenen Arbeitsblatt

4.13 xlsx.GetColumnWidth

BEZEICHNUNG

xlsx.GetColumnWidth – gibt die Spaltenbreite zurück

ÜBERSICHT

```
width = xlsx.GetColumnWidth(id, col[, sheet])
```

BESCHREIBUNG

Dieser Befehl gibt die Breite der in `col` angegebenen Spalte zurück. Die Spaltenindizes beginnen bei 1. Die Breite wird in Schriftarteinheiten der normalen Display-Schriftart zurückgegeben und kann ein Bruchwert sein. Optional können Sie auch den Index des zu verwendenden Arbeitsblatts im optionalen Parameter `sheet` übergeben (beginnend mit 1 für das erste Arbeitsblatt). Wenn der Parameter `sheet` weggelassen wird, wird das mit `xlsx.SetDefaultSheet()` eingestellte Arbeitsblatt verwendet.

EINGABEN

`id` ID des zu verwendenden XLSX-Dokuments

`col` zu verwendender Spaltenindex (beginnend bei 1)

`sheet` optional: Index des zu verwendenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

RÜCKGABEWERTE

`width` Spaltenbreite in Schriftarteneinheiten

4.14 xlsx.GetObjectType

BEZEICHNUNG

xlsx.GetObjectType – gibt den XLSX-Dokumentobjekttyp zurück

ÜBERSICHT

```
type = xlsx.GetObjectType()
```

BESCHREIBUNG

Dieser Befehl gibt den Objekttyp zurück, der von XLSX-Dokumenten verwendet wird, die mit den Befehlen `xlsx.Open()` oder `xlsx.Create()` geöffnet wurden. Sie können diesen Objekttyp dann mit Befehlen aus Hollywoods Objektbibliothek wie `GetAttribute()`, `SetObjectData()`, `GetObjectData()` usw. verwenden.

Insbesondere der Befehl `GetAttribute()` von Hollywood kann verwendet werden, um bestimmte Eigenschaften von XLSX-Dokumenten abzufragen. Die folgenden Attribute werden derzeit von `GetAttribute()` für XLSX-Dokumente unterstützt:

`#XLSXATTRSHEETS:`
: Gibt die Anzahl der Blätter im XLSX-Dokument zurück.

EINGABEN

keine

RÜCKGABEWERTE

`type` interner XLSX-Dokumenttyp zur Verwendung mit Hollywoods Objektbibliothek

BEISPIEL

```
xlsx.Open(1, "test.xlsx")
XLSX_DOCUMENT = xlsx.GetObjectType()
numsheets = GetAttribute(XLSX_DOCUMENT, 1, #XLSXATTRSHEETS)
```

Der obige Code öffnet die Datei `test.xlsx` und fragt die Anzahl der Blätter im Dokument über `GetAttribute()` ab.

4.15 xlsx.GetProperty

BEZEICHNUNG

xlsx.GetProperty – gibt die Dokumenteneigenschaft zurück

ÜBERSICHT

```
val$ = xlsx.GetProperty(id, prop)
```

BESCHREIBUNG

Mit diesem Befehl können Sie den Wert der durch `prop` angegebenen Dokumenteigenschaft abrufen. Der Parameter `prop` muss eine der folgenden speziellen Konstanten sein:

```
#XLSX_PROPERTY_TITLE
#XLSX_PROPERTY_SUBJECT
#XLSX_PROPERTY_CREATOR
#XLSX_PROPERTY_KEYWORDS
#XLSX_PROPERTY_DESCRIPTION
#XLSX_PROPERTY_LASTMODIFIEDBY
#XLSX_PROPERTY_LASTPRINTED
#XLSX_PROPERTY_CREATIONDATE
#XLSX_PROPERTY_MODIFICATIONDATE
#XLSX_PROPERTY_CATEGORY
#XLSX_PROPERTY_APPLICATION
#XLSX_PROPERTY_DOCSECURITY
#XLSX_PROPERTY_SCALECROP
#XLSX_PROPERTY_MANAGER
#XLSX_PROPERTY_COMPANY
#XLSX_PROPERTY_LINKSUPTODATE
```

```
#XLSX_PROPERTY_SHAREDDOC
#XLSX_PROPERTY_HYPERLINKBASE
#XLSX_PROPERTY_HYPERLINKSCHANGED
#XLSX_PROPERTY_APPVERSION
```

EINGABEN

`id` ID des zu verwendenden XLSX-Dokuments

`prop` die zu ermittelnde Eigenschaft (siehe oben für mögliche Werte)

RÜCKGABEWERTE

`val$` Wert von der Eigenschaft

4.16 xlsx.GetRowCount

BEZEICHNUNG

xlsx.GetRowCount – gibt die Anzahl der Arbeitsblattzeilen zurück

ÜBERSICHT

```
rows = xlsx.GetRowCount(id[, idx])
```

BESCHREIBUNG

Dieser Befehl gibt die Anzahl der Zeilen im Arbeitsblatt zurück, das sich an dem durch `idx` angegebenen Index im XLSX-Dokument befindet. Wenn das Argument `idx` weggelassen wird, wird das Standardarbeitsblatt vom Befehl `xlsx.SetDefaultSheet()` verwendet. Arbeitsblattindizes beginnen bei 1 für das erste Arbeitsblatt.

EINGABEN

`id` ID des zu verwendenden XLSX-Dokuments

`idx` optional: Index des abzufragenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

RÜCKGABEWERTE

`rows` Anzahl der Zeilen im angegebenen Arbeitsblatt

4.17 xlsx.GetRowHeight

BEZEICHNUNG

xlsx.GetRowHeight – gibt die Zeilenhöhe zurück

ÜBERSICHT

```
height = xlsx.GetRowHeight(id, row[, sheet])
```

BESCHREIBUNG

Dieser Befehl gibt die Höhe der in `row` angegebenen Zeile zurück. Die Zeilenindizes beginnen bei 1. Die Höhe wird in Schriftarteinheiten der normalen Display-Schriftart zurückgegeben und kann ein Bruchwert sein. Optional können Sie auch den Index des zu verwendenden Arbeitsblatts im optionalen Parameter `sheet` übergeben (beginnend mit 1 für das erste Arbeitsblatt). Wenn der Parameter `sheet` weggelassen wird, wird das mit `xlsx.SetDefaultSheet()` eingestellte Arbeitsblatt verwendet.

EINGABEN

`id` ID des zu verwendenden XLSX-Dokuments

`row` zu verwendender Zeilenindex (beginnend bei 1)

`sheet` optional: Index des zu verwendenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

RÜCKGABEWERTE

`height` Zeilenhöhe in Schriftarteneinheiten

4.18 xlsx.GetSheetCount

BEZEICHNUNG

xlsx.GetSheetCount – gibt die Anzahl der Arbeitsblätter im Dokument zurück

ÜBERSICHT

```
n = xlsx.GetSheetCount(id)
```

BESCHREIBUNG

Dieser Befehl gibt die Anzahl der Arbeitsblätter im dem durch `id` angegebenen XLSX-Dokument zurück. Da es keine XLSX-Dokumente ohne Arbeitsblätter geben kann, ist der Rückgabewert immer mindestens 1.

EINGABEN

`id` ID des zu verwendenden XLSX-Dokuments

RÜCKGABEWERTE

`n` Anzahl der Arbeitsblätter im XLSX-Dokument

4.19 xlsx.GetSheetIndex

BEZEICHNUNG

xlsx.GetSheetIndex – gibt den Arbeitsblattindex zurück

ÜBERSICHT

```
idx = xlsx.GetSheetIndex(id, name$)
```

BESCHREIBUNG

Dieser Befehl gibt die Position des durch `name$` angegebenen Arbeitsblatts in dem durch `id` angegebenen XLSX-Dokument zurück. Arbeitsblattindizes beginnen bei 1. Wenn das Arbeitsblatt im XLSX-Dokument nicht gefunden werden kann, wird 0 zurückgegeben.

EINGABEN

`id` ID des zu verwendenden XLSX-Dokuments

`name$` Name des Arbeitsblatts, dessen Position abgerufen werden soll

RÜCKGABEWERTE

`idx` Position des Arbeitsblatts oder 0, wenn es nicht gefunden wurde

4.20 xlsx.GetSheetName

BEZEICHNUNG

xlsx.GetSheetName – gibt den Arbeitsblattname zurück

ÜBERSICHT

```
name$ = xlsx.GetSheetName(id, idx)
```

BESCHREIBUNG

Dieser Befehl gibt den Namen des Arbeitsblatts an der durch `idx` angegebenen Position zurück. Arbeitsblattindizes beginnen bei 1.

EINGABEN

`id` ID des zu verwendenden XLSX-Dokuments

`idx` Position des Arbeitsblatts, dessen Name abgerufen werden soll

RÜCKGABEWERTE

`name$` Name des Arbeitsblatts an der angegebenen Position

4.21 xlsx.GetSheetType

BEZEICHNUNG

xlsx.GetSheetType – gibt den Arbeitsblatttyp zurück

ÜBERSICHT

```
type = xlsx.GetSheetType(id, idx)
```

BESCHREIBUNG

Dieser Befehl gibt den Typ des Arbeitsblatts an der durch `idx` angegebenen Position zurück. Arbeitsblattindizes beginnen bei 1. Der Rückgabewert ist eine der folgenden Konstanten:

`#XLSX_SHEETTYPE_WORKSHEET`
: Ein normales Arbeitsblatt.

`#XLSX_SHEETTYPE_CHARTSHEET`
: Ein Diagrammarbeitsblatt.

`#XLSX_SHEETTYPE_DIALOGSHEET`
: Ein Dialogarbeitsblatt.

`#XLSX_SHEETTYPE_MACROSHEET`
: Ein Makro-Arbeitsblatt.

EINGABEN

`id` ID des zu verwendenden XLSX-Dokuments

`idx` Position des Arbeitsblatts, dessen Typ abgerufen werden soll (beginnend bei 1)

RÜCKGABEWERTE

`type` Typ des Arbeitsblatts an der angegebenen Position

4.22 xlsx.GetSheetVisibility

BEZEICHNUNG

xlsx.GetSheetVisibility – gibt den Arbeitsblatt-Sichtbarkeitstatus zurück

ÜBERSICHT

```
vis = xlsx.GetSheetVisibility(id[, sheet])
```

BESCHREIBUNG

Dieser Befehl kann verwendet werden, um den Sichtbarkeitsstatus des Arbeitsblatts abzurufen, das durch den Parameter `sheet` angegeben ist. Der Rückgabewert ist eine der folgenden speziellen Konstanten:

`#XLSX_VISIBILITY_VISIBLE`
: Das Arbeitsblatt ist sichtbar.

`#XLSX_VISIBILITY_HIDDEN`
: Das Blatt ist ausgeblendet, kann aber von Benutzern eingeblendet werden, die die XLSX-Datei in einer Tabellenkalkulations-App öffnen.

`#XLSX_VISIBILITY_VERYHIDDEN`
: Das Blatt ist ausgeblendet und kann von Benutzern, die die XLSX-Datei in einer Tabellenkalkulations-App öffnen, nicht eingeblendet werden.

Der Parameter `sheet` ist optional. Wenn er weggelassen wird, wird das mit `xlsx.SetDefaultSheet()` eingestellte Arbeitsblatt verwendet. Die Blattindizes beginnen bei 1 für das erste Arbeitsblatt.

EINGABEN

`id` ID des zu verwendenden XLSX-Dokuments

`sheet` optional: Index des zu verwendenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

RÜCKGABEWERTE

`vis` Arbeitsblatt-Sichtbarkeitstatus (siehe oben für mögliche Werte)

4.23 xlsx.HaveCellFormula

BEZEICHNUNG

xlsx.HaveCellFormula – überprüft, ob die Zelle eine Formel enthält

ÜBERSICHT

```
bool = xlsx.HaveCellFormula(id, x, y[, sheet])
bool = xlsx.HaveCellFormula(id, ref[, sheet])
```

BESCHREIBUNG

Dieser Befehl gibt `True` zurück, wenn die angegebene Zelle eine Formel enthält, andernfalls wird `False` zurückgegeben. Es gibt zwei Möglichkeiten, die Zelle anzugeben, die Sie überprüfen möchten: Sie können entweder die zu verwendende Zelle angeben, indem Sie die Spalten- (x) und Zeilenposition (y) der Zelle in den Argumenten `x` und `y` übergeben. Diese Positionen beginnen bei 1 für die erste Spalte und Zeile. Alternativ können Sie

die Zelle auch angeben, indem Sie ihre Position im Parameter `ref` übergeben. Dies kann entweder eine Zeichenkette sein, z.B. `"A10"` für die erste Zelle in der 10. Zeile oder ein Iteratorreferenz, die von dem Befehl `xlsx.CellRange()` zurückgegeben wird. Optional können Sie auch den Index des zu verwendenden Arbeitsblatts im optionalen Parameter `sheet` übergeben (beginnend mit 1 für das erste Arbeitsblatt). Wenn der Parameter `sheet` weggelassen wird, wird das mit `xlsx.SetDefaultSheet()` eingestellte Arbeitsblatt verwendet.

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments
`x`	Spaltenindex der zu verwendenden Zelle (beginnend bei 1)
`y`	Zeilenindex der zu verwendenden Zelle (beginnend bei 1)
`ref`	Zellposition (z.B. `"A1"` oder eine Iteratorreferenz), wird nur verwendet, wenn `x` und `y` weggelassen werden
`sheet`	optional: Index des zu verwendenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

RÜCKGABEWERTE

`bool`	`True`, wenn die Zelle eine Formel enthält, andernfalls `False`

4.24 xlsx.HideColumn

BEZEICHNUNG

xlsx.HideColumn – blendet eine Spalte ein- oder aus

ÜBERSICHT

```
xlsx.HideColumn(id, col, hidden[, sheet])
```

BESCHREIBUNG

Mit diesem Befehl kann die durch `col` angegebene Spalte ein- oder ausgeblendet werden. Die Spaltenindizes beginnen bei 1. Das Argument `hidden` muss auf `True` gesetzt werden, um die Spalte auszublenden, oder auf `False`, um sie wieder einzublenden. Optional können Sie auch den Index des zu verwendenden Arbeitsblatts im optionalen Parameter `sheet` übergeben (beginnend mit 1 für das erste Arbeitsblatt). Wenn der Parameter `sheet` weggelassen wird, wird das mit `xlsx.SetDefaultSheet()` eingestellte Arbeitsblatt verwendet.

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments
`col`	zu verwendender Spaltenindex (beginnend bei 1)
`hidden`	`True`, um die Spalte auszublenden, `False`, um sie anzuzeigen
`sheet`	optional: Index des zu verwendenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

4.25 xlsx.HideRow

BEZEICHNUNG

xlsx.HideRow – blendet eine Zeile ein- oder aus

ÜBERSICHT

```
xlsx.HideRow(id, row, hidden[, sheet])
```

BESCHREIBUNG

Mit diesem Befehl kann die durch `row` angegebene Zeile ein- oder ausgeblendet werden. Die Zeilenindizes beginnen bei 1. Das Argument `hidden` muss auf `True` gesetzt werden, um die Zeile auszublenden, oder auf `False`, um sie einzublenden. Optional können Sie auch den Index des zu verwendenden Arbeitsblatts im optionalen Parameter `sheet` übergeben (beginnend mit 1 für das erste Arbeitsblatt). Wenn der Parameter `sheet` weggelassen wird, wird das mit `xlsx.SetDefaultSheet()` eingestellte Arbeitsblatt verwendet.

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments
`row`	zu verwendender Zeilenindex (beginnend mit 1 für die erste Zeile)
`hidden`	`True`, um die Zeile auszublenden, `False`, um sie anzuzeigen
`sheet`	optional: Index des zu verwendenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

4.26 xlsx.IsColumnHidden

BEZEICHNUNG

xlsx.IsColumnHidden – gibt den Sichtbarkeitsstatus der Spalte zurück

ÜBERSICHT

```
hidden = xlsx.IsColumnHidden(id, col[, sheet])
```

BESCHREIBUNG

Dieser Befehl gibt `True` zurück, wenn die Spalte am Index `col` derzeit ausgeblendet ist, oder `False`, wenn sie sichtbar ist. Spaltenindizes beginnen bei 1. Optional können Sie auch den Index des zu verwendenden Arbeitsblatts im optionalen Parameter `sheet` übergeben (beginnend bei 1 für das erste Arbeitsblatt). Wenn der Parameter `sheet` weggelassen wird, wird das mit `xlsx.SetDefaultSheet()` eingestellte Arbeitsblatt verwendet.

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments
`col`	zu verwendender Spaltenindex (beginnend bei 1)
`sheet`	optional: Index des zu verwendenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

RÜCKGABEWERTE

`hidden`	`True`, wenn die Spalte ausgeblendet ist, andernfalls `False`

4.27 xlsx.IsRowHidden

BEZEICHNUNG

xlsx.IsRowHidden – gibt den Sichtbarkeitsstatus der Zeile zurück

ÜBERSICHT

```
hidden = xlsx.IsRowHidden(id, row[, sheet])
```

BESCHREIBUNG

Dieser Befehl gibt `True` zurück, wenn die Zeile am Index `col` derzeit ausgeblendet ist, oder `False`, wenn sie sichtbar ist. Zeilenindizes beginnen bei 1. Optional können Sie auch den Index des zu verwendenden Arbeitsblatts im optionalen Parameter `sheet` übergeben (beginnend bei 1 für das erste Arbeitsblatt). Wenn der Parameter `sheet` weggelassen wird, wird das mit `xlsx.SetDefaultSheet()` eingestellte Arbeitsblatt verwendet.

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments
`row`	zu verwendender Zeilenindex (beginnend bei 1)
`sheet`	optional: Index des zu verwendenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

RÜCKGABEWERTE

`hidden`	`True`, wenn die Zeile ausgeblendet ist, andernfalls `False`

4.28 xlsx.IsSheetActive

BEZEICHNUNG

xlsx.IsSheetActive – überprüft, ob das Arbeitsblatt aktiv ist

ÜBERSICHT

```
active = xlsx.IsSheetActive(id[, sheet])
```

BESCHREIBUNG

Dieser Befehl gibt `True` zurück, wenn das durch `sheet` angegebene Arbeitsblatt aktiv ist, andernfalls `False`. Der Parameter `sheet` ist optional. Wenn es weggelassen wird, wird das mit `xlsx.SetDefaultSheet()` eingestellte Arbeitsblatt verwendet. Die Blattindizes beginnen bei 1 für das erste Arbeitsblatt.

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments
`sheet`	optional: Index des zu verwendenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

RÜCKGABEWERTE

`active`	`True`, wenn das Arbeitsblatt aktiv ist, sonst `False`

4.29 xlsx.IsSheetSelected

BEZEICHNUNG
xlsx.IsSheetSelected – überprüft, ob das Arbeitsblatt ausgewählt ist

ÜBERSICHT
```
sel = xlsx.IsSheetSelected(id[, sheet])
```

BESCHREIBUNG
Dieser Befehl gibt `True` zurück, wenn das durch `sheet` angegebene Arbeitsblatt ausgewählt ist, andernfalls `False`. Der Parameter `sheet` ist optional. Wenn es weggelassen wird, wird das mit `xlsx.SetDefaultSheet()` eingestellte Arbeitsblatt verwendet. Die Blattindizes beginnen bei 1 für das erste Arbeitsblatt.

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments
`sheet`	optional: Index des zu verwendenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

RÜCKGABEWERTE

`sel`	`True`, wenn das Arbeitsblatt ausgewählt ist, andernfalls `False`

4.30 xlsx.MoveSheet

BEZEICHNUNG
xlsx.MoveSheet – ändert die Arbeitsblattposition

ÜBERSICHT
```
xlsx.MoveSheet(id, idx, newpos)
```

BESCHREIBUNG
Mit diesem Befehl kann die Position des Arbeitsblatts bei `idx` auf die durch `newpos` angegebene Position geändert werden. Arbeitsblattindizes werden ab 1 gezählt.

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments
`idx`	Arbeitsblatt, dessen Position geändert werden soll (beginnend bei 1)
`newpos`	gewünschte neue Position für das Arbeitsblatt (ab 1)

4.31 xlsx.Open

BEZEICHNUNG
xlsx.Open – öffnet ein XLSX-Dokument zum Lesen und/oder Schreiben

ÜBERSICHT
```
[id] = xlsx.Open(id, filename$)
```

BESCHREIBUNG

Dieser Befehl versucht, das durch `filename$` angegebene XLSX-Dokument zu öffnen und weist ihm `id` zu. Wenn Sie `Nil` in `id` übergeben, wählt `xlsx.Open()` automatisch einen freien Identifikator aus und gibt ihn zurück. Die in `filename$` angegebene Datei muss existieren, sonst schlägt dieser Befehl fehl. Wenn Sie ein neues xlsx-Dokument erstellen möchten, verwenden Sie den Befehl `xlsx.Create()`.

Obwohl xlsx.hwp automatisch alle geöffneten XLSX-Dokumente schließt, wenn es beendet wird, wird dringend empfohlen, dass Sie ein geöffnetes XLSX-Dokument schließen, wenn Sie damit fertig sind, indem Sie den Befehl `xlsx.Close()` verwenden, da Sie sonst Ressourcen verschwenden.

Beachten Sie, dass `xlsx.Open()` ein Standard-Hollywood-Objekt erstellt, das auch mit Befehlen aus Hollywoods Objektbibliothek wie `GetAttribute()`, `SetObjectData()`, `GetObjectData()` usw. verwendet werden kann. Siehe Abschnitt 4.14 [xlsx.GetObjectType], Seite 16, für Details.

EINGABEN

`id` ID des XLSX-Dokuments oder `Nil` für die automatische Id-Auswahl

`filename$`
Name der zu öffnenden Datei

RÜCKGABEWERTE

`id` optional: ID des Dokuments; wird nur zurückgegeben, wenn Sie `Nil` als Argument 1 übergeben (siehe oben)

BEISPIEL

```
xlsx.Open(1, "test.xlsx")
cols = xlsx.GetColumnCount(1)
rows = xlsx.GetRowCount(1)
For Local y = 1 To rows
   For Local x = 1 to cols
      DebugPrint((xlsx.GetCellValue(1, x, y)))
   Next
   DebugPrint("************************")
Next
xlsx.Close(1)
```

Der obige Code öffnet `test.xlsx` und gibt die Werte aller Zellen aus.

4.32 xlsx.Save

BEZEICHNUNG

xlsx.Save – speichert ein XLSX-Dokument

ÜBERSICHT

```
xlsx.Save(id)
```

BESCHREIBUNG

Dieser Befehl speichert das durch `id` angegebene XLSX-Dokument in der Datei, die beim Öffnen des XLSX-Dokuments mit `xlsx.Open()` oder beim Erstellen mit `xlsx.Create()`

angegeben wurde. Wenn Sie das XLSX-Dokument an einem anderen Ort speichern möchten, verwenden Sie `xlsx.SaveAs()`.

Beachten Sie, dass dieser Befehl das XLSX-Dokument nicht schließt. Sie müssen dennoch `xlsx.Close()` aufrufen, um alle mit dem XLSX-Dokument verknüpften Ressourcen freizugeben.

EINGABEN

`id` ID des zu verwendenden XLSX-Dokuments

4.33 xlsx.SaveAs

BEZEICHNUNG

xlsx.SaveAs – speichert ein XLSX-Dokument an neuem Ort

ÜBERSICHT

```
xlsx.SaveAs(id, filename$)
```

BESCHREIBUNG

Dieser Befehl speichert das durch `id` angegebene XLSX-Dokument an dem durch `filename$` angegebenen Speicherort. Wenn Sie das XLSX-Dokument nicht an einem neuen Ort speichern möchten, verwenden Sie stattdessen `xlsx.Save()`.

Beachten Sie, dass dieser Befehl das XLSX-Dokument nicht schließt. Sie müssen dennoch `xlsx.Close()` aufrufen, um alle mit dem XLSX-Dokument verknüpften Ressourcen freizugeben.

EINGABEN

`id` ID des zu verwendenden XLSX-Dokuments

`filename$`
gewünschter Speicherort für das XLSX-Dokument

4.34 xlsx.SetCellFormula

BEZEICHNUNG

xlsx.SetCellFormula – setzt die Zellenformel

ÜBERSICHT

```
xlsx.SetCellFormula(id, x, y, f$[, sheet])
xlsx.SetCellFormula(id, ref, f$[, sheet])
```

BESCHREIBUNG

Dieser Befehl setzt die Formel der angegebenen Zelle auf die in `f$` angegebene. Nach Aufruf von diesem Befehl gibt `xlsx.HaveCellFormula() True` zurück. Es gibt zwei Möglichkeiten, die Zelle anzugeben, deren Wert festgelegt werden soll: Sie können die zu verwendende Zelle entweder angeben, indem Sie die Spalten- (x) und Zeilenposition (y) der Zelle in den Argumenten `x` und `y` übergeben. Diese Positionen beginnen bei 1 für die erste Spalte und Zeile. Alternativ können Sie die Zelle auch angeben, indem Sie ihre Position im Parameter `ref` übergeben. Dies kann entweder eine Zeichenkette sein,

z.B. "A10" für die erste Zelle in der 10. Zeile oder ein Iteratorreferenz, die vom Befehl `xlsx.CellRange()` zurückgegeben wird. Optional können Sie auch den Index des zu verwendenden Arbeitsblatts im optionalen Parameter `sheet` übergeben (beginnend mit 1 für das erste Arbeitsblatt). Wenn der Parameter `sheet` weggelassen wird, wird das mit `xlsx.SetDefaultSheet()` eingestellte Arbeitsblatt verwendet.

Beachten Sie, dass die Formel ohne Gleichheitszeichen angegeben werden muss, z.B. müssen Sie "A1+A2" anstelle von "=A1+A2" verwenden. Beachten Sie auch, dass das XLSX-Plugin das Ergebnis der Formel nicht berechnet, d.h. Sie können nicht erwarten, dass `xlsx.GetCellValue()` das Berechnungsergebnis nach dem Festlegen einer Zellformel erhält. Um Formelwerte berechnen zu lassen, müssen Sie das XLSX-Dokument in Excel oder Calc von LibreOffice öffnen und speichern.

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments
`x`	Spaltenindex der zu verwendenden Zelle (beginnend bei 1)
`y`	Zeilenindex der zu verwendenden Zelle (beginnend bei 1)
`ref`	Zellposition (z.B. "A1" oder ein Iteratorreferenz), wird nur verwendet, wenn `x` und `y` weggelassen werden
`f$`	gewünschte Zellformel (ohne Gleichheitszeichen)
`sheet`	optional: Index des zu verwendenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

BEISPIEL

```
xlsx.SetCellFormula(1, "A3", "A1+A2")
```

Der obige Code setzt Zelle A3 auf die Summe der Zellen A1+A2.

4.35 xlsx.SetCellValue

BEZEICHNUNG

xlsx.SetCellValue – setzt den Zellenwert

ÜBERSICHT

```
xlsx.SetCellValue(id, x, y, val[, type, sheet])
xlsx.SetCellValue(id, ref, val[, type, sheet])
```

BESCHREIBUNG

Dieser Befehl setzt den Wert der angegebenen Zelle auf den in `val` angegebenen Wert. Es gibt zwei Möglichkeiten, die Zelle anzugeben, deren Wert festgelegt werden soll: Sie können die zu verwendende Zelle entweder angeben, indem Sie die Spalten- (x) und Zeilenposition (y) der Zelle in den Argumenten `x` und `y` übergeben. Diese Positionen beginnen bei 1 für die erste Spalte und Zeile. Alternativ können Sie die Zelle auch angeben, indem Sie ihre Position im Parameter `ref` übergeben. Dies kann entweder eine Zeichenkette sein, z.B. "A10" für die erste Zelle in der 10. Zeile oder ein Iteratorreferenz, die vom Befehl `xlsx.CellRange()` zurückgegeben wird. Optional können Sie auch den Index des zu verwendenden Arbeitsblatts im optionalen Parameter `sheet` übergeben

(beginnend mit 1 für das erste Arbeitsblatt). Wenn der Parameter `sheet` weggelassen wird, wird das mit `xlsx.SetDefaultSheet()` eingestellte Arbeitsblatt verwendet.

Optional können Sie auch den Werttyp im Argument `type` angeben. Normalerweise ist dies nicht notwendig, da `xlsx.SetCellValue()` den Werttyp anhand des Typs des in `val` übergebenen Arguments bestimmt. Aber da Hollywood nicht zwischen booleschen, ganzzahligen und Fließkommawerten unterscheidet, ist dies möglicherweise erforderlich. Übergeben Sie den Parameter `type`, um sicherzustellen, dass die Zelle auf den gewünschten Typ eingestellt wird. Der Parameter `type` kann eine der folgenden speziellen Konstanten sein:

`#INTEGER` Eine Ganzzahl.

`#DOUBLE` Ein Fließkommawert.

`#STRING` Ein Zeichenkettenwert.

`#BOOLEAN` Ein boolescher Wert (entweder `True` oder `False`).

`#NIL` Die Zelle ist leer.

Beachten Sie, dass es beim Versuch, die Werte vieler Zellen zu setzen, normalerweise viel schneller ist, den Befehl `xlsx.CellRange()` zusammen mit einer generischen For-Schleife zu verwenden, um die gewünschten Zellen zu durchlaufen. Dies wird besonders empfohlen, wenn es sich um große XLSX-Dokumente mit Tausenden von Zellen handelt.

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments
`x`	Spaltenindex der zu verwendenden Zelle (beginnend bei 1)
`y`	Zeilenindex der zu verwendenden Zelle (beginnend bei 1)
`ref`	Zellposition (z.B. `"A1"` oder ein Iteratorreferenz),
`val`	gewünschter Zellenwert
`type`	optional: Typ des Wertes (mögliche Konstanten siehe oben)
`sheet`	optional: Index des zu verwendenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

BEISPIEL

```
xlsx.Create(1, "test.xlsx")
For Local y = 1 To 100
   For Local x = 1 to 30
      xlsx.SetCellValue(1, x, y, "Cell " .. x .. "/" .. y)
   Next
Next
xlsx.Save(1)
xlsx.Close(1)
```

Der obige Code erstellt ein neues XLSX-Dokument und fügt ihm 30 Spalten und 100 Zeilen hinzu. Das Dokument wird als `test.xlsx` gespeichert.

4.36 xlsx.SetColumnWidth

BEZEICHNUNG

xlsx.SetColumnWidth – setzt die Spaltenbreite

ÜBERSICHT

```
xlsx.SetColumnWidth(id, col, width[, sheet])
```

BESCHREIBUNG

Dieser Befehl setzt die Breite der in `col` angegebenen Spalte auf `width`. Die Spaltenindizes beginnen bei 1. Die Breite wird in Schrifteinheiten der normalen Display-Schriftart angegeben und kann ein Bruchwert sein. Optional können Sie auch den Index des zu verwendenden Arbeitsblatts im optionalen Parameter `sheet` übergeben (beginnend mit 1 für das erste Arbeitsblatt). Wenn der Parameter `sheet` weggelassen wird, wird das mit `xlsx.SetDefaultSheet()` eingestellte Arbeitsblatt verwendet.

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments
`col`	zu verwendender Spaltenindex (beginnend bei 1)
`width`	gewünschte Spaltenbreite in Schriftarteinheiten
`sheet`	optional: Index des zu verwendenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

BEISPIEL

```
xlsx.SetColumnWidth(1, 1, 8.43)
```

Der obige Code legt die Breite der ersten Spalte auf 8.43 fest.

4.37 xlsx.SetDefaultSheet

BEZEICHNUNG

xlsx.SetDefaultSheet – legt das Standardarbeitsblatt fest

ÜBERSICHT

```
xlsx.SetDefaultSheet(id, idx)
```

BESCHREIBUNG

Dieser Befehl kann verwendet werden, um das Standardarbeitsblatt für das durch `id` angegebene XLSX-Dokument festzulegen. Das Standardarbeitsblatt eines XLSX-Dokuments ist das Arbeitsblatt, das verwendet werden soll, falls kein Arbeitsblatt explizit angegeben wird, wenn Befehle wie `xlsx.SetCellValue()` oder `xlsx.GetCellValue()` aufgerufen werden. Sie müssen die Position des gewünschten Standardarbeitsblatts im Argument `idx` übergeben. Die Arbeitsblattindizes beginnen bei 1 für das erste Arbeitsblatt.

Voreingestellt ist, dass das erste Arbeitsblatt im XLSX-Dokument das Standardarbeitsblatt ist.

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments

`idx` Position des Arbeitsblatts, das zum Standard gemacht werden soll (beginnend mit 1)

4.38 xlsx.SetProperty

BEZEICHNUNG

xlsx.SetProperty – stellt die Dokumenteigenschaft ein

ÜBERSICHT

```
xlsx.SetProperty(id, prop, val$)
```

BESCHREIBUNG

Mit diesem Befehl können Sie die durch `prop` angegebene Dokumenteigenschaft auf den durch `val$` angegebenen Wert setzen. Der Parameter `prop` muss eine der folgenden speziellen Konstanten sein:

```
#XLSX_PROPERTY_TITLE
#XLSX_PROPERTY_SUBJECT
#XLSX_PROPERTY_CREATOR
#XLSX_PROPERTY_KEYWORDS
#XLSX_PROPERTY_DESCRIPTION
#XLSX_PROPERTY_LASTMODIFIEDBY
#XLSX_PROPERTY_LASTPRINTED
#XLSX_PROPERTY_CREATIONDATE
#XLSX_PROPERTY_MODIFICATIONDATE
#XLSX_PROPERTY_CATEGORY
#XLSX_PROPERTY_APPLICATION
#XLSX_PROPERTY_DOCSECURITY
#XLSX_PROPERTY_SCALECROP
#XLSX_PROPERTY_MANAGER
#XLSX_PROPERTY_COMPANY
#XLSX_PROPERTY_LINKSUPTODATE
#XLSX_PROPERTY_SHAREDDOC
#XLSX_PROPERTY_HYPERLINKBASE
#XLSX_PROPERTY_HYPERLINKSCHANGED
#XLSX_PROPERTY_APPVERSION
```

EINGABEN

`id` ID des zu verwendenden XLSX-Dokuments

`prop` zu setzende Eigenschaft (siehe oben für mögliche Werte)

`val$` gewünschter Wert für die Dokumenteigenschaft

4.39 xlsx.SetRowHeight

BEZEICHNUNG

xlsx.SetRowHeight – setzt die Zeilenhöhe

ÜBERSICHT

```
xlsx.SetRowHeight(id, row, height[, sheet])
```

BESCHREIBUNG

Dieser Befehl setzt die Höhe der in `row` angegebenen Zeile auf `height`. Die Zeilenindizes beginnen bei 1. Die Höhe wird in Schriftarteinheiten der normalen Display-Schriftart angegeben und kann ein Bruchwert sein. Optional können Sie auch den Index des zu verwendenden Arbeitsblatts im optionalen Parameter `sheet` übergeben (beginnend mit 1 für das erste Arbeitsblatt). Wenn der Parameter `sheet` weggelassen wird, wird das mit `xlsx.SetDefaultSheet()` eingestellte Arbeitsblatt verwendet.

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments
`row`	zu verwendender Zeilenindex (beginnend mit 1)
`height`	gewünschte Zeilenhöhe in Schriftarteinheiten
`sheet`	optional: Index des zu verwendenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

BEISPIEL

```
xlsx.SetRowHeight(1, 1, 12.75)
```

Der obige Code setzt die Höhe der ersten Zeile auf 12.75.

4.40 xlsx.SetSheetActive

BEZEICHNUNG

xlsx.SetSheetActive – macht ein Arbeitsblatt aktiv

ÜBERSICHT

```
xlsx.SetSheetActive(id[, sheet])
```

BESCHREIBUNG

Dieser Befehl macht das durch den Parameter `sheet` angegebene Arbeitsblatt zum Aktiven. Der `sheet`-Parameter ist optional. Wenn er weggelassen wird, wird das von `xlsx.SetDefaultSheet()` eingestellte Arbeitsblatt verwendet. Die Blattindizes beginnen bei 1 für das erste Arbeitsblatt.

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments
`sheet`	optional: Index des zu verwendenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

4.41 xlsx.SetSheetName

BEZEICHNUNG

xlsx.SetSheetName – setzt den Arbeitsblattnamen

ÜBERSICHT

```
xlsx.SetSheetName(id, idx, name$)
```

BESCHREIBUNG

Dieser Befehl setzt den Namen des Arbeitsblatts an der durch `idx` angegebenen Position auf die in `name$` übergebene Zeichenkette. Die Arbeitsblattpositionen beginnen bei 1.

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments
`idx`	Position des Arbeitsblatts, dessen Name festgelegt werden soll
`name$`	gewünschter Arbeitsblattname

4.42 xlsx.SetSheetSelected

BEZEICHNUNG

xlsx.SetSheetSelected – wählt ein Arbeitsblatt aus oder ab

ÜBERSICHT

```
xlsx.SetSheetSelected(id, sel[, sheet])
```

BESCHREIBUNG

Dieser Befehl kann verwendet werden, um das durch den Parameter `sheet` angegebene Arbeitsblatt aus- oder abzuwählen. Wenn der Parameter `sel` auf `True` gesetzt ist, wird das Blatt ausgewählt, andernfalls wird es abgewählt. Der Parameter `sheet` ist optional. Wenn es weggelassen wird, wird das mit `xlsx.SetDefaultSheet()` eingestellte Arbeitsblatt verwendet. Die Blattindizes beginnen bei 1 für das erste Arbeitsblatt.

EINGABEN

`id`	ID des zu verwendenden XLSX-Dokuments
`sel`	`True`, um das Blatt auszuwählen, `False`, um es abzuwählen
`sheet`	optional: Index des zu verwendenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

4.43 xlsx.SetSheetVisibility

BEZEICHNUNG

xlsx.SetSheetVisibility – setzt den Arbeitsblatt-Sichtbarkeitstatus

ÜBERSICHT

```
xlsx.SetSheetVisibility(id, vis[, sheet])
```

BESCHREIBUNG

Dieser Befehl kann verwendet werden, um den Sichtbarkeitsstatus des durch den Parameter `sheet` angegebenen Arbeitsblatts festzulegen. Der Parameter `vis` muss eine der folgenden speziellen Konstanten sein:

`#XLSX_VISIBILITY_VISIBLE`
: Das Arbeitsblatt ist sichtbar.

`#XLSX_VISIBILITY_HIDDEN`
: Das Blatt ist ausgeblendet, kann aber von Benutzern eingeblendet werden, die die XLSX-Datei in einer Tabellenkalkulations-App öffnen.

`#XLSX_VISIBILITY_VERYHIDDEN`
: Das Blatt ist ausgeblendet und kann von Benutzern, die die XLSX-Datei in einer Tabellenkalkulations-App öffnen, nicht eingeblendet werden.

Der Parameter `sheet` ist optional. Wenn er weggelassen wird, wird das mit `xlsx.SetDefaultSheet()` eingestellte Arbeitsblatt verwendet. Die Blattindizes beginnen bei 1 für das erste Arbeitsblatt.

EINGABEN

`id` ID des zu verwendenden XLSX-Dokuments

`vis` gewünschter Sichtbarkeitsstatus des Arbeitsblattes (siehe oben für mögliche Werte)

`sheet` optional: Index des zu verwendenden Arbeitsblatts (standardmäßig der Index des Standardarbeitsblatts)

4.44 xlsx.UseSharedStrings

BEZEICHNUNG

xlsx.UseSharedStrings – schaltet den SharedString-Modus um

ÜBERSICHT

`xlsx.UseSharedStrings(on)`

BESCHREIBUNG

Mit diesem Befehl können Sie steuern, ob Zeichenketten, die Zellen zugewiesen sind, in einer globalen gemeinsam genutzten Zeichenkettentabelle (SharedString-Tabelle) im XLSX gespeichert werden sollen oder ob sie einzeln in die Zellknoten eingebettet werden sollen. Es ist in der Regel effizienter, eine globale gemeinsam genutzte Zeichenkettentabelle zu verwenden, da identische Zeichenketten nur einmal in dieser Tabelle gespeichert werden müssen, wodurch die Dateigröße verringert wird, falls viele identische Zeichenketten vorhanden sind. Wenn Sie aus irgendeinem Grund keine globale gemeinsam genutzte Zeichenkettentabelle verwenden möchten, können Sie `xlsx.UseSharedStrings()` verwenden, um diese Funktionalität zu deaktivieren, indem Sie `False` im `on`-Parameter übergeben.

Beachten Sie, dass dieser Befehl nur wirksam ist, wenn einem Dokument neue Zeichenketten hinzugefügt werden. Wenn Sie ein Dokument öffnen, das gemeinsam genutzte Zeichenketten verwendet, und es erneut speichern, bleiben die gemeinsam genutzten Zeichenketten erhalten, auch wenn Sie den Modus für gemeinsam genutzte Zeichenketten mit diesem Befehl deaktiviert haben. `xlsx.UseSharedStrings()` wirkt sich nur auf neue Zeichenketten aus, die dem Dokument hinzugefügt werden.

Standardmäßig ist die globale gemeinsame Zeichenkettentabelle (SharedString-Tabelle) aktiviert.

EINGABEN

`on`	`True`, um die globale gemeinsame Stringtabelle zu aktivieren, `False`, um sie zu deaktivieren

Anhang A Lizenzen

A.1 OpenXLSX license

A.2 pugixml license

A.3 MiniZ license

Index