MyrScript Language Guide

For MyrScript 1.6.3

Authors : Didier & Olivier Guillion
Date : 3/17/2020
Copyright : Myriad Software, http://www.myriad-online.com

MyrScript Project is now 112736 C source lines.

(c) 2003-2020 **Myriad Software**
All rights reserved.
This book is intended to assist MyrScript developers to develop scripts for **Myriad Software** products.
No licenses, express or implied, are granted with respect to any of the technology described in this book.
Myriad Software retains all intellectual property rights associated with the technology described in this book.
Every effort has been made to ensure that the information in this manual is accurate. **Myriad Software** is not responsible for typographical errors.

Even though **Myriad Software** has reviewed this manual, and the MyrScript language **Myriad Software** MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
AS A RESULT, THIS MANUAL AND THE MYRSCRIPT LANGUAGE ARE PROVIDED "AS IS", AND YOU, THE USER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.
IN NO EVENT WILL **Myriad Software** BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY OF MYRSCRIPT LANGUAGE.
THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED.
No **Myriad Software** dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Myriad Software, the **Myriad Software** logo, **MyrScript**, **Harmony Assistant**, **Virtual Singer** are trademarks of **Myriad Software**
Use of the **Myriad Software** logo for commercial purposes without the prior written consent of **Myriad Software** may constitute trademark infringement and unfair competition in violation of federal and state laws.
Apple and Macintosh are trademarks of Apple, Inc., registered in the U.S. and other countries.
Windows is a trademark of Microsoft, Inc., registered in the U.S. and other countries.
Adobe, Adobe Illustator and Adobe Photoshop are trademarks of Adobe, Inc., registered in the U.S. and other countries.

Myriad Software
26 rue Michel de Montaigne
31200 Toulouse, France

Contents

1. General chapters (page 27)

1.1. The developer manual (page 27)

1.1.1. Overview (page 27)
1.1.2. Using this hypertext manual (page 27)
The index (page 27)
Searching the manual (page 27)
Contextual help (page 27)
Internet link (page 27)
Jump to target (page 28)
Interactives Examples (page 28)
1.1.3. Printing the manual (page 28)
1.1.4. Platform dependence (page 28)
1.1.5. Notes to MyrScript developer (page 28)

1.2. MyrScript Versions (page 29)

1.2.1. Version 0.1 (page 29)
1.2.2. Version 0.1.6 (page 29)
1.2.3. Version 0.1.7 (page 29)
1.2.4. Version 0.1.8 (page 29)
1.2.5. Version 0.1.9 (page 29)
1.2.6. Version 0.2.0 (page 30)
1.2.7. Version 0.2.1 (page 30)
1.2.8. Version 0.2.2 (page 31)
1.2.9. Version 1.0.0 (page 32)
1.2.10. Version 1.1.0 (page 32)
1.2.11. Version 1.1.1 (page 32)
1.2.12. Version 1.1.2 (page 32)
1.2.13. Version 1.1.3 (page 33)
1.2.14. Version 1.1.4 (page 33)
1.2.15. Version 1.2 (page 34)
1.2.16. Version 1.2.1 (page 35)
1.2.17. Version 1.2.2 (page 35)
1.2.18. Version 1.2.3 (page 36)
1.2.19. Version 1.3.0 (page 37)
1.2.20. Version 1.3.1 (page 38)
1.2.21. Version 1.4 (page 39)
1.2.22. Version 1.4.0 (page 39)
1.2.23. Version 1.4.1 (page 39)
1.2.24. Version 1.4.2 (page 39)
1.2.25. Version 1.4.3 (page 40)
1.2.26. Version 1.4.4 (page 40)
1.2.27. Version 1.4.5 (page 41)
1.2.28. Version 1.4.6 (page 42)
1.2.29. Version 1.4.7 (page 43)
1.2.30. Version 1.4.8 (page 43)
1.2.31. Version 1.4.9 (page 43)
1.2.32. Version 1.5.0 (page 44)
1.2.33. Version 1.5.1 (page 44)
1.2.34. Version 1.6.2 (page 44)
1.2.35. Version 1.6.3 (page 44)

1.3. Scripts Menu (page 45)

1.3.1. Overview (page 45)
1.3.2. Launching a script from the menu (page 45)
1.3.3. Priority of script (page 45)
1.3.4. Creating a new script (page 45)
1.3.5. Opening and existing script (page 45)
1.3.6. Stopping script execution (page 45)
1.3.7. Downloading the most recent version of this manual (page 45)
1.3.8. Script version (page 45)
1.3.9. About scripts... (page 46)
1.3.10. Hiding a script on a given type of computer (page 47)
1.3.11. Packaged scripts (page 47)
1.3.12. Adding scripts to standard menus (page 47)
1.3.13. File importer (page 48)
1.3.14. Keyboard shorcuts (page 48)
1.3.15. Searching for scripts (page 49)
1.3.16. Special considerations (page 49)

1.4. Editing scripts (page 50)

1.4.1. Overview (page 50)
1.4.2. What is a Script ? (page 50)
1.4.3. Creating a new script file (page 50)
1.4.4. Editing a script file (page 51)
1.4.5. Autocomplete (page 51)
1.4.6. Printing a script (page 51)
1.4.7. Protecting a script (page 51)
1.4.8. Undo/redo (page 51)
1.4.9. Starting from a predefined script (page 52)
1.4.10. The placard (page 52)
Interface Composer area (page 52)
Include area (page 52)
Functions area (page 52)
Command area (page 52)
Target area (page 52)
Line number area (page 52)
1.4.11. Saving a script file (page 52)
1.4.12. Scripts and user's palette (page 52)
1.4.13. Creating a MyrScript Archive file (page 52)
Overview (page 53)
Creating an archive (page 53)
Archiving packaged scripts (page 53)
Archiving unpackaged scripts (page 53)
Archive file format (page 54)
1.4.14. Submitting a script (page 54)
1.4.15. Deleting a script file (page 54)
1.4.16. Syntax balance (page 54)
1.4.17. Syntax coloring (page 55)
1.4.18. Running a script (page 55)
1.4.19. Adding and deleting breakpoints (page 55)

1.5. The Interface Composer (page 56)

1.5.1. Overview (page 56)
1.5.2. What is Dialog Box ? (page 56)
1.5.3. What is an Item ? (page 57)
1.5.4. What is an Interface Composer file ? (page 57)
1.5.5. Editing (page 57)
Editing the dialog box (page 57)
Editing the script related to an item (page 58)
Editing the script related to the dialog box (page 58)
1.5.6. Saving an Interface Composer file (page 58)
1.5.7. Deleting an Interface Composer file (page 58)
1.5.8. Debugging (page 58)
1.5.9. Opening a Dialog Box (page 58)
1.5.10. Closing a Dialog Box (page 58)
1.5.11. Methods (page 59)
1.5.12. Item alignment (page 60)
1.5.13. Generating report (page 60)
1.5.14. Optimizing (page 60)
1.5.15. Items types (page 61)
Button item type (page 61)
Checkbox item type (page 62)
Radio item type (page 62)
Static Text item type (page 62)
Edit Text item type (page 62)
Pop-up Menu item type (page 63)
Color item type (page 63)
Style item type (page 63)
HeadShape item type (page 64)
Duration item type (page 65)
Scroll bar item type (page 65)
Slider item type (page 66)
Knob item type (page 66)
Frame item type (page 66)
List item type (page 67)
Tab View item type (page 68)
Text item type (page 68)
User item type (page 69)

1.6. Floating windows (page 70)

1.6.1. Overview (page 70)
1.6.2. What is a Floating window ? (page 70)
1.6.3. How to create a Floating window ? (page 70)
1.6.4. Special considerations (page 70)
1.6.5. High-level events (page 71)
1.6.6. Floating window from the user point of view (page 72)
1.6.7. Multiple Floating windows consideration (page 72)
1.6.8. Palette or dialog (page 72)
1.6.9. Translucent ratio (page 72)
1.6.10. Optimizing (page 72)

1.7. The Search Tool (page 73)

1.7.1. Overview (page 73)
1.7.2. Searching (page 73)
1.7.3. Searching again (page 73)
1.7.4. Replacing (page 73)
1.7.5. Fast Search (page 73)

1.8. Printing texts (page 74)

1.8.1. Overview (page 74)
1.8.2. Selecting a printing scale (page 74)
1.8.3. Syntax coloring (page 74)

1.9. The Log Window (page 75)

1.10. Debugging a script (page 76)

1.10.1. Overview (page 76)
1.10.2. Debugging Interface (page 76)
Run Icon (page 76)
Step into Icon (page 76)
Step outside Icon (page 76)
Stop Icon (page 76)
Delete all breakpoints Icon (page 76)
1.10.3. Breakpoints (page 76)
1.10.4. Data browser (page 77)
1.10.5. Performances (page 77)
1.10.6. Crashes (page 77)

1.11. The Lua language (page 78)

1.11.1. Overview (page 78)
1.11.2. Types and Tags (page 78)
1.11.3. The Language (page 78)
Lexical Conventions (page 79)
Coercion (page 79)
Adjustment (page 80)
Statements (page 80)
Blocks (page 80)
Assignment (page 80)
Control Structures (page 81)
For Statement (page 82)
Function Calls as Statements (page 83)
Local Declarations (page 84)
Expressions (page 85)
Visibility and Upvalues (page 89)
1.11.4. Standard Libraries (page 89)
Basic Functions (page 90)
String Manipulation (page 94)
Mathematical Functions (page 96)
1.11.5. The Complete Syntax of Lua (page 97)
1.11.6. Notes from the Myriad Team (page 98)
Division by zero (page 98)
Checking false and true or nil (page 98)
Non-integer table (array) index (page 98)
1.11.7. Deviation from the Lua standard syntax (page 99)
Object types (page 99)
Calling methods (page 99)
Type values (page 99)
Object content (page 99)
Object comparison (page 99)

1.12. MSDefine file (page 100)

1.12.1. Overview (page 100)
1.12.2. Using it (page 100)
1.12.3. Modifying it (page 100)
1.12.4. Content (page 101)

1.13. MSSounds file (page 110)

1.13.1. MSSounds file Overview (page 110)
1.13.2. Using MSSounds definitions (page 110)
1.13.3. Modifying it (page 110)

1.14. MSLibrary file (page 111)

1.14.1. Overview (page 111)

1.14.2. Using it (page 111)
1.14.3. Modifying it (page 111)
1.14.4. Content (page 112)

1.15. Guidelines (page 113)

1.15.1. Overview (page 113)

1.16. FAQ (page 114)

Using DLL calls (page 114)

2. Manuel Francais (page 115)

2.1. Manuel du programmeur (page 115)

2.1.1. Généralités (page 115)
2.1.2. Comment utiliser le manuel en hypertexte (page 115)
L'index (page 115)
Recherche dans le manuel (page 115)
Aide contextuelle (page 115)
Lien internet (page 115)
Saut vers une cible (page 116)
Exemples interactifs (page 116)
2.1.3. Imprimer le manuel (page 116)
2.1.4. Dépendance de l'environnement (page 116)
2.1.5. Notes à l'intention du programmeur MyrScript (page 116)

2.2. Le menu Scripts (page 117)

2.2.1. Introduction (page 117)
2.2.2. Lancement d'un script depuis le menu (page 117)
2.2.3. Création d'un nouveau script (page 117)
2.2.4. Ouverture d'un script existant (page 117)
2.2.5. Arrêt de l'exécution d'un script (page 117)
2.2.6. Téléchargement de la dernière version de ce manuel (page 117)
2.2.7. Version d'un script (page 117)
2.2.8. À propos des scripts... (page 118)
2.2.9. Cacher un script sur un type donné d'ordinateur (page 118)
2.2.10. Scripts empaquetés (page 119)
2.2.11. Ajout de scripts dans les menus standard (page 119)
2.2.12. Raccourci clavier (page 119)
2.2.13. Recherche de scripts (page 119)
2.2.14. À noter (page 119)

2.3. Édition de scripts (page 120)

2.3.1. Généralités (page 120)
2.3.2. Qu'est-ce qu'un script ? (page 120)
2.3.3. Création d'un script (page 120)
2.3.4. Édition d'un nouveau fichier script (page 121)
2.3.5. Impression d'un script (page 121)
2.3.6. Protection d'un script (page 121)
2.3.7. Démarrage à partir d'un script existant (page 121)
2.3.8. Le placard (page 121)
Zone d'Interface Composer (page 122)
2.3.9. Enregistrement d'un script (page 122)
2.3.10. Scripts et palette utilisateur (page 122)
2.3.11. Création d'une archive MyrScript (page 122)
Généralités (page 122)
Création d'une archive (page 123)
Archivage des scripts empaquetés (page 123)
Archivage de scripts non-empaquetés (page 123)
Format du fichier archive (page 123)
2.3.12. Proposition de script (page 123)
2.3.13. Annuler/Refaire (page 124)
2.3.14. Effacement d'un fichier de script (page 124)
2.3.15. Équilibrage syntaxique (page 124)
2.3.16. Couleurs et syntaxe (page 124)
2.3.17. Exécution d'un script (page 125)
2.3.18. Ajout et suppression de points d'arrêt (page 125)

2.4. L'Interface Composer (page 126)

2.4.1. Généralités (page 126)
2.4.2. Qu'est-ce qu'une boîte de dialogue ? (page 126)
2.4.3. Qu'est qu'un composant (item)? (page 127)
2.4.4. Qu'est-ce qu'un fichier de l'Interface Composer ? (page 127)
2.4.5. Edition (page 127)
Edition de la boîte de dialogue (page 128)
Edition du script lié à un composant (page 128)

Edition d'un script lié à la boîte de dialogue (page 128)
2.4.6. Enregistrer un fichier Interface Composer (page 128)
2.4.7. Débogage (page 128)
2.4.8. Fermeture d'une boîte de dialogue (page 129)
2.4.9. Alignement des composants (page 130)
2.4.10. Création de rapport (page 130)
2.4.11. Optimisation (page 130)
2.4.12. Types de composants (page 131)
Bouton (page 132)
Case à cocher (page 132)
Bouton Radio (page 132)
Texte Statique (page 132)
Texte Editable (page 133)
Menu déroulant (page 133)
Sélecteur de couleurs (page 133)
Sélecteur de Style de Texte (page 133)
Sélecteur de Tête de Note (page 134)
Sélecteur de durée (page 135)
Barre de Défilement (page 135)
Barre Graduée (page 136)
Potentiomètre (page 136)
Cadre (page 136)
Liste (page 137)
Onglets (page 138)
Memo (page 138)
Utilisateur (page 139)

2.5. Fenêtre flottante (page 140)

2.5.1. Généralités (page 140)
2.5.2. Qu'est-ce qu'une fenêtre flottante ? (page 140)
2.5.3. Comment créer une fenêtre flottante ? (page 140)
2.5.4. A noter (page 140)
2.5.5. Événements (page 141)
2.5.6. Les fenêtres flottantes du point de vue de l'utilisateur (page 142)
2.5.7. A propos des fenêtres flottantes multiples (page 142)
2.5.8. Palette ou Dialogue (page 142)
2.5.9. Transparence (page 142)
2.5.10. Optimisation (page 142)

2.6. L'Outil de Recherche (page 143)

2.6.1. Généralités (page 143)
2.6.2. Recherche (page 143)
2.6.3. Continuer la recherche (page 143)
2.6.4. Remplacer (page 143)
2.6.5. Recherche rapide (page 143)

2.7. Impression de Textes (page 144)

2.7.1. Généralités (page 144)
2.7.2. Choix de l'échelle d'impression (page 144)
2.7.3. Coloration syntaxique (page 144)

2.8. Fenêtre des Logs (compte-rendu) (page 145)

2.9. Débogage d'un script (page 146)

2.9.1. Généralités (page 146)
2.9.2. L'interface de débogage (page 146)
L'icône Exécuter (page 146)
L'icône Pas à Pas détaillé (page 146)
L'icône Pas à Pas global (page 146)
L'icône Arrêt (page 146)
L'icône Effacer tous les points d'arrêt (page 146)
2.9.3. Points d'arrêt (page 146)
2.9.4. Fenêtre des variables (page 147)
2.9.5. Performances (page 147)
2.9.6. Plantages (page 147)

2.10. Le langage Lua (page 148)

2.10.1. Généralités (page 148)
2.10.2. Types et Balises (page 148)
2.10.3. Le Langage (page 149)
Conventions Lexicales (page 149)
Conversion (page 150)
Ajustement (page 150)
Déclarations (page 150)
Blocs (page 150)
Assignation (page 151)

Structures de Contrôle (page 151)
Déclaration For (page 152)
Déclarations d'appels de fonctions (page 153)
Déclarations locales (page 154)
Expressions (page 155)
Visibilité and valeurs (page 159)
2.10.4. Bibliothèques Standard (page 160)
Fonctions de base (page 160)
Manipulation de chaîne (page 165)
Fonction mathématiques (page 167)
2.10.5. La syntaxe complète de Lua (page 168)
2.10.6. Notes de l'équipe Myriad (page 169)
Division par zéro (page 169)
Vérification du faux, vrai et nul (page 169)
index de table (tableau) non-entier (page 169)
2.10.7. Modification de la syntaxe standard de Lua (page 169)
Types Objets (page 170)
Méthodes d'appel (page 170)
Valeurs (page 170)
Contenu des objets (page 170)
Comparaison d'objets (page 170)

2.11. Lexique (page 171)

2.11.1. Généralités (page 171)

3. Global keywords (page 172)
3.1. Script (page 172)
Include(...) (page 172)
Use(...) (page 172)
Run(...) (page 172)
4. Object types (page 173)

4.1. Overview (page 173)

4.2. Application Object Type (page 174)

4.2.1. Application global functions (page 174)
4.2.1.1. Script (page 174)
SetErrorMessage(...) (page 174)
4.2.1.2. Binary operation (page 174)
BinAnd(...) (page 174)
BinOr(...) (page 174)
BinXor(...) (page 174)
BinShl(...) (page 174)
BinShr(...) (page 174)
4.2.1.3. Bit operation (page 174)
BitTst(...) (page 175)
BitClr(...) (page 175)
BitSet(...) (page 175)
4.2.1.4. String operations (page 175)
ConvertStringEncoding(...) (page 175)
RawStrlen(...) (page 176)
GetRawChar(...) (page 176)
SetRawChar(...) (page 176)
AddRawChar(...) (page 176)
DeleteRawChar(...) (page 177)
4.2.1.5. Local object copy management (page 177)
GetLocalCopy(...) (page 177)
PutLocalCopy(...) (page 177)
4.2.1.6. Styled string (page 177)
PurgeStringStyle(...) (page 177)
DeleteStringStyle(...) (page 178)
InsertStringStyle(...) (page 178)
SetStringStyle(...) (page 178)
GetStringStyle(...) (page 178)
4.2.2. Application values (page 179)
4.2.2.1. Human Voice (page 179)
Application.humanVoice (page 179)
4.2.2.2. Interface (page 179)
Application.Palette (page 179)
Application.CurrentSymbolDuration (page 179)
Application.IsNotePaletteActive (page 179)
4.2.2.3. Playing Music (page 179)
Application.CurrentPlayingScore (page 179)
4.2.2.4. Settings (page 179)
Application.Language (page 180)
Application.MyrScriptVersion (page 180)
Application.Version (page 180)
Application.Computer (page 180)
Application.IsActive (page 181)
Application.CurrentSoundBaseVersion (page 181)

Application.IsAngloSaxonNotation (page 181)
Application.IsLeftHandedNotation (page 181)
Application.IsInvertionNotation (page 181)
Application.RGBSelectColor (page 181)
Application.StorageArea (page 181)
Application.LimitEditingToOneBar (page 181)
Application.NumberOfChordDiagrams (page 181)
4.2.2.5. Sound Base (page 181)
Application.CurrentSoundBaseName (page 181)
4.2.2.6. Virtual Singer (page 181)
Application.IsVirtualSingerInstalled (page 182)
Application.VirtualSingerVersion (page 182)
Application.IsVirtualSingerActive (page 182)
4.2.3. Application methods (page 182)
4.2.3.1. Converting (page 182)
Application.ConvertStringToBase64(...) (page 182)
Application.ConvertFileToBase64(...) (page 182)
Application.ConvertBase64ToFile(...) (page 182)
Application.ConvertBase64ToString(...) (page 182)
Application.ConvertChordNameToElements(...) (page 183)
Application.ConvertElementsToChordName(...) (page 183)
Application.ConvertPitchListToChord(...) (page 183)
Application.GetChordDiagramFromIndex(...) (page 184)
Application.GetDigitalOutputDelay(...) (page 184)
Application.SetDigitalOutputDelay(...) (page 184)
Application.StartRecording(...) (page 184)
Application.StopRecording(...) (page 185)
Application.AbortRecording(...) (page 185)
Application.GetCurrentRecordedLength(...) (page 185)
Application.GetLiveRecordedData(...) (page 185)
4.2.3.2. Date & Time (page 186)
Application.GetCurrentDate(...) (page 186)
Application.ConvertSystemTimeToDate(...) (page 186)
Application.ConvertDateToSystemTime(...) (page 186)
Application.TickCount(...) (page 186)
Application.WaitDelay(...) (page 187)
Application.StringDuration(...) (page 187)
4.2.3.3. Devices (page 187)
Application.UserBreak(...) (page 187)
Application.KeyStroke(...) (page 187)
Application.IsKeyCodePressed(...) (page 188)
Application.SetMouseCursorShape(...) (page 188)
Application.GetMouseButton(...) (page 189)
Application.GetMouseDoubleClickButton(...) (page 189)
Application.GetMousePosition(...) (page 189)
Application.GetMouseGlobalPosition(...) (page 189)
4.2.3.4. Display (page 189)
Application.UpdateScores(...) (page 189)
Application.UpdateAllWindows(...) (page 189)
4.2.3.5. Floating Window (page 189)
Application.IsFloatingWindowOpen(...) (page 189)
Application.Close(...) (page 189)
4.2.3.6. Getting Virtual Singer language info (page 189)
Application.GetVSLangDetails(...) (page 190)
4.2.3.7. Getting Virtual Singer language list (page 190)
Application.GetVSLangList(...) (page 190)
4.2.3.8. Human Voice (page 190)
Application.Say(...) (page 190)
Application.IsSaying(...) (page 191)
Application.SayOnFile(...) (page 191)
Application.SetHumanVoice(...) (page 191)
Application.SetHumanVoiceDefault(...) (page 191)
Application.SetHumanVoiceRelativePitch(...) (page 191)
Application.SetHumanVoicePitch(...) (page 191)
Application.GetHumanVoicePitch(...) (page 191)
Application.SetHumanVoicePitchModulation(...) (page 191)
Application.GetHumanVoicePitchModulation(...) (page 192)
Application.SetHumanVoiceRate(...) (page 192)
Application.GetHumanVoiceRate(...) (page 192)
Application.SetHumanVoiceVolume(...) (page 192)
Application.GetHumanVoiceVolume(...) (page 192)
Application.GetNumberOfHumanVoices(...) (page 192)
Application.GetHumanVoiceInformation(...) (page 192)
Application.SearchHumanVoice(...) (page 193)
4.2.3.9. Language (page 194)
Application.DumpKeyWordList(...) (page 194)
4.2.3.10. Log Window (page 195)
Application.ClearLog(...) (page 195)
Application.SaveLog(...) (page 195)
4.2.3.11. Manual (page 195)
Application.OpenManual(...) (page 195)
4.2.3.12. MyrWeb (page 195)
Application.WebExportSetup(...) (page 195)

Application.GetWebExportSetup(...) (page 195)
4.2.3.13. Reserved (page 196)
Application.GenerateManual(...) (page 196)
Application.SwitchKooplet(...) (page 196)
Application.IsKoopletMode(...) (page 196)
Application.UpdateScriptMenu(...) (page 196)
Application.GenerateWebFolder(...) (page 196)
4.2.3.14. Settings (page 196)
Application.ScriptBackGround(...) (page 196)
Application.GetCapability(...) (page 197)
4.2.3.15. Sound Base (page 197)
Application.GetSoundName(...) (page 197)
Application.IsSoundAlias(...) (page 197)

4.3. Notation (page 198)

4.3.1. Score Object Type (page 198)

Note about protected scores (page 198)
4.3.1.2. Score global functions (page 198)
4.3.1.2.1. Creating (page 198)
4.3.1.2.2. Managing (page 199)
4.3.1.3. Score values (page 200)
4.3.1.3.1. Ambitus (page 200)
4.3.1.3.2. Browsing scores (page 200)
4.3.1.3.3. Chord grid computation (page 200)
4.3.1.3.4. Display Setup (page 200)
4.3.1.3.5. Engraver (page 201)
4.3.1.3.6. FreeObjects (page 202)
4.3.1.3.7. Information (page 202)
4.3.1.3.8. Managing (page 204)
4.3.1.3.9. Managing barlines and breaks (page 205)
4.3.1.3.10. Managing Instruments (page 205)
4.3.1.3.11. Managing Metronome (page 207)
4.3.1.3.12. Managing staves in score (page 208)
4.3.1.3.13. Page Setup (page 208)
4.3.1.3.14. Paper size setup (page 209)
4.3.1.3.15. Played bars (page 210)
4.3.1.3.16. Playing Modes (page 210)
4.3.1.3.17. Playing music (page 210)
4.3.1.3.18. Printing (page 210)
4.3.1.3.19. Score appearance (page 211)
4.3.1.3.20. Selection (page 219)
4.3.1.3.21. Staves group (page 219)
4.3.1.3.22. Targets (page 220)
4.3.1.3.23. Views (page 220)
4.3.1.3.24. Window appearance (page 221)
4.3.1.4. Score methods (page 222)
4.3.1.4.1. Background picture (page 222)
4.3.1.4.2. Adding a BarAction object (page 222)
4.3.1.4.3. Bar management (page 223)
4.3.1.4.4. Beaming (page 224)
4.3.1.4.5. Chord grid (page 224)
4.3.1.4.6. Converting (page 225)
4.3.1.4.7. Engraver (page 228)
4.3.1.4.8. Exporting (page 228)
4.3.1.4.9. FreeObject (page 234)
4.3.1.4.10. Global bar settings (page 234)
4.3.1.4.11. Information (page 234)
4.3.1.4.12. Local metrics (page 235)
4.3.1.4.13. Managing (page 235)
4.3.1.4.14. Managing instruments (page 235)
4.3.1.4.15. Managing Metronome (page 236)
4.3.1.4.16. Managing Slurs (page 236)
4.3.1.4.17. Played bars (page 236)
4.3.1.4.18. Playing music (page 237)
4.3.1.4.19. Positioning (page 238)
4.3.1.4.20. Printing (page 239)
4.3.1.4.21. Protection (page 240)
4.3.1.4.22. Saving (page 240)
4.3.1.4.23. Selection (page 240)
4.3.1.4.24. Staves (page 242)
4.3.1.4.25. Staves group (page 244)
4.3.1.4.26. Systems (page 244)
4.3.1.4.27. Tags (page 246)
4.3.1.4.28. Targets (page 247)
4.3.1.4.29. Tempo (page 248)
4.3.1.4.30. Undo/redo (page 248)
4.3.1.4.31. Views (page 249)
4.3.1.4.32. Window (page 250)

4.3.2. Staff Object Type (page 252)

4.3.2.1. Staff values (page 252)
4.3.2.1.1. Browsing staves (page 252)
4.3.2.1.2. Staff title (page 252)
4.3.2.1.3. Bar numbering (page 254)
4.3.2.1.4. Digital Track (page 255)
4.3.2.1.5. Display (page 255)
4.3.2.1.6. Lyrics (page 256)
4.3.2.1.7. Managing chord diagrams (page 257)
4.3.2.1.8. Managing chord line (page 258)
4.3.2.1.9. Managing Clef, and Key signature changes (page 260)
4.3.2.1.10. Managing Dynamics/Tempo marks (page 261)
4.3.2.1.11. Managing FX processors (page 261)
4.3.2.1.12. Managing parameter curves (page 262)
4.3.2.1.13. Managing Pictures (page 262)
4.3.2.1.14. Managing Rules (page 263)
4.3.2.1.15. Managing staff lines and barlines (page 263)
4.3.2.1.16. Managing symbols (page 264)
4.3.2.1.17. Managing Text elements (page 266)
4.3.2.1.18. Managing Virtual Singer voice (page 267)
4.3.2.1.19. Miscellaneous (page 267)
4.3.2.1.20. Multi-voices (page 268)
4.3.2.1.21. Selection (page 268)
4.3.2.1.22. Spoken Human Voice (page 269)
4.3.2.1.23. Staff Attributes (page 269)
4.3.2.1.24. Staff metrics (page 270)
4.3.2.2. Staff methods (page 271)
4.3.2.2.1. Adding a BarSetting object (page 271)
4.3.2.2.2. Adding a Curve object (page 271)
4.3.2.2.3. Beaming (page 272)
4.3.2.2.4. Deleting a staff (page 272)
4.3.2.2.5. Drum Staff (page 273)
4.3.2.2.6. Edit (page 273)
4.3.2.2.7. Local metrics (page 274)
4.3.2.2.8. Managing chord diagrams (page 277)
4.3.2.2.9. Managing chord line (page 277)
4.3.2.2.10. Managing Dynamics/Tempo marks (page 279)
4.3.2.2.11. Managing FX processors (page 280)
4.3.2.2.12. Managing lyric lines (page 280)
4.3.2.2.13. Managing lyrics (page 281)
4.3.2.2.14. Managing Pictures (page 287)
4.3.2.2.15. Managing rules (page 289)
4.3.2.2.16. Managing Staves (page 289)
4.3.2.2.17. Managing symbols (page 290)
4.3.2.2.18. Managing texts (page 293)
4.3.2.2.19. Managing transposing instruments (page 293)
4.3.2.2.20. Merging Staves (page 294)
4.3.2.2.21. Miscellaneous (page 294)
4.3.2.2.22. Page setup (page 294)
4.3.2.2.23. Playing notes (page 295)
4.3.2.2.24. Tablature (page 295)
4.3.2.2.25. Tags (page 296)
4.3.2.2.26. Virtual Singer (page 296)

4.3.3. Symbol Object Type (page 298)

Note pitch (page 298)
Symbol duration (page 298)
Time position (page 298)
Absolute pixel position (page 298)
4.3.3.5. Symbol global functions (page 298)
4.3.3.5.1. Converting (page 298)
4.3.3.6. Symbol values (page 299)
4.3.3.6.1. Browsing symbols (page 299)
4.3.3.6.2. Managing note pitch (page 300)
4.3.3.6.3. Managing time position (page 301)
4.3.3.6.4. Managing ornaments (page 301)
4.3.3.6.5. Managing Grace notes (page 302)
4.3.3.6.6. Managing slurs (page 303)
4.3.3.6.7. Managing special beams (page 303)
4.3.3.6.8. Tablature (page 303)
4.3.3.6.9. Note miscellaneous parameters (page 305)
4.3.3.6.10. Accidental (page 308)
4.3.3.6.11. Managing display (page 309)
4.3.3.7. Symbol methods (page 312)
4.3.3.7.1. Appearance (page 312)
4.3.3.7.2. Deleting symbols (page 312)
4.3.3.7.3. Echo (page 313)
4.3.3.7.4. Managing Grace notes (page 313)
4.3.3.7.5. Managing ornaments (page 314)
4.3.3.7.6. Managing Slurs (page 314)
4.3.3.7.7. Managing special beams (page 315)
4.3.3.7.8. Managing symbol settings (page 316)

4.3.3.7.9. Managing tied notes (page 316)
4.3.3.7.10. Rules (page 316)

4.3.4. GlobalBarSetting Object Type (page 318)

4.3.4.1. GlobalBarSetting values (page 318)
4.3.4.1.1. Time Signature (page 318)
4.3.4.1.2. Beaming defaults (page 319)
4.3.4.1.3. Bar scale and display size (page 319)
4.3.4.1.4. Bar numbering settings (page 319)
4.3.4.1.5. Miscellaneous display settings (page 320)
4.3.4.2. GlobalBarSetting methods (page 320)
4.3.4.2.1. Managing (page 320)

4.3.5. BarAction Object Type (page 322)

4.3.5.1. BarAction values (page 322)
4.3.5.1.1. Browsing BarActions (page 322)
4.3.5.1.2. BarActions miscellaneous values (page 322)
4.3.5.1.3. Barlines (page 324)
4.3.5.1.4. Break symbol action (page 325)
4.3.5.1.5. Break symbol action triggering (page 326)
4.3.5.2. BarAction methods (page 326)
4.3.5.2.1. Deleting a BarAction (page 326)
4.3.5.2.2. Managing (page 327)

4.3.6. StavesGroup Object Type (page 328)

4.3.6.1. StavesGroup values (page 328)
4.3.6.1.1. Browsing Staves groups (page 328)
4.3.6.1.2. Staves group title (page 329)
4.3.6.1.3. Staves groups general parameters (page 330)
4.3.6.2. StavesGroup methods (page 331)
4.3.6.2.1. Managing (page 331)

4.3.7. BarSetting Object Type (page 332)

4.3.7.1. BarSetting values (page 332)
4.3.7.1.1. Browsing BarSettings (page 332)
4.3.7.1.2. BarSetting details (page 332)
4.3.7.1.3. BarSettings miscellaneous values (page 333)
4.3.7.1.4. BarSettings visibility (page 333)
4.3.7.1.5. Managing ornaments (page 333)
4.3.7.2. BarSetting methods (page 334)
4.3.7.2.1. Deleting a BarSetting (page 334)
4.3.7.2.2. Managing (page 334)
4.3.7.2.3. Managing ornaments (page 334)

4.3.8. Clef Object Type (page 336)

4.3.8.1. Clef values (page 336)
4.3.8.1.1. Clef settings (page 336)
4.3.8.1.2. Clef display values (page 336)
4.3.8.1.3. Staff display (page 336)
4.3.8.2. Clef methods (page 336)
4.3.8.2.1. Changing a clef and transposing (page 336)

4.3.9. KeySignature Object Type (page 338)

4.3.9.1. KeySignature values (page 338)
4.3.9.1.1. KeySignature settings (page 338)
4.3.9.2. KeySignature methods (page 338)
4.3.9.2.1. Changing key signature and transposing (page 338)
4.3.9.2.2. Custom key signature (page 339)

4.3.10. Ornament Object Type (page 341)

4.3.10.1. Ornament values (page 341)
4.3.10.1.1. Browsing Ornaments (page 341)
4.3.10.1.2. Ornament general attributes (page 341)
4.3.10.1.3. Ornament display settings (page 342)
4.3.10.1.4. Ornament play settings (page 343)
4.3.10.1.5. Ornament miscellaneous values (page 343)
4.3.10.1.6. Picture (page 343)
4.3.10.1.7. Slide (page 343)
4.3.10.2. Ornament methods (page 343)
4.3.10.2.1. Deleting an ornament (page 343)
4.3.10.2.2. Managing Pictures (page 344)
4.3.10.2.3. Ornament play settings (page 346)
4.3.10.2.4. Position (page 348)

4.3.11. GraceNote Object Type (page 349)

4.3.11.1. GraceNote values (page 349)
4.3.11.1.1. Browsing grace notes (page 349)
4.3.11.1.2. Grace note attributes (page 349)
4.3.11.2. GraceNote methods (page 350)
4.3.11.2.1. Deleting a grace note (page 350)

4.3.12. Dynamic Object Type (page 351)

4.3.12.1. Dynamic values (page 351)
4.3.12.1.1. Browsing Dynamics (page 351)
4.3.12.1.2. Dynamics general parameters (page 352)
4.3.12.1.3. Dynamics display settings (page 352)
4.3.12.1.4. Dynamics play settings (page 354)
4.3.12.1.5. Tempo-specific settings (page 355)
4.3.12.2. Dynamic methods (page 356)
4.3.12.2.1. Conditionnal (page 356)
4.3.12.2.2. Deleting a dynamic (page 356)
4.3.12.2.3. Editing a dynamic (page 356)
4.3.12.2.4. Managing (page 357)

4.3.13. Picture Object Type (page 358)

4.3.13.1. Picture values (page 358)
4.3.13.1.1. Browsing Pictures (page 358)
4.3.13.1.2. General parameters (page 358)
4.3.13.1.3. Position and size (page 359)
4.3.13.1.4. Display (page 360)
4.3.13.1.5. Tags (page 361)
4.3.13.2. Picture methods (page 361)
4.3.13.2.1. Deleting a Picture (page 361)
4.3.13.2.2. Drawing a Picture (page 361)
4.3.13.2.3. Selecting a Picture (page 361)

4.3.14. Diagram Object Type (page 363)

4.3.14.1. Diagram values (page 364)
4.3.14.1.1. Browsing Diagrams (page 364)
4.3.14.1.2. Diagrams general parameters (page 364)
4.3.14.1.3. Diagrams fingering (page 364)
4.3.14.2. Diagram methods (page 365)
4.3.14.2.1. Deleting a Diagram (page 365)
4.3.14.2.2. Editing a Diagram (page 365)
4.3.14.2.3. Fingering (page 365)
4.3.14.2.4. Miscellaneous (page 366)

4.3.15. Tablature Object Type (page 368)

4.3.15.1. Tablature values (page 368)
4.3.15.1.1. Appearance (page 368)
4.3.15.1.2. Attribute (page 370)
4.3.15.1.3. Computing (page 371)
4.3.15.1.4. Instruments (page 371)
4.3.15.2. Tablature methods (page 372)
4.3.15.2.1. Activating (page 372)
4.3.15.2.2. Computing (page 372)
4.3.15.2.3. Local metrics (page 372)
4.3.15.2.4. Miscellaneous (page 373)

4.3.16. FrettedTab Object Type (page 374)

4.3.16.1. FrettedTab values (page 374)
4.3.16.1.1. Computing (page 374)
4.3.16.1.2. Definition (page 374)
4.3.16.1.3. Display (page 375)
4.3.16.2. FrettedTab methods (page 375)
4.3.16.2.1. Strings (page 375)

4.3.17. HarmonicaTab Object Type (page 376)

4.3.17.1. HarmonicaTab values (page 376)
4.3.17.1.1. Computing (page 376)
4.3.17.1.2. Definition (page 376)
4.3.17.1.3. Display (page 376)
4.3.17.2. HarmonicaTab methods (page 377)
4.3.17.2.1. Holes (page 377)

4.3.18. AccordionTab Object Type (page 379)

4.3.18.1. AccordionTab values (page 379)
4.3.18.1.1. Definition (page 379)
4.3.18.1.2. Display (page 379)
4.3.18.2. AccordionTab methods (page 380)
4.3.18.2.1. Left Hand (page 380)

4.3.18.2.2. Right Hand (page 381)

4.3.19. StaffText Object Type (page 382)

4.3.19.1. StaffText values (page 382)
4.3.19.1.1. Browsing Staff Text elements (page 382)
4.3.19.1.2. Staff Text attributes (page 382)
4.3.19.2. StaffText methods (page 385)
4.3.19.2.1. Appearance (page 385)
4.3.19.2.2. Deleting a text (page 385)

4.3.20. Chord Object Type (page 386)

4.3.20.1. Chord values (page 386)
4.3.20.1.1. General (page 386)
4.3.20.1.2. Position (page 388)
4.3.20.2. Chord methods (page 389)
4.3.20.2.1. Editing (page 389)
4.3.20.2.2. Managing (page 389)

4.3.21. FreeObject Object Type (page 391)

4.3.21.1. FreeObject values (page 391)
4.3.21.1.1. Browsing FreeObjects (page 391)
4.3.21.1.2. General values (page 391)
4.3.21.1.3. Related files (page 392)
4.3.21.1.4. Position (page 392)
4.3.21.1.5. Selecting (page 393)
4.3.21.1.6. Specific data (page 393)
4.3.21.1.7. Display (page 394)
4.3.21.1.8. Tags (page 394)
4.3.21.2. FreeObject methods (page 394)
4.3.21.2.1. Deleting (page 394)
4.3.21.2.2. Managing Pictures (page 394)

4.3.22. StorageArea Object Type (page 396)

4.3.22.1. StorageArea methods (page 396)
4.3.22.1.1. Managing (page 396)

4.3.23. Target Object Type (page 398)

4.3.23.1. Target values (page 398)
4.3.23.1.1. Browsing targets (page 398)
4.3.23.1.2. Target miscellaneous values (page 398)
4.3.23.2. Target methods (page 399)
4.3.23.2.1. Deleting a Target (page 399)
4.3.23.2.2. Managing (page 399)
4.3.23.2.3. Play music (page 399)

4.4. Display (page 400)

4.4.1. View Object Type (page 400)

4.4.1.1. View values (page 400)
4.4.1.1.1. Browsing Views (page 400)
4.4.1.1.2. View settings (page 401)
4.4.1.2. View methods (page 402)
4.4.1.2.1. Deleting/Sorting views (page 402)
4.4.1.2.2. View settings (page 402)

4.4.2. EngraverSetup Object Type (page 403)

4.4.2.1. EngraverSetup global functions (page 403)
4.4.2.1.1. Managing (page 403)
4.4.2.1.2. Editing (page 403)
4.4.2.2. EngraverSetup values (page 403)
4.4.2.2.1. Bars (page 403)
4.4.2.2.2. General (page 404)
4.4.2.2.3. Lyrics (page 404)
4.4.2.2.4. Notes (page 404)

4.4.3. PageSetup Object Type (page 406)

4.4.3.1. PageSetup global functions (page 406)
4.4.3.1.1. Managing (page 406)
4.4.3.1.2. Editing (page 406)
4.4.3.2. PageSetup values (page 406)
4.4.3.2.1. General (page 407)
4.4.3.2.2. Margin (page 407)
4.4.3.2.3. Special (page 407)
4.4.3.2.4. Staves group (page 408)

4.4.4. PageHeader Object Type (page 410)

4.4.4.1. PageHeader global functions (page 410)
4.4.4.1.1. Managing (page 410)
4.4.4.1.2. Editing (page 410)
4.4.4.2. PageHeader values (page 411)
4.4.4.2.1. General (page 411)
4.4.4.2.2. Left definition (page 411)
4.4.4.2.3. Margins (page 411)
4.4.4.2.4. Middle definition (page 412)
4.4.4.2.5. Right definition (page 412)

4.4.5. Comment Object Type (page 413)

4.4.5.1. Comment values (page 413)
4.4.5.1.1. Working with Comments (page 413)

4.4.6. TextStyle Object Type (page 414)

4.4.6.1. TextStyle global functions (page 414)
4.4.6.1.1. Creating (page 414)
4.4.6.1.2. Asking the user (page 414)
4.4.6.1.3. Working on TextStyle (page 414)
4.4.6.1.4. Disposing TextStyle (page 414)
4.4.6.2. TextStyle values (page 414)
4.4.6.2.1. Working on text style (page 414)

4.5. Music (page 416)

4.5.1. Rule Object Type (page 416)

4.5.1.1. Rule values (page 416)
4.5.1.1.1. Browsing rules (page 416)
4.5.1.1.2. Rule display settings (page 416)
4.5.1.1.3. Rule matching criteria (page 417)
4.5.1.1.4. Applied rule parameters (page 420)
4.5.1.1.5. Managing applied grace notes (page 423)
4.5.1.1.6. Managing applied ornaments (page 424)
4.5.1.1.7. Symbols (page 424)
4.5.1.2. Rule methods (page 425)
4.5.1.2.1. Deleting a rule (page 425)
4.5.1.2.2. Managing Applied grace notes (page 425)
4.5.1.2.3. Managing applied grace notes (page 426)
4.5.1.2.4. Managing applied ornaments (page 426)

4.5.2. Instrument Object Type (page 428)

4.5.2.1. Instrument global functions (page 428)
4.5.2.1.1. Managing Sound IDs (page 428)
4.5.2.2. Instrument values (page 429)
4.5.2.2.1. Browsing Instruments (page 429)
4.5.2.2.2. Instrument general properties (page 430)
4.5.2.2.3. Instrument Layer Control (page 432)
4.5.2.2.4. Instrument MIDI settings (page 434)
4.5.2.2.5. Instrument Velocity filtering (page 435)
4.5.2.2.6. Instrument's Digital data (page 436)
4.5.2.2.7. Instrument's Digital output settings (page 436)
4.5.2.2.8. Virtual Instrument (page 437)
4.5.2.3. Instrument methods (page 438)
4.5.2.3.1. Managing (page 438)
4.5.2.3.2. Managing digital sounds (page 438)
4.5.2.3.3. Managing instruments (page 438)
4.5.2.3.4. Managing Instruments (page 439)

4.5.3. VelocityFilter Object Type (page 440)

4.5.3.1. VelocityFilter values (page 440)
4.5.3.1.1. Setup (page 440)

4.5.4. DigitalSound Object Type (page 441)

4.5.4.1. DigitalSound values (page 441)
4.5.4.1.1. Browsing digital sounds (page 441)
4.5.4.1.2. Digital sound envelope (page 441)
4.5.4.1.3. Digital sound format (page 441)
4.5.4.1.4. Digital sound miscellaneous settings (page 442)
4.5.4.1.5. Digital sound data (page 443)
4.5.4.2. DigitalSound methods (page 443)
4.5.4.2.1. Managing digital sounds (page 443)

4.5.5. DigitalData Object Type (page 445)

4.5.5.1. DigitalData values (page 445)
4.5.5.1.1. Managing digital data (page 445)
4.5.5.2. DigitalData methods (page 446)
4.5.5.2.1. Managing digital data (page 446)

4.5.6. DigitalTrack Object Type (page 449)

4.5.6.1. DigitalTrack values (page 449)
4.5.6.1.1. Digital Data (page 449)
4.5.6.1.2. Digital track attributes (page 449)
4.5.6.1.3. Markers (page 449)
4.5.6.1.4. Selection (page 450)
4.5.6.2. DigitalTrack methods (page 450)
4.5.6.2.1. Managing Markers (page 450)

4.5.7. RawTable Object Type (page 451)

4.5.7.1. RawTable global functions (page 451)
4.5.7.1.1. Creating (page 451)
4.5.7.2. RawTable values (page 451)
4.5.7.2.1. Getting information about a raw table (page 451)
4.5.7.3. RawTable methods (page 452)
4.5.7.3.1. Disposing (page 452)
4.5.7.3.2. Fast operation on RawTable (page 452)
4.5.7.3.3. Spectrum operation on RawTable (page 453)
4.5.7.3.4. Spectrum operation on RawTables (page 455)

4.5.8. FXProcessor Object Type (page 458)

4.5.8.1. FXProcessor values (page 458)
4.5.8.1.1. Browsing FX Processors (page 458)
4.5.8.1.2. FX Processor display settings (page 459)
4.5.8.1.3. Resonator settings (page 461)
4.5.8.1.4. Distortion/Overdrive settings (page 463)
4.5.8.1.5. Flanger/Chorus settings (page 464)
4.5.8.1.6. Equalizer settings (page 464)
4.5.8.1.7. Panning settings (page 466)
4.5.8.1.8. Delay/Reverb settings (page 467)
4.5.8.2. FXProcessor methods (page 468)
4.5.8.2.1. Deleting an indexed FX processor (page 468)
4.5.8.2.2. Editing an FX processor (page 468)
4.5.8.2.3. Managing FX processor icon (page 468)

4.5.9. Curve Object Type (page 471)

4.5.9.1. Curve values (page 471)
4.5.9.1.1. Browsing Curves (page 471)
4.5.9.1.2. Curve settings (page 471)
4.5.9.1.3. Managing curve data (page 472)
4.5.9.2. Curve methods (page 472)
4.5.9.2.1. Deleting a Curve (page 473)
4.5.9.2.2. Managing curve values (page 473)

4.5.10. VSVoice Object Type (page 476)

4.5.10.1. VSVoice values (page 476)
4.5.10.1.1. Placement on stage (page 476)
4.5.10.1.2. Attribute (page 476)
4.5.10.1.3. General Setting (page 477)
4.5.10.1.4. Timbre (page 477)
4.5.10.1.5. Glotal source (page 478)
4.5.10.1.6. Interpretation (page 478)
4.5.10.1.7. Volume (page 478)
4.5.10.1.8. Pitch Settings (page 479)
4.5.10.1.9. Vibrato (page 480)
4.5.10.1.10. Advanced (page 481)
4.5.10.2. VSVoice methods (page 482)
4.5.10.2.1. Setting voice language to the most probable one (page 482)

4.5.11. VirtualInstrument Object Type (page 483)

4.5.11.1. VirtualInstrument values (page 483)
4.5.11.1.1. Performer: Finger sliding (left hand) (page 483)
4.5.11.1.2. Performer: Golpe (page 485)
4.5.11.1.3. Performer: Microphone Volume (page 487)
4.5.11.1.4. Performer: Muting (page 488)
4.5.11.1.5. Performer: Slapping (page 489)
4.5.11.1.6. Performer: Strumming (right hand) (page 489)
4.5.11.1.7. Physical: Body (page 491)
4.5.11.1.8. Physical: Microphones (page 491)
4.5.11.1.9. Physical: Strings (page 491)
4.5.11.1.10. Physical: Sympathy resonance (page 492)

4.5.11.1.11. Unsorted (page 492)
4.5.11.2. VirtualInstrument methods (page 492)
4.5.11.2.1. I/O (page 492)
4.5.11.2.2. Unsorted (page 493)

4.6. Interface (page 494)

4.6.1. Dialog Object Type (page 494)

4.6.1.1. Dialog global functions (page 495)
4.6.1.1.1. Predefined (page 495)
4.6.1.1.2. Creating (page 496)
4.6.1.2. Dialog values (page 497)
4.6.1.2.1. Display (page 497)
4.6.1.2.2. Help (page 497)
4.6.1.2.3. Items (page 497)
4.6.1.2.4. Path (page 497)
4.6.1.2.5. Private data (page 497)
4.6.1.2.6. Window appearance (page 497)
4.6.1.3. Dialog methods (page 498)
4.6.1.3.1. Items (page 498)
4.6.1.3.2. Managing (page 503)
4.6.1.3.3. Mouse (page 504)

4.6.2. DialogItem Object Type (page 506)

4.6.2.1. DialogItem values (page 506)
4.6.2.1.1. Browsing dialog items (page 506)
4.6.2.1.2. Dialog item general parameters (page 506)
4.6.2.1.3. Managing extra values (page 510)
4.6.2.1.4. Help (page 510)
4.6.2.1.5. Radio button / Group parameters (page 511)
4.6.2.1.6. Text item (page 511)
4.6.2.1.7. Display (page 511)
4.6.2.2. DialogItem methods (page 511)
4.6.2.2.1. Graphic (page 511)
4.6.2.2.2. Managing extra values (page 512)
4.6.2.2.3. Managing items (page 515)

4.6.3. Graph Object Type (page 516)

Drawing in an user item (page 516)
Drawing in a Dialog (page 516)
Drawing in a TextWindow (page 516)
Creating a Picture object (page 516)
Creating a Surface object (page 517)
Update consideration (page 517)
4.6.3.7. Graph values (page 519)
4.6.3.7.1. Graph Metrics (page 519)
4.6.3.8. Graph methods (page 519)
4.6.3.8.1. Pen (page 519)
4.6.3.8.2. Background (page 520)
4.6.3.8.3. Color (page 521)
4.6.3.8.4. Line (page 522)
4.6.3.8.5. Rectangles (page 523)
4.6.3.8.6. Rounded Rectangles (page 524)
4.6.3.8.7. Ovals (page 524)
4.6.3.8.8. Arcs (page 525)
4.6.3.8.9. Text (page 526)
4.6.3.8.10. Picture (page 529)
4.6.3.8.11. Polygon (page 529)
4.6.3.8.12. Musical objects (page 532)
4.6.3.8.13. Clipping (page 532)
4.6.3.8.14. Palette button (page 533)
4.6.3.8.15. Pict file (page 533)
4.6.3.8.16. Sizes (page 533)
4.6.3.8.17. Miscellaneous (page 534)

4.6.4. Surface Object Type (page 535)

4.6.4.1. Surface values (page 535)
4.6.4.1.1. Surface Metrics (page 535)
4.6.4.2. Surface methods (page 536)
4.6.4.2.1. Create/Dispose a Surface (page 536)
4.6.4.2.2. Surface blurring and drawing (page 538)
4.6.4.2.3. Surface I/O (page 540)
4.6.4.2.4. Surface Opacity layer (page 541)
4.6.4.2.5. Surface pixel color change (page 547)
4.6.4.2.6. Surface pixels (page 551)
4.6.4.2.7. Surface resizing and deformation (page 552)
4.6.4.2.8. Surface shadowing (page 558)

4.6.5. TextWindow Object Type (page 560)

4.6.5.1. TextWindow global functions (page 560)
4.6.5.1.1. Creating (page 560)
4.6.5.2. TextWindow values (page 560)
4.6.5.2.1. Display (page 560)
4.6.5.2.2. Window appearance (page 561)
4.6.5.3. TextWindow methods (page 562)
4.6.5.3.1. Managing (page 562)
4.6.5.3.2. Text (page 563)
4.6.5.3.3. Printing (page 563)
4.6.5.3.4. Saving (page 563)

4.6.6. Palette Object Type (page 565)

4.6.6.1. Palette values (page 570)
4.6.6.1.1. Attributes (page 570)
4.6.6.1.2. Browsing (page 571)
4.6.6.2. Palette methods (page 571)
4.6.6.2.1. General (page 571)
4.6.6.2.2. Appearance (page 572)
4.6.6.2.3. Browsing (page 572)
4.6.6.2.4. Managing (page 572)
4.6.6.2.5. Window Position (page 573)
4.6.6.2.6. Window size (page 573)
4.6.6.2.7. Actions (page 574)

4.6.7. ProgressBar Object Type (page 576)

4.6.7.1. ProgressBar methods (page 576)
4.6.7.1.1. Creating (page 576)
4.6.7.1.2. Current status (page 576)
4.6.7.1.3. Disposing (page 577)

4.6.8. Text Object Type (page 578)

4.6.8.1. Text values (page 578)
4.6.8.1.1. Data (page 578)
4.6.8.1.2. Selection (page 578)
4.6.8.1.3. Style (page 579)
4.6.8.2. Text methods (page 579)
4.6.8.2.1. Browsing (page 579)
4.6.8.2.2. Deleting (page 579)
4.6.8.2.3. Inserting character (page 579)
4.6.8.2.4. Scrap book (page 580)
4.6.8.2.5. Style (page 581)
4.6.8.2.6. Selecting (page 581)

4.6.9. Jukebox Object Type (page 582)

4.6.9.1. Jukebox values (page 582)
4.6.9.1.1. Attributes (page 582)
4.6.9.1.2. Playing modes (page 582)
4.6.9.1.3. Browsing (page 583)
4.6.9.1.4. General informations (page 583)
4.6.9.1.5. Jukebox file (page 583)
4.6.9.1.6. Window appearance (page 583)
4.6.9.2. Jukebox methods (page 584)
4.6.9.2.1. Managing Jukebox (page 584)
4.6.9.2.2. Managing entries (page 585)
4.6.9.2.3. Managing Columns (page 586)
4.6.9.2.4. Playing (page 588)
4.6.9.2.5. Browsing (page 588)
4.6.9.2.6. Export (page 588)

4.7. Devices (page 590)

4.7.1. File Object Type (page 590)

Related methods (page 590)
Security (page 590)
Note about file names (page 590)
Note about current path (page 590)
Note about path names (page 591)
Note about character encoding (page 591)
4.7.1.7. File global functions (page 591)
4.7.1.7.1. Selecting (page 591)
4.7.1.7.2. Information (page 592)
4.7.1.7.3. Working on name (page 593)
4.7.1.7.4. Folders (page 594)
4.7.1.7.5. Loading (page 595)
4.7.1.7.6. Paths (page 596)

4.7.1.7.7. Files (page 599)
4.7.1.7.8. Private (page 603)
4.7.1.8. File values (page 603)
4.7.1.8.1. Encoding & charsets (page 603)
4.7.1.8.2. Position (page 604)
4.7.1.9. File methods (page 604)
4.7.1.9.1. Disposing (page 605)
4.7.1.9.2. Files (page 605)
4.7.1.9.3. Reading/Writing (page 605)
4.7.1.9.4. Position (page 609)

4.7.2. Midi Object Type (page 610)

4.7.2.1. Midi methods (page 610)
4.7.2.1.1. Setup (page 610)
4.7.2.1.2. Midi In (page 610)
4.7.2.1.3. Midi Out (page 611)

4.7.3. Internet Object Type (page 614)

4.7.3.1. Internet methods (page 614)
4.7.3.1.1. Email (page 614)
4.7.3.1.2. Internet files (page 614)
4.7.3.1.3. Web Pages (page 614)

4.7.4. Movie Object Type (page 615)

4.7.4.1. Movie global functions (page 616)
4.7.4.1.1. Files (page 616)
4.7.4.1.2. Sound (page 616)
4.7.4.2. Movie values (page 617)
4.7.4.2.1. Graphic (page 617)
4.7.4.2.2. Managing (page 617)
4.7.4.2.3. Sound (page 617)
4.7.4.2.4. Time (page 617)
4.7.4.3. Movie methods (page 618)
4.7.4.3.1. Graphic (page 618)
4.7.4.3.2. Managing (page 619)
4.7.4.3.3. Sound (page 620)
4.7.4.3.4. Texts (page 622)

5. Appendices (page 623)

5.1. Keywords abstract (page 624)
5.2. Not Yet Done (page 646)

5.2.1. Overview (page 646)
5.2.2. Movie object (page 646)
5.2.3. Score object (page 646)
5.2.4. Staff object (page 646)

5.3. Possible features (page 647)

5.3.1. Overview (page 647)
5.3.2. Custom staves (page 647)
5.3.3. Search tool (page 647)

Example scripts provided in this book :

This book includes interactive script samples that can be run directly by clicking an icon in the manual.
You can also open the sample into a new script window (Shift+click the icon) then step the script (icon) to understand what it does.
Here is the list of all runable scripts :

Using this hypertext manual

1-Hello world sample. (page 28)

Crashes

2-Script with error. (page 77)

The Language

3-Lua variable names are case sensitive. (page 79)
4-Comments. (page 79)
5-Automatic conversion of strings. (page 80)
6-Returning multiple values. (page 80)
7-Multiple assignment. (page 81)
8-"while" control structure. (page 81)
9-"until" control structure. (page 81)
10-"if-else" control structure. (page 82)
11-"for" statement. (page 83)
12-Local variables (page 84)
13-Using tables in Lua (page 87)

Standard Libraries

14-Defining an alternate way for displaying error messages (page 91)

Notes from the Myriad Team

15-Divide by zero error. (page 98)
16-Boolean comparison. (page 98)
17-Bidimensional array. (page 99)

MSSounds file Overview

18-Displaying all the sound IDs (page 110)

Using MSSounds definitions

19-Displaying the standard name of a sound (page 110)
20-Displaying the standard ID of a sound (page 110)

Comment utiliser le manuel en hypertexte

21-Example MyrScript "Bonjour". (page 116)

Plantages

22-Script with error. (page 147)

Le Langage

23-Lua : les noms de variable tiennent compte de la casse. (page 149)
24-Comments. (page 149)
25-Automatic conversion automatique de chaînes. (page 150)
26-Returning retour de valeurs multiples. (page 150)
27-Multiple assignation multiple. (page 151)
28-"while" control structure. (page 151)
29-"until" control structure. (page 151)
30-"if-else" control structure. (page 152)
31-"for" statement. (page 153)
32-Local variables (page 154)
33-Using Utilisation des tables en Lua (page 157)

Bibliothèques Standard

34-Defining définir une autre manière d'afficher les messages d'erreur (page 161)

Notes de l'équipe Myriad

35-Divide by zero error. (page 169)
36-Boolean comparison. (page 169)

Application global functions

37-Converting strings encoding. (page 176)
38-Using object local copy to display its fields. (page 177)
39-Transfering staff parameters into another staff. (page 177)
40-Changing the staff title style. (page 178)
41-Extracting styles from a styled string. (page 179)

Application values

42-Displaying current voice name. (page 179)
43-Opening a palette. (page 179)
44-Displaying the current application language. (page 180)
45-Displaying the current version of MyrScript. (page 180)
46-Displaying application name and version. (page 180)
47-Displaying computer model. (page 180)
48-Displaying the current SoundBase version. (page 181)
49-Displaying the current SoundBase name. (page 181)
50-Checking if Virtual Singer is installed. (page 182)
51-Displaying the Virtual Singer version. (page 182)

Application methods

52-Computing complex chord name. (page 183)
53-Converting chord name. (page 183)
54-Converting pitches to chord name. (page 184)
55-Displayng live digital input volume (page 185)
56-Displaying current date. (page 186)
57-Displaying modification date of a file. (page 186)
58-Converting current date to system time. (page 186)
59-Displaying elapsed time since the computer started. (page 186)
60-Waiting for a delay. (page 187)
61-Displaying a duration. (page 187)
62-Scanning computer keyboard and displaying key value. (page 187)
63-Checking if a particular key is pressed. (page 188)
64-Changing mouse cursor shape. (page 188)
65-Displaying info about all languages supported by Virtual Singer. (page 190)
66-Say a sentence. (Manual wait) (page 190)
67-Say a sentence and wait (Internal wait). (page 190)
68-Create a file with a spoken sentence. (page 191)
69-Activate a voice,and say a sentence. (page 191)
70-Get number of available human voices. (page 192)
71-Dump Information about all installed voices. (page 192)
72-Dump Informations about various voice. (page 193)
73-How to extract all voices from a criterion. (page 193)
74-Search a voice by name (page 194)
75-How to extract all voices by pattern on name (page 194)
76-Searching for english voices (page 194)
77-Searching for kids voices (page 194)
78-Dumping the language keyword list. (page 194)
79-Scanning Midi In in background. (page 197)

Score Object Type

80-Creating a new score. (page 198)
81-Creating a new invisible score. (page 199)
82-Creating a MyrScript file from a script to extract jingle. (page 199)
83-Creating a new score from a template. (page 199)
84-Locating front score. (page 200)
85-Reverting a score to the previous saved version. (page 204)
86-Displaying how many instruments are in a score. (page 205)
87-Displaying the name of an instrument. (page 206)
88-Displaying all sound names related to the instruments. (page 206)
89-Displaying instruments name in reverse order. (page 207)
90-Setting metronome active. (page 207)
91-Acceding to the instruments related to the Metronome. (page 207)
92-Displaying number of staves. (page 208)
93-Displaying staff names (page 208)
94-Browsing Views by index (page 221)
95-Removing a score background. (page 222)
96-Setting a picture file as score background. (page 222)
97-Adding a BarAction (page 223)
98-Deleting one bar at the beginning of the score. (page 224)
99-Inserting one bar at the beginning of the score. (page 224)
100-Extracting the chord at a time position. (page 225)
101-Dump all custom chord diagrams of front score (page 225)
102-Adding a engraver setup to the default engravers. (page 228)
103-Exporting the current score in a WAV file. (page 230)
104-Exporting the current score in a Myrweb file. (page 231)
105-Exporting the current score as a movie. (page 231)
106-Exporting all files in a folder as movie. (page 232)
107-Exporting the first page of the current score in BMP file format. (page 233)
108-Adding an instrument to a score. (page 236)
109-Pausing, unpausing the front score. (page 238)
110-Change the volume of the front score to the optimal volume. (page 238)

111-Save the current score asking for file name. (page 240)
112-Asking the user for choosing staves. (page 242)
113-Adding a staff (page 243)
114- Counting systems in pages. (page 245)
115- Counting staves in systems. (page 245)
116-Extracting bar range of each system. (page 245)
117-Extracting staves from systems. (page 246)
118-Extracting systems area. (page 246)
119-Converting bar number to page and system. (page 246)
120-Adding a Target (page 247)
121-Displaying the general tempo. (page 248)
122-Displaying the current tempo. (page 248)
123-Changing the general tempo. (page 248)
124-Getting currently selected View (page 249)
125-Selecting another View (page 249)
126-Creating a new View (page 250)

Staff Object Type

127-Displaying staff names (page 252)
128-Displaying staff names in reverse order (page 252)
129-Renaming staves (page 253)
130-Changing staff height. (page 270)
131-Adding a BarSetting (page 271)
132-Deleting a staff (page 273)
133-Dumping staves local heights (page 275)
134-Adding a Diagram in a staff (page 277)
135-Display chord diagram fingering of a chord line (page 279)
136-Dump all custom chord diagrams available for a staff (page 279)
137-Adding a tempo change in a staff (page 280)
138-Adding an FX processor in a staff (page 280)
139-Shift lyrics line on a score (page 281)
140-Displaying the lyrics. (page 282)
141-Replacing lyric texts by fragments. (page 283)
142-Extracting lyrics from written time position and line. (page 284)
143-Extracting lyrics from bar. (page 284)
144-Displaying the lyrics : fast version with array. (page 286)
145-Displaying Sampa notation. (page 286)
146-Displaying played Sampa. (page 287)
147-Adding a picture in a staff. (page 288)
148-Creating a picture with a script. (page 288)
149-Adding a rule to a staff (page 289)
150-Duplicating a staff (page 290)
151-Changing staff order in a score (page 290)
152-Getting symbols from their index in staff (page 292)
153-Deleting all symbols in a staff (page 293)
154-Adding a text object in a staff (page 293)
155-Displaying the page number for each bar. (page 295)
156-Relating a tablature to a staff. (page 295)
157-Adding a Virtual Singer voice to the default voices. (page 297)

Symbol Object Type

158- Converting a pitch in string. (page 298)
159- Converting a string in pitch. (page 299)
160-Converting pitch to frequency. (page 299)
161-Converting frequency to pitch. (page 299)
162-Browsing symbols. (page 300)
163-Exporting a chromatic harmonica tablature in Harping ! text format. (page 304)
164-Dumping the fingering of an accordion tablature. (page 305)
165-Changing note duration (page 308)
166-Deleting a symbol. (page 313)

GlobalBarSetting Object Type

167-Changing time signature. (page 318)
168-Copying GlobalBarSetting (page 321)

BarAction Object Type

169-Browsing BarActions (page 323)
170-Deleting BarActions (page 327)

StavesGroup Object Type

171-Dumping staves groups (page 328)

BarSetting Object Type

172-Browsing BarSettings (page 333)
173-Deleting BarSettings (page 334)
174-Add an ornament picture to second bar setting of first staff (page 335)

Clef Object Type

175-Changing clef and transposing (page 337)

KeySignature Object Type

176-Changing key signature and transposing (page 339)
177-Displaying accidentals from a key signature (page 339)
178-Setting custom accidentals in a key signature (page 340)

Ornament Object Type

179-Deletes all ornaments related to bar setting in the first staff. (page 344)
180-Adding a picture to an ornament. (page 345)
181-Creating a picture with a script and add to ornament. (page 345)

Picture Object Type

182-Select a picture (page 362)

Diagram Object Type

183-Extracting fingers from a chord diagram (page 366)
184-Convert all chord diagrams in a chord line to local (page 367)
185-Copy the first chord diagram of the chord grid to the first staff chord line (page 367)
186-Copy the first chord diagram of the global database to the score chord grid (page 367)

Tablature Object Type

187-Checking the tablature height. (page 369)
188-Relating a tablature to a staff. (page 372)

FrettedTab Object Type

189-Displaying the standard tuning of a guitar tablature. (page 375)

HarmonicaTab Object Type

190-Displaying the standard tuning of an harmonica. (page 378)

AccordionTab Object Type

191-Displaying the informations about an accordion, left hand. (page 380)
192-Displaying information about an accordion, right hand. (page 381)

Chord Object Type

193-Displaying the names of all chords in the chord grid. (page 386)
194-Changing a component of a chord. (page 388)
195-Displaying bar position, position in bar, time position of each chord of the chord grid. (page 389)
196-Editing a chord. (page 389)
197-Splitting chords in chord grid. (page 389)
198-Deleting chord in chord grid. (page 390)
199-Copying chords in the chord grid. (page 390)
200-Copying all chords of a bar. (page 390)

FreeObject Object Type

201-Deleting FreeObjects (page 394)
202-Creating a picture in a free object. (page 395)

StorageArea Object Type

203-Getting the number of items in a storage area. (page 396)
204-Enumerating items in a storage area. (page 396)
205-Storing, retrieving and deleting elements in a storage area. (page 397)

Target Object Type

206-Browsing myTarget (page 398)
207-Deleting Targets (page 399)

View Object Type

208-Browsing Views (ascending) (page 400)
209-Browsing Views (descending) (page 400)
210-Deleting a view (page 402)
211-Moving a view (page 402)
212-Copying view parameters (page 402)

EngraverSetup Object Type

213-Asking the user to define an engraver setup. (page 403)

PageSetup Object Type

214-Editing a page setup. (page 406)

PageHeader Object Type

215-Editing a page header setup. (page 411)

Rule Object Type

216-Display used rules (page 425)
217-Deleting rules. (page 425)

Instrument Object Type

218-Availability of instrument sounds in the current sound base. (page 428)
219-Changing sound IDs to their closest available value in the current sound base. (page 429)
220-Displaying the name of an instrument. (page 430)
221-Browsing instruments by index. (page 430)
222-Displaying ID of the sounds related to the instruments. (page 431)

FXProcessor Object Type

223-Adding an "old-telephone" FX processor (page 466)

Curve Object Type

224-Browsing curves (page 471)
225-Deleting Curves (page 473)
226-Getting curve values (page 473)
227-Adding curve points (page 474)
228-Deleting curve points (page 474)
229-Browsing curve points (page 475)

VSVoice Object Type

230-Checking if a Virtual Singer voice is related to a staff. (page 476)

Dialog Object Type

231-Creating a complete dialog from a script. (page 495)
232-Displaying an alert message. (page 496)
233-Displaying a message. (page 496)
234-Asking confirmation to the user. (page 496)

Graph Object Type

235-Drawing graphics in a text window. (page 518)
236-Displaying the RGB value for all color indexes. (page 521)
237-Displaying the color index of a RGB color. (page 521)
238-Displaying the color components of a RGB color. (page 521)
239-Displaying the RGB color from its color components. (page 522)
240-Checking if a font exist. (page 526)
241-Extracting the height of the current font. (page 527)
242-Using polygon methods. (page 530)
243-Displaying the screen size. (page 533)
244-Displaying the menu bar height. (page 534)

Surface Object Type

245- Drawing on a Surface through a Graph object (page 536)
246-Creating and disposing a surface object (page 537)
247-Saving the content of a dialog to a PNG file (page 537)
248-Duplicating a Surface object (page 538)
249-Blurring a Surface (page 538)
250-Drawing a transparent rectangle to a Surface (page 539)
251-Drawing a Surface within a Surface (page 540)
252-Loading a graphic file into a Surface object (page 540)
253-Saving a Surface (page 541)
254-Setting global opacity of a Surface (page 541)
255-Setting opacity of a Surface from a chosen color (page 542)
256-Setting opacity of a Surface from pixel luminance (page 542)
257-Altering Surface opacity by a constant (page 543)
258-Altering Surface opacity by a bidirectional gradient (page 544)
259-Blurring Surface opacity layer (page 544)
260-Blurring Surface opacity layer of non-transparent parts (page 545)
261-TRansferring Surface opacity layer into pixel brightness (page 545)
262-Inverting Surface opacity (page 546)
263-Combining Opacity Layers of two Surfaces (page 547)
264-Highlighting a Surface (page 547)
265-Making a Surface glow (page 548)
266-Adjusting Surface Brightness, Saturation and Contrast (page 548)

267-Adjusting Surface Hue, Saturation and Lightness (page 549)
268-Adjusting Surface Brightness curve (page 549)
269-Adjusting Surface Component curves (page 550)
270-Mixing Surface RGB Channels (page 551)
271-Using Surface.GetPixel and Surface.SetPixel to Distort a surface (page 551)
272-Adding margins around a surface (page 552)
273-Cropping a surface (page 553)
274-Rescaling a surface (page 553)
275-Resizing a surface (page 554)
276-Inverting a surface vertically (page 554)
277-Inverting a surface horizontally (page 555)
278-Rotating a Surface (page 555)
279-Skewing right a Surface (page 556)
280-Skewing down a Surface (page 557)
281-Creating perspective effect from a Surface (page 557)
282-Distorting a Surface (page 558)
283-Adding a shadow to a Surface (page 559)
284-Adding a inner shadow to a Surface (page 559)

TextWindow Object Type

285-Creating a new text window. (page 560)
286-Writing values in a text window. (page 560)
287-Creating a new text window only if it don't exist. (page 560)
288-Moving a text window on the screen. (page 561)
289-Creating, writing in and disposing a text window. (page 562)
290-Writing in an invisible text window then showing it. (page 562)
291-Hiding and showing a text window. (page 562)
292-Writing numbers in a text window. (page 563)
293-Changing the style of a text window. (page 563)
294-Printing the content of a text window. (page 563)

Palette Object Type

295-Opening the user's palette. (page 566)
296-Toggle a palette mode. (page 570)
297-Displaying all palette names. (page 571)
298-Checking if a palette is open. (page 571)
299-Checking if a palette is vertical. (page 571)
300-Displaying all palette names. (page 571)
301-Hiding all the palette windows. (page 571)
302-Showing all the palette windows. (page 572)
303-Inverting the play palette orientation. (page 572)
304-Locating the first palette in the palette list. (page 572)
305-Opening the menu palette. (page 572)
306-Getting a palette object from a palette name. (page 572)
307-Closing the menu palette. (page 573)
308-Closing the menu palette (alternate way). (page 573)
309-Changing the position of a palette. (page 573)
310-Getting the palette position. (page 573)
311-Changing the palette window size. (page 573)
312-Getting the palette size. (page 574)
313-Launching a music using a pseudo click (page 574)
314-Sending a message on the help window. (page 574)
315-Resetting all palette states. (page 575)

ProgressBar Object Type

316-Displaying a progress bar. (page 576)
317-Changing the text in a progress bar. (page 576)

Text Object Type

318-Inserting text in a TextWindow using low level Text object function. (page 578)
319-Copy/paste in a text related to a TextWindow. (page 580)
320-Changing the default font, size, face of a Text object (page 581)

Jukebox Object Type

321-Checking if the Jukebox list was modified. (page 582)
322-Displaying number of entries in the jukebox. (page 583)
323-Setting the Jukebox window to minimal size. (page 584)
324-Opening the Jukebox window. (page 584)
325-Creating a new empty jukebox list. (page 584)
326-Closing the Jukebox window. (page 585)
327-Saving the Jukebox list. (page 585)
328-Adding the front score to the jukebox. (page 585)
329-Displaying informations about files in jukebox list.. (page 586)
330-Displaying column identifier names. (page 587)
331-Sorting the jukebox list by date. (page 587)
332-Extracting all columns info. (page 587)
333-Setting jukebox columns size. (page 588)
334-Start playing the jukebox. (page 588)

335-Stop playing the jukebox. (page 588)

File Object Type

336-Asking for selecting a file and displaying its name. (page 591)
337-Using extension list to select a file. (page 592)
338-Asking the user to select the file name to save. (page 592)
339-Selecting a folder name. (page 592)
340-Displaying a file size. (page 592)
341-Displaying time of last change of a file. (page 593)
342-Splitting a complete path name. (page 593)
343-Adding an extension to a full path name. (page 593)
344-Browsing a folder and displaying file names. (page 594)
345-Browsing a folder and displaying file names. (page 595)
346-Creating a new folder in the Settings (page 595)
347-Opening a folder on the desktop. (page 595)
348-Loading a document. (page 596)
349-Launching an external program. (page 596)
350-Opening a file with an external program. (page 596)
351-Displaying the current path. (page 597)
352-Displaying path of the documentation folder. (page 597)
353-Displaying path of the temporary files folder. (page 597)
354-Displaying path of the preferences folder. (page 597)
355-Displaying path on the lyrics export folder. (page 598)
356-Displaying path on the user's music folder. (page 598)
357-Displaying the path on the engraver settings folder. (page 598)
358-Displaying the path on the voices system setting folder. (page 599)
359-Converting a string to a possible file name (page 599)
360-Converting a string to a possible file name (page 599)
361-Writing and reading data in a text file. (page 601)
362-Removing a file. (page 601)
363-Changing file name. (page 602)
364-Creating a temporary file name. (page 602)
365-Browsing a file. (page 604)
366-Copying a file in binary mode. (page 606)
367-Reading a file in binary mode. (page 607)
368- Dump a complete XML file. (page 609)

Midi Object Type

369-Setting up the Midi ports. (page 610)
370-Sending a Program Change to Midi Echo device (page 610)
371-Scanning the Midi in port and displaying the events. (page 611)

Internet Object Type

372-Sending a mail. (page 614)
373-Opening a web page address. (page 614)

Movie Object Type

374-Playing a movie in a dialog window. (page 616)
375-Creating a slideshow. (page 618)
376-Creating a slideshow with transition effects. (page 619)
377-Importing the sound track stored in a movie file (page 621)
378-Exporting music using system resources. (page 621)

1. General chapters

1.1. The developer manual

1.1.1. Overview

Harmony Assistant now opens its sources to fit your needs even better. A feature is lacking? Just write it...
The MyrScript language provides all you need to control the Harmony Assistant internal processing.
You will find here all necessary information to build your own scripts.
A first section explains the basics of the graphical interface.
A second section comments all objects types, values, methods and functions you can use from within your scripts.
You will find a pre-printed version of this manual (PDF file) here : http://www.myriad-online.com/en/docs/preprinted.htm

1.1.2. Using this hypertext manual

The index

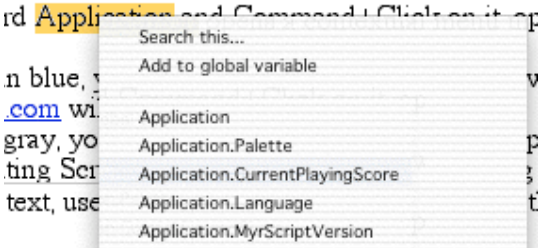
On the left of this window, you can see the complete index. Click an arrow : ▷, to open or close the matching section. Click a topic to jump to it. The currently displayed position is written in blue.
When you browse the manual you can use the Command+I keyboard command to close all opened sections in the index and open the current one.

Searching the manual

You can use the search tool (page 73) to search for a specific word or sentence.

Contextual help

You can Command+click on a word (or right click on PC) to open a contextual menu about this word, showing the related topics. You can also select a group of word then Command+click the selection range. Selecting one of these topics jumps to it.
For example, Command+Click on the following word: **Application**. It opens a contextual menu that shows all the " Application"-related keywords.



Internet link

When a word is written underlined and in blue, you can click it to open a Web page using your current Web browser. For example : http://www.myriad-online.com will lead to our main web site.

Jump to target

When a word is written underlined and in gray, you can click it to jump to the word explanation.
For example : Click here to see the "Editing Script" section (page 50) will jump to the "Editing Script" section.
To go back to a previous location in the text, use the icon in the placard (page 52) to open a pop-up menu that sho the history of previously read locations.
To get back to the previous location, hit Command+Backspace.

Interactives Examples

Some MyrScript examples provided in this manual can be run directly. Just click the blue arrow : .
For example, if you click the blue arrow shown below, the script will run and display "Hello world, welcome to MyrScript." in the log window (page 75).

```
print("Hello world, welcome to MyrScript.")
```

Note: If you click the same blue arrow with Shift key down, the script won't be launched, but loaded into a new script edit window.

1.1.3. Printing the manual

To print this manual use Command+P (Command key is Ctrl on PC, Apple key on Macintosh)

1.1.4. Platform dependence

A script is **platform independent**. The same script will run without change on Windows, Mac OS or UNIX (MacOS X).
A very few features are reserved to some platforms. For example the WMF graphic file export is only available on Windows.

These special features are tagged with the icon , and .

1.1.5. Notes to MyrScript developer

Thanks for writing a script. Please keep in mind a script can be useful to many people in many different countries.
So, if you want to share your script, do not forgot to make your script readable : use English as primary language then call the MyrScript features to test what language is currently selected by the user.
You can then display the graphical interface of your script in the required language. Try to provide at least an English version of your script.
Much more, do not forget to insert many comments in your source file... If you need any help on a specific problem, the Myriad Team will require a well-commented script.

At last, when your script works well, and you want it to be shared with other users, send it to us using the icon for example.

Here are somes rules we recommend to follow.

· As any other kind of program, a script has to be made of many comments with a few instructions around. Comment meticulously your script source file.
· Use clear variable and function names.
· When declaring a variable, use the keyword *local* if the variable is local. This kind of variable is directly displayed in the variable window while debugging.
· Always save your script source file before running it.

1.2. MyrScript Versions

1.2.1. Version 0.1

November 2002/ March 2003
The first version of MyrScript. Development and Beta test.

1.2.2. Version 0.1.6

April 1 2003 (Harmony V 8.3.0)
First public version.

1.2.3. Version 0.1.7

April 2003 (Harmony V 8.3.1)

- Added Score.NumberOfPlayedBars (page 202) value
- Added FORMAT_OGG in MSDefine
- Added Staff.IsStemCenteredOnHead (page 266) value
- Added output value to Symbol.GetSlurSettings(...) (page 315) and input value to Symbol.SetSlurSettings(...) (page slur appearance
- Bug fixed in select text while editing scripts on Windows.
- New appearance for the "About Scripts" section

1.2.4. Version 0.1.8

April 2003 (Harmony V 8.3.3)

- Changing some methods in Palette: they work now by palette object instead of palette names.
- Added Palette.PseudoClick(...) (page 574)
- Added values Symbol.IsTupletSlur (page 310),Symbol.IsTupletDontDisplayNumber (page 310)
- Added Staff.MergeWithNext(...) (page 294) and Staff.Unmerge(...) (page 294)
- Full management of Jukebox (page 582) window.
- Chord (page 386) grid management.
- Fixed bad name: Score.IsAccompaniementsMatrixDisplayed (page 201)
- Chord conversions: Application.ConvertElementsToChordName(...) (page 183) and Application.ConvertChordNameToElements(...) (page 183)
- Ornament positionning: Ornament.AutoPosition(...) (page 348)
- Added ConvertFileExtensionToFileFormat(...) (page 594)

1.2.5. Version 0.1.9

May 2003 (Harmony V 8.3.5)

- New method to compute the normalized volume: Score.ComputeNormalizedVolume(...) (page 238)
- New parameter for digital export: normalized export.
- New method DigitalData.FFTBlock(...)
- New global functions FrequencyToPitch(...) (page 299) and PitchToFrequency(...) (page 299)
- Fixed: problem with StringToPitch(...) (page 299)
- List object now support multiple select mode and various custom mode.
- Fixed problem on progress bar when max is >32000.
- New Symbol values: Symbol.VerticalPixelPosition (page 309) and Symbol.VerticalPixelSize (page 309)
- New methods: Application.StartRecording(...) (page 184) and Application.StopRecording(...) (page 185) to record d from various devices
- New methods: Score.InsertBars(...) (page 224) and Score.DeleteBars(...) (page 223)
- Fixed problem when defining a complete path in Score.Save(...) (page 240)
- Fixed problem while providing a full path to GetFileNameSave(...) (page 592)

1.2.6. Version 0.2.0

September 2003 (Harmony V 8.4)

- New value Score.NumberOfPrintedPages (page 211)
- New parameters for Score.Print(...) (page 239)
- New methods Score.DoPrinterJob(...) (page 239), Score.PreparePageForPrinting(...) (page 239), Score.PrintPendingPages(...) (page 239)
- Fixed problem with SplitName(...) (page 593) when the name include more than one '.' character
- Fixed problem with Symbol.IsCautionaryAccidental (page 308)
- New method Score.DeleteTemporaryStaves(...) (page 243)
- High level events on floating windows
- Removed method DigitalData.FFTBlock(...)
- Fixed problem with Graph.DrawChar(...) (page 528)

1.2.7. Version 0.2.1

October 2003 (Harmony V 8.4)

- New method ProgressBar.Maximum(...) (page 577)
- New method Score.TimeToBeat(...) (page 226)
- New method Score.BeatToTime(...) (page 226)
- New method Score.GetCurrentTempo(...) (page 248)
- The three parameters for VS/RS singing formant can be read and written by script
- Fixed: phonetic extraction from lyrics when a syllable included an unbreakable space.
- Fixed: Origin problem in Symbol.VerticalPixelPosition, and in bounding for Staff.ChordYPosition, Staff.LyricsYPosition and Staff.BarNumberingYPosition
- Improvement: The BrowseFolder function is not affected by adding files or deleting files in the target folder while processing.
- New methods to manage current sound base: IsSoundIDAvailable and ClosestAvailableSoundID
- A new parameter in File.Load() avoid the sound conversion in prefered sounds when current sound base is GOLD.
- New method Score.AddSelection(...) (page 241)
- New value GlobalBarSetting.IsForcedDisplay (page 319)
- New value GlobalBarSetting.IsDisabledDisplay (page 319)
- New method Score.ComputePageSetup(...) (page 239)

1.2.8. Version 0.2.2

January 2004 (Harmony V 8.4.2)

- A Lua session now closes all opened files when aborting.
- New global functions GetLocalCopy(...) (page 177),PutLocalCopy(...) (page 177)
- New global functions: Binary operations BinAnd(...) (page 174),[BinOr(...) (page 174),BinXor(...) (page 174),BinShl ,BinShr(...) (page 174)
- New global functions: Bit operations BitTst(...) (page 175),BitClr(...) (page 175),BitSet(...) (page 175)
- New global function: String encoding conversion ConvertStringEncoding(...) (page 175)
- New object type FreeObject (page 391)
- New values for Harmonica tablature Symbol.IsBlow (page 304), Symbol.IsDraw (page 304), Symbol.HoleNumber (Symbol.IsSlide (page 304)
- New values for Accordion tablature Symbol.IsAccordionPush (page 305), Symbol.IsAccordionPull (page 305), Symbol.AccordionRow (page 305), Symbol.AccordionLine (page 305)
- New value Staff.SymbolDefaultSize (page 264)
- New methods: Score.ComputeBeam(...) (page 224), Staff.ComputeBeam(...) (page 272),
- New method: Score.GetWrittenDuration(...) (page 226)
- New method: Score.PlayedBarToWrittenBar(...) (page 227)
- New method: Score.CopyTo(...) (page 235)
- New method: Score.HumanizeSelection(...) (page 242)
- New method: Score.SelectAll(...) (page 240)
- New method: Staff.ApplyTransposingInstrument(...) (page 294)
- New method: Staff.CreateDefaultDrumRules(...) (page 273)
- New method: Staff.GetPage(...) (page 294)
- New method: Symbol.GetTiedElements(...) (page 316)
- New method: Staff.RemoveUnusedLyrics(...) (page 287)
- New method: Tablature.SearchClosestSound(...) (page 373)
- New methods: File.SetPos(...) (page 609), File.GetPos(...) (page 609)
- New method: Instrument.Delete(...) (page 438)
- New parameters for Score.CopySelection(...) (page 241)
- New parameters for Score.EraseSelection(...) (page 241)
- New parameters for Staff.AddRule(...) (page 289)
- Fix: File.Position (page 604)
- Fix: Rule.AppliedInstrument (page 420) (was 0 based)
- Fix: HarmonicaTab.SetHole(...) (page 378) (was bugged)
- Fix: Crash on variable window when displayed string longer than 255 characters
- Fix: Symbol.GetSlurSettings(...) (page 315) returned values were wrong
- Fix: Staff.SetChordName(...) (page 278)
- Fix: Staff.LyricsText (page 257) now equals nil when the staff has no lyrics
- Fix: Staff.ChordText (page 260) now equals nil when the staff has no chord line
- Fix: PageSetup.Scale (page 407) can now be set properly, and is expressed in percent
- Fix: Clef.Scale (page 336) can now be set properly to 0 (defaultscale)
- Fix: Staff.SymbolDefaultColor (page 265): the wrong color index was returned
- Fix: Symbol.RGBColor (page 309) now returns the staff's default symbol color when the symbol color is set to 0 (default)
- Fix: Staff.OverlayNote(...) (page 291) the object returned by this method was sometimes wrong

1.2.9. Version 1.0.0

June 2004 (Harmony V 9.0)

· New object: Movie (page 615)
· New object: StorageArea (page 396) to store persistent private data in score or staff object, as well as volatile priva data related to the current program session.
· New method: High-level event for floating palettes: OnInsertSymbol, to apply a script when a symbol is inserted on score
· New method: High-level event for floating palettes: OnInjectSymbol, to apply a script when a symbol is changed through the injector tool
· New value: Symbol.RestVerticalOffset (page 309)
· New value: Symbol.IsInvertedTie (page 311)
· New value: Application.Computer (page 180)
· New methods: Graph.GetFontID(...) (page 527) Graph.GetFontName(...) (page 527) Graph.GetCurrentFontHeight(527)
· New values: Staff.ChordVerticalSize (page 260) Staff.LyricsVerticalSize (page 257)
· New value: PageSetup.JustifyGroupButNotLastPage (page 409)
· New methods: Symbol.GetRelatedRule(...) (page 316) Symbol.SetRelatedRule(...) (page 317)
· New value: Rule.IsUseful (page 424)
· New value: DialogItem.ContextHelp (page 511)
· New value: Dialog.ContextHelp (page 497)
· New value: Dialog.DefaultPath (page 497)
· New value: Symbol.IsSelectedIndividually (page 306)
· New method: Score.DeselectIndividually(...) (page 240)
· New value: Staff.IsAutoTranspose (page 258)
· New value: Dynamic.IsCumulative (page 354)
· New value: Dynamic.Title (page 353)
· New value: Dynamic.TitleFont (page 353)
· New value: Dynamic.TitleFace (page 354)
· New value: Dynamic.TitleSize (page 354)
· New methods: Dynamic.GetActivationState(...) (page 356) Dynamic.SetActivationState(...) (page 356)
· New value Dynamic.TempoNoteScale (page 356)
· New value Dynamic.TempoCurvePoint1 (page 356)
· New value Dynamic.TempoCurvePoint2 (page 356)
· New value: Symbol.IsGraceNoteDisplayed (page 303)
· New value: Clef.IsDisplayStaffLines (page 336)
· New methods: Symbol.GetBeamSettings(...) (page 316) Symbol.SetBeamSettings(...) (page 316)
· New parameter for Score.Preserve(...) (page 248)
· New high-level event:
· Fix: crash in SetLyricsCellContent when input string was empty
· Fix: Various fixes in lyrics functions
· Fix: Score.PreparePageForPrinting(...) (page 239) was using an incorrect scale
· Fix: DigitalData.Erase(...) (page 447) was crashing
· Fix: RawTable.Operation(...) (page 453) was providing unexpected result when applying a constant
· Fix: Score.MetronomeInstrumentForQuarterNoteTick (page 208) Score.MetronomeInstrumentForBarTick (page 20
· Fix: GlobalBarSetting.CopyTo(...) (page 320) does not affect anymore the bar number
· Fix: all members of the note related to Harmonica and Accordion were corrupted in reading and writing.

1.2.10. Version 1.1.0

July 2004

· New value: Staff.IsLayerActive (page 268)
· Fix: IsDisplayWholeRest renamed in IsDisplayWholeRests
· Fix: Comment Justification values

1.2.11. Version 1.1.1

August 2004

· New methods for Score (page 198) object: Score.PlayedTimeToPlayedBar(...) (page 226) Score.PlayedBarToPlayedTime(...) (page 226) Score.WrittenTimeToPlayedTime(...) (page 227) Score.WrittenBarToPlayedBar(...) (page 227) Score.GetPlayedDuration(...) (page 226)
· Fix: Score.GetWrittenDuration(...) (page 226) was providing played duration instead of written duration.

1.2.12. Version 1.1.2

August 2004

· Fix: PageSetup.DistanceBetweenGroup (page 408)

1.2.13. Version 1.1.3

October 2004

- Fix: file.Read("a") now read all characters including cariage return.
- Fix: Symbol.Delete(...) (page 312) now takes the parameter into account
- Fix: Staff.InsertNote(...) (page 291) now takes "insert operation is limited to one bar" parameter into account
- Fix: Staff.NumberOfChords (page 260)
- New method Graph.GetPictureSize(...) (page 529)
- New value Symbol.IsInvertedHeadDisplayed (page 311)
- New method Score.DeselectAll(...) (page 240)
- New value PageSetup.SwapLeftRightMargin (page 407)

1.2.14. Version 1.1.4

December 2004

- French translation of the manual by Jean-Pierre Butin
- New value: Symbol.IsGraceNoteStemsDown (page 303)
- New value: GlobalBarSetting.IsEndOfPage (page 320)
- New High-Levent event for floating windows: OnWillInjectSymbol
- New method: Dynamic.CopyTo(...) (page 357)
- New method: Dialog.Hide(...) (page 503)
- New method: Staff.ComputeStemsDirection(...) (page 294)
- New value: Staff.IsDisplayOnlyIfFirstStaffOfGroup (page 255)
- New value: Staff.LyricsNumberOfLines (page 257)
- New method: Dialog.NewFrameRect(...) (page 501)
- New method: File.XMLParse(...) (page 609)
- ConvertStringEncoding can now convert from and to HTML
- ConvertStringEncoding can now convert from and to UTF-8
- New Interface Composer object : Tab View
- RAW_TABLE_LONG RawTable type now supported in DigitalData.GetBlock(...) (page 447), DigitalData.SetBlock(. (page 448), and RawTable.Operation(...) (page 453)
- Fix: Changing crescendo dynamic value
- Fix: Miscellaneous Jukebox values are now availables
- Fix: Symbol.VerticalPixelPosition (page 309)

1.2.15. Version 1.2

April 2005

- New definition for MSSounds.mys including GOLD Base II sounds.
- New object: StavesGroup
- New method: Score.AddStavesGroup(...) (page 244)
- New value: Score.FirstStavesGroup (page 219)
- New value: Score.NumberOfStavesGroups (page 220)
- New value: Score.StavesGroups (page 219)
- New value: Score.IsColoredMultiVoices (page 201)
- New method: StavesGroup.Delete(...) (page 331)
- New method: StavesGroup.CopyTo(...) (page 331)
- New value: StavesGroup.Symbol (page 330)
- New value: StavesGroup.JoinMode (page 330)
- New value: StavesGroup.Symbol (page 330)
- New value: StavesGroup.StaffBegin (page 331)
- New value: StavesGroup.StaffEnd (page 331)
- New value: StavesGroup.HorizontalOffset (page 331)
- New value: StavesGroup.Title (page 329)
- New value: StavesGroup.TitleXOffset (page 329)
- New value: StavesGroup.TitleYOffset (page 329)
- New value: StavesGroup.TitleFont (page 329)
- New value: StavesGroup.TitleFace (page 329)
- New value: StavesGroup.TitleSize (page 329)
- New value: StavesGroup.TitleRGBColor (page 329)
- New value: StavesGroup.ShortTitle (page 330)
- New value: StavesGroup.ShortTitleXOffset (page 330)
- New value: StavesGroup.ShortTitleYOffset (page 330)
- New method: BarAction.CopyTo(...) (page 327)
- New method: Instrument.CopyTo(...) (page 439)
- New method: Staff.GetLyricAtTime(...) (page 283)
- New method: Staff.SetLyricAtTime(...) (page 284)
- New method: Staff.GetLyricsAtBar(...) (page 284)
- New method: Staff.SetLyricsAtBar(...) (page 285)
- New value: Staff.Index (page 252)
- New method: Score.SetPlayMark(...) (page 237)
- New method: Score.NumberOfSystemsInPage(...) (page 244)
- New method: Score.NumberOfStavesInSystem(...) (page 245)
- New method: Score.BarRangeInSystem(...) (page 245)
- New method: Score.GetStaffInSystem(...) (page 245)
- New method: Score.AdjustLocalMetrics(...) (page 235)
- New method: Score.SystemArea(...) (page 246)
- New method: Score.BarToSystem(...) (page 246)
- New method: Staff.AdjustLocalMetrics(...) (page 274)
- New method: Score.DeleteLocalMetrics(...) (page 235)
- New method: Staff.DeleteLocalMetrics(...) (page 274)
- New method: Staff.SetLocalHeight(...) (page 274)
- New method: Staff.GetLocalHeight(...) (page 274)
- New method: Staff.SetLocalBaseLine(...) (page 275)
- New method: Staff.GetLocalBaseLine(...) (page 275)
- New method: Staff.SetLocalChordYPosition(...) (page 275)
- New method: Staff.GetLocalChordYPosition(...) (page 275)
- New method: Staff.SetLocalLyricsYPosition(...) (page 276)
- New method: Staff.GetLocalLyricsYPosition(...) (page 276)
- New method: Staff.SetLocalBarNumberingYPosition(...) (page 276)
- New method: Staff.GetLocalBarNumberingYPosition(...) (page 276)
- New method: Staff.SetLocalLyricLineVisibility(...) (page 276)
- New method: Staff.GetLocalLyricLineVisibility(...) (page 276)
- New method: Staff.SetLocalLeftMargin(...) (page 276)
- New method: Staff.GetLocalLeftMargin(...) (page 276)
- New method: Staff.SetLocalRightMargin(...) (page 277)
- New method: Staff.GetLocalRightMargin(...) (page 277)
- New method: Staff.SetLocalTopMargin(...) (page 277)
- New method: Staff.GetLocalTopMargin(...) (page 277)
- New method: Staff.AddGraphicLine(...) (page 289)
- New value: Staff.GroupsMargin (page 271)
- New value: Picture.GraphicLineBeginKind (page 360)
- New value: Picture.GraphicLineEndKind (page 360)
- New value: Picture.GraphicLineBeginTime (page 360)
- New value: Picture.GraphicLineBeginYPos (page 360)
- New value: Picture.GraphicLineEndTime (page 360)

- New value: Picture.GraphicLineEndYPos (page 360)
- New value: Picture.GraphicLineWidth (page 360)
- New value: Picture.GraphicLineBracketSize (page 360)
- New value: Staff.FirstMergedStaff (page 268)
- New value: Staff.LastMergedStaff (page 268)
- New value: Symbol.IsRealDurationRest (page 312)
- New value: DigitalTrack.BeginSelection (page 450)
- New value: DigitalTrack.EndSelection (page 450)
- New method: Tablature.SetLocalBaseLine(...) (page 373)
- New method: Tablature.GetLocalBaseLine(...) (page 373)
- New method: Tablature.VerticalSize (page 369)
- New method: Graph.StringSizes(...) (page 529)
- New method: SetFileSize(...) (page 601)
- New method: File.Flush(...) (page 605)
- New method: File.ReadRawTable(...) (page 608)
- New method: File.WriteRawTable(...) (page 608)
- New method: Midi.ProgramChange(...) (page 612)
- New method: Midi.BankChange(...) (page 612)
- New method: Midi.Pedal(...) (page 613)
- New method: Midi.RawPacket(...) (page 613)
- New method: Graph.DrawPaletteButton(...) (page 533)
- New parameters: Score.PlayMusic(...) (page 237)
- New parameter in Score.Undo(...) (page 249) to disable "Redo"
- KeyDown method can now be applied to floating window
- Graph.DrawStaff(...) (page 532) accept a new parameter: the display scale.
- Deprecated value: Staff.Bracket, now managed with StavesGroup object
- Deprecated value: Staff.Brace, now managed with StavesGroup object
- Fix: dynamic type for SFP and other type greater than TEMPO
- Fix: IsInvisibleRest was true if note own an inverted head.
- Fix: PichToFrequency renamed into PitchToFrequency

1.2.16. Version 1.2.1

May 2005

- New value: PageHeader.LeftMargin (page 411)
- New value: PageHeader.RightMargin (page 411)
- New value: PageHeader.VerticalMargin (page 412)
- Fix: Crashes in Staff.CreateLyricsArray(...) (page 285)
- Fix: Staff.IsLyrics (page 256) creates a default VS voice when switched on
- Fix: Computation errors in Score.TimeToBeat(...) (page 226) and Score.BeatToTime(...) (page 226)
- Fix: Wrong default values in Staff.SetLyricsAtBar(...) (page 285)

1.2.17. Version 1.2.2

September 2005

- New method: GetSoundRange(...) (page 429)
- New method: Staff.AddTablature(...) (page 295)
- New method: Internet.DownloadFile(...) (page 614)
- New method: PackFile(...) (page 602)
- New method: UnpackFile(...) (page 602)
- New value: Symbol.ComputeTab (page 304)
- New value: GlobalBarSetting.DontBreakLine (page 320)
- New value: Rule.IsSeparateBuffer (page 420)
- Fixed default accidental on grace note
- Fixed: the fine adjustement on lyric Y position was only applied to master staff.

1.2.18. Version 1.2.3

October 2005

- New global method: GetSystemSettingsPathName(...) (page 599)
- New method: File.ReadSignedChar(...) (page 607)
- New method: File.ReadShortIntel(...) (page 607)
- New method: File.ReadUnsignedShortIntel(...) (page 607)
- New method: File.ReadShortMotorola(...) (page 607)
- New method: File.ReadUnsignedShortMotorola(...) (page 607)
- New method: File.ReadLongIntel(...) (page 607)
- New method: File.ReadLongMotorola(...) (page 608)
- New method: File.ReadUnsignedLongIntel(...) (page 608)
- New method: File.ReadUnsignedLongMotorola(...) (page 608)
- New method: Staff.VirtualSingerVoiceKind(...) (page 296)
- New method: Staff.SaveVirtualSingerVoice(...) (page 296)
- New method: Staff.LoadVirtualSingerVoice(...) (page 297)
- New method: Symbol.ComputeStemDirection(...) (page 312)
- New method: Score.WrittenTimeToTempo(...) (page 228)
- New method: Score.DeleteEmptyStaves(...) (page 243)
- New method: Score.DeleteEmptyStaves(...) (page 243)
- New method: Application.ClearLog(...) (page 195)
- New method: Application.SaveLog(...) (page 195)
- New method: Score.SubstTokenString(...) (page 240)
- New value: Picture.GraphicLineCurveWidth (page 360)
- New value: Picture.GraphicLineCurveHeight (page 360)
- New value: FreeObject.NoWrap (page 393)

1.2.19. Version 1.3.0

February 2006

- New value: PageHeader.DontPrintOnFirstPage (page 411)
- New value: Staff.MaximumBeamTilt (page 266)
- New value: Staff.BeamThickness (page 266)
- New value: Staff.BeamGap (page 266)
- New value: Staff.UnbreakableSpaceSlurWidth (page 266)
- New value: Staff.TitleRotation (page 253)
- New value: Staff.ShortTitleRotation (page 254)
- New value: Staff.TitleAlignment (page 253)
- New value: Staff.TitleOrigin (page 253)
- New value: Staff.DefaultStemDirection (page 266)
- New value: Score.TransposedMode (page 201)
- New value: Score.SetPlayedBars(...) (page 237)
- New value: Score.GetPlayedBars(...) (page 237)
- New value: Score.ComputeSystemsLeftMargin(...) (page 235)
- New value: Score.IsForcedPlayedBarsList (page 210)
- New value: Score.DisplayGhostRests (page 211)
- New value: Score.AccidentalAppliesToAllOctaves (page 211)
- New value: Score.JoinParts (page 211)
- New value: Score.SpaceBetweenAccidentalAndNote (page 211)
- New value: Score.SpaceBetweenAccidentalOnNotes (page 211)
- New value: Score.MinimumVerticalSpacingBetweenAccidental (page 211)
- New value: Score.SpaceBetweenDotAndNote (page 211)
- New value: Score.SpaceBetweenDotAndRest (page 212)
- New value: Score.SpaceBetweenDotAndDot (page 212)
- New value: Score.HorizontalDotAdjustementForUpstemFlag (page 212)
- New value: Score.VerticalDotOffset (page 212)
- New value: Score.DotScale (page 212)
- New value: Score.BreakSymbolHeavyLineThickNess (page 212)
- New value: Score.BreakSymbolThinLineThickNess (page 212)
- New value: Score.BreakSymbolSpaceBetweenDoubleBarlines (page 212)
- New value: Score.BreakSymbolSpaceBetweenHeavyLineAndThinLine (page 212)
- New value: Score.BreakSymbolDashLineLength (page 212)
- New value: Score.BreakSymbolDashSpaceLength (page 212)
- New value: Score.SpaceBeforeClef (page 212)
- New value: Score.SpaceBeforeKeySignature (page 212)
- New value: Score.SpaceAfterKeySignature (page 213)
- New value: Score.SpaceBeforeTimeSignature (page 213)
- New value: Score.SpaceAfterTimeSignature (page 213)
- New value: Score.SpaceBetweenKeySignatureAccidentals (page 213)
- New value: Score.SpaceAfterCancelledKey (page 213)
- New value: Score.GraceNoteSize (page 213)
- New value: Score.StemLineThickNess (page 213)
- New value: Score.StemOffsetLength (page 213)
- New value: Score.SpaceBeforeMultiRest (page 213)
- New value: Score.VerticalOffsetMultiRest (page 213)
- New value: Score.VerticalOffsetMultiRestNumber (page 214)
- New value: Score.MultiRestLineWidth (page 214)
- New value: Score.MultiRestEdgeLineHeight (page 214)
- New value: Score.MultiRestTextSize (page 214)
- New value: Score.SpaceBeforeMultiBar (page 214)
- New value: Score.VerticalOffsetMultiBar (page 214)
- New value: Score.VerticalOffsetMultiBarNumber (page 214)
- New value: Score.MultiBarTextSize (page 214)
- New value: Score.MultiBarSymbolScale (page 214)
- New value: Score.StaffLineThickNess (page 214)
- New value: Score.StaffEndingLineThickNess (page 214)
- New value: Score.LedgerLineThickNess (page 214)
- New value: Score.LyricDashLineThickNess (page 214)
- New value: Score.LyricDashLineLength (page 215)
- New value: Score.LyricUnderLineThickNess (page 215)
- New value: Score.LyricSpaceBetweenDashLineAndLetter (page 215)
- New value: Score.LyricSpaceBetweenMelismaAndLetter (page 215)
- New value: Score.LyricMinimalSpaceBetweenSyllable (page 215)
- New value: Score.LyricMaximalSpaceBetweenDashLine (page 215)
- New value: Score.StaffTabLineThickNess (page 215)
- New value: Score.TabAccordionPushPullLineThickNess (page 215)
- New value: Score.TabHarmonicaTriangleLineThickNess (page 215)
- New value: Score.TabHarmonicaArrowLineThickNess (page 215)

- New value: Score.DownwardTieWidth (page 215)
- New value: Score.DownwardTieDeltaXStart (page 215)
- New value: Score.DownwardTieDeltaYStart (page 215)
- New value: Score.DownwardTieDeltaXEnd (page 216)
- New value: Score.DownwardTieDeltaYEnd (page 216)
- New value: Score.DownwardTieDeltaXCurve (page 216)
- New value: Score.DownwardTieDeltaYCurve (page 216)
- New value: Score.UpwardTieWidth (page 216)
- New value: Score.UpwardTieDeltaXStart (page 216)
- New value: Score.UpwardTieDeltaYStart (page 216)
- New value: Score.UpwardTieDeltaXEnd (page 216)
- New value: Score.UpwardTieDeltaYEnd (page 216)
- New value: Score.UpwardTieDeltaXCurve (page 216)
- New value: Score.UpwardTieDeltaYCurve (page 216)
- New value: Score.EffectWidth (page 217)
- New value: Score.StaccatoWidth (page 217)
- New value: Score.ChordDiagramLineThickNess (page 217)
- New value: Score.ChordDiagramNutSize (page 217)
- New value: Score.ChordDiagramFingerTextScale (page 217)
- New value: Score.ChordDiagramFingerCircleScale (page 217)
- New value: Score.ChordDiagramFingerBarreScale (page 217)
- New value: Score.ChordDiagramBarreFretNumberTextScale (page 217)
- New value: Score.ChordDiagramFontName (page 217)
- New value: Score.ChordDiagramFontFace (page 217)
- New value: Score.ChordDiagramFontSize (page 217)
- New value: Score.ChordDiagramFontColor (page 217)
- New value: Score.TextFrameThickness (page 218)
- New value: Score.TextFrameMargin (page 218)
- New value: Score.BraceWidth (page 218)
- New value: Score.BraceCenterOffset (page 218)
- New value: Score.BracketWidth (page 218)
- New value: Score.BracketEdgeOffset (page 218)
- New value: Staff.HiddenTransposition (page 268)
- New value: Symbol.IsRepeatTwoLastBars (page 307)
- New value: Ornament.Rotation (page 342)
- New value: Ornament.AlongStemDirection (page 342)
- New value: HarmonicaTab.InvertNumberSymbol (page 377)
- New value: HarmonicaTab.SlideAsCircle (page 377)
- New value: Comment.Rotation (page 413)
- New value: StaffText.Rotation (page 383)
- New value: StaffText.IsEraseBackground (page 384)
- New method: Staff.SetDefaultTablature(...) (page 296)
- New value: FreeObject.Rotation (page 393)
- New value: FreeObject.IsEraseBackground (page 393)
- New value: PageSetup.ReservedPages (page 407)
- New value:
- New optional parameter for Staff.OverlayNote(...) (page 291)
- New method PurgeStringStyle(...) (page 177)
- New method SetStringStyle(...) (page 178)
- New method GetStringStyle(...) (page 178)
- New method SetErrorMessage(...) (page 174)
- New method Graph.RGBToComponents(...) (page 521)
- New method Graph.ComponentsToRGB(...) (page 521)
- New high-level event for floating palette OnLoad

1.2.20. Version 1.3.1

January 2007

- New value: Dynamic.IsAppliedToMergedStaves (page 354)
- New value: Dynamic.IsAppliedToWholeGroup (page 354)

1.2.21. Version 1.4

January 2008

- New object type View (page 400)
- New method Score.NewView(...) (page 249)
- New method Score.GetCurrentView(...) (page 249)
- New method Score.SetCurrentView(...) (page 249)
- New method Score.DeleteUnusedInstruments(...) (page 236)

- New value: Dynamic.IsAppliedToMergedStaves (page 354)
- New value: Dynamic.IsAppliedToWholeGroup (page 354)
- New high-level events for floating palette OnViewsChange,OnViewChange,OnMouseWheel
- New method Staff.EditDisplayMode(...) (page 273)
- New method Score.EditBarActions(...) (page 224)
- New method Score.EditGeneralTempo(...) (page 248)
- New method Score.EditMetronome(...) (page 236)
- New method Score.EditGlobalDigitalEffects(...) (page 237)
- New method Symbol.ComputeSlursApparence(...) (page 315)
- New value PageSetup.MultiRepeatBar (page 409)
- New value VSVoice.FragmentString (page 481)

1.2.22. Version 1.4.0

June 2008

- New method Score.EditChordGrid(...) (page 224)
- New method Staff.EditChordDisplay(...) (page 273)
- New method Staff.EditChordLine(...) (page 273)

1.2.23. Version 1.4.1

October 2008

- New method Score.EditComments(...) (page 234)
- New method Staff.EditBarNumbering(...) (page 273)
- New method Staff.EditLyrics(...) (page 273)
- New method Staff.EditTablature(...) (page 274)
- New value Score.CurrentDisplayDpi (page 201)
- New method Dialog.IsFrontFloatingWindow (page 498)
- New value Dynamic.TitleColor (page 354)

1.2.24. Version 1.4.2

April 2009

- New value Score.ChordGridFontName (page 218)
- New value Score.ChordGridFontFace (page 218)
- New value Score.ChordGridFontSize (page 218)
- New value Score.ChordGridFontColor (page 218)
- New method Palette.ResetAll(...) (page 575)
- New value PageSetup.PaperOrientation (page 407)
- New method Staff.OutputLevel(...) (page 295)
- New method Score.OutputLevel(...) (page 237)
- Fixed Symbol.DiatonicOctave (page 301) value (wrong octave base)
- New value FreeObject.DontPrint (page 391)
- New value FreeObject.Locked (page 391)

1.2.25. Version 1.4.3

March 2010

- Changed entry in Staff.InsertNote(...) (page 291)
- Changed entry in Staff.InsertRest(...) (page 292)
- Changed entry in Staff.OverlayNote(...) (page 291)
- New value Application.CurrentSymbolDuration (page 179)
- New value Application.IsNotePaletteActive (page 179)
- New method Symbol.StartEcho(...) (page 313)
- New method Application.OpenManual(...) (page 195)
- New method FreeObject.BeginRecord(...) (page 395)
- New method FreeObject.EndRecord(...) (page 395)
- New method Score.GetPageMode(...) (page 251)
- New value Score.IsMusicPaused (page 210)
- New method Score.PauseMusic(...) (page 238)
- New value GlobalBarSetting.IsInvisible (page 320)
- Added entry in BrowseFolder(...) (page 594)
- New value Staff.AutoSizeLyrics (page 257)
- New value Staff.IsChanged (page 268)
- Fixed SetLyricsCellContent
- New value Application.LimitEditingToOneBar (page 181)
- Fixed User's curves

1.2.26. Version 1.4.4

February 2011

- New method Score.GetPrintMode(...) (page 251)
- New value Application.IsLeftHandedNotation (page 181)
- New method Score.GetPlayMark(...) (page 237)
- New value Score.AutomaticTargetCaptions (page 220)
- New value Score.TargetCaptionsMode (page 220)
- New value Score.FrameTargetCaptions (page 220)
- New value Score.TargetCaptionsHorizontalShift (page 220)
- New value Score.TargetCaptionsVerticalShift (page 220)
- New value Score.FirstTarget (page 220)
- New value Score.LastTarget (page 220)
- New value Score.TargetCaptionsTextStyle (page 220)
- New method Score.RegenerateTargets(...) (page 248)
- New method Score.InsertTarget(...) (page 247)
- New value VSVoice.SemiToneShift (page 477)
- New Object Target (page 398)
- New value: Score.OldQuarterRest (page 211)
- New method: Tablature.SetDefaultAccordion(...) (page 373)
- New value: Score.BreakSymbolBracket (page 212)
- New parameter for OnInjectScore : time position
- Fixed GetMouseDoubleClickButton
- Fixed Score.PositionView(...) (page 238)
- New Method Score.GetViewPosition(...) (page 239)

Version 1.4.4c
March 2011

- New high-level event for floating palette: OnScoreChange
- The method Palette.SetSize(...) (page 573) work now with Complete Mixer

1.2.27. Version 1.4.5

April 2012

- New result for Midi.GetEvent() : the time position
- New value: Tablature.ApplyToMergedStaves (page 371)
- New value: Score.StaffRightLineThickNess (page 214)
- New method: Ornament.AddPicture(...) (page 344)
- New method: Ornament.BeginRecord(...) (page 345)
- New method: Ornament.EndRecord(...) (page 346)
- New value: Ornament.GraphicScale (page 343)
- New value: Ornament.GraphRecord (page 343)
- New method: BarSetting.DeleteAllOrnaments(...) (page 335)
- New method: BarSetting.AddOrnament(...) (page 335)
- New parameter for Staff.InsertBarSetting(...) (page 271)
- New parameter for Staff.EditLyrics(...) (page 273)
- New method: Staff.InsertLineInLyrics(...) (page 280)
- New method: Staff.HideLineInLyrics(...) (page 281)
- New method: Staff.CompactLinesInLyrics(...) (page 281)
- New method: Staff.UnHideLinesInLyrics(...) (page 281)
- Change on value: Staff.IsHideText (page 267)
- New value: Staff.IsHideLyric (page 267)
- New value: Staff.IsHideChord (page 267)
- New value: Staff.IsHideChordDiagram (page 267)
- New value: Staff.IsHideGraphic (page 267)
- New value: Staff.IsHideDynamic (page 267)
- New value: Staff.IsHideFXProcessor (page 267)
- Added output value to Symbol.GetSlurSettings(...) (page 315) and input value to Symbol.SetSlurSettings(...) (page slur tension
- New method: Graph.CurveTo(...) (page 523)
- New method: Graph.EraseCurvePolygon(...) (page 531)
- New method: Graph.PaintCurvePolygon(...) (page 531)
- New method: Graph.FrameCurvePolygon(...) (page 531)
- New method: Graph.RGBForeColor(...) (page 522)
- New method: Graph.RGBBackColor(...) (page 522)
- New value: Ornament.IsFramed (page 343)
- New value: Picture.IsRelatedToOrnament (page 359)
- New method: DialogItem.Scroll(...) (page 512)
- Fixed: Dialog.GetItemUnderMouse(...) (page 504) on scroll bars related to some items
- Added JPG, PNG, TIFF and GIF support to Graph.DrawPicture(...) (page 529)
- New method: Graph.NewPictFile(...) (page 533)
- New method: Graph.WritePictFile(...) (page 533)
- New global function : CopyFile(...) (page 603)
- New value: Score.DownwardTieTension (page 215)
- New value: Score.UpwardTieTension (page 216)
- Application.Palette.SendHelpMessage: the string is now styled.
- New value:
- New method: Application.Close(...) (page 189)
- New parameter for Palette.SendHelpMessage(...) (page 574)
- New value:
- New value:
- New parameters for : ID3 tags management

1.2.28. Version 1.4.6

June 2013

- New feature, autocomplete in MyrScript editor
- New parameter for : played note color
- New method: Symbol.Draw(...) (page 312)
- New method: StaffText.Draw(...) (page 385)
- New method: Staff.ComputePitchDisplayedHeight(...) (page 294)
- New value: Symbol.Tag (page 307)
- New value: FreeObject.Tag (page 394)
- New value: Picture.Tag (page 361)
- New value: Picture.Staff (page 359)
- New method: Score.SearchSymbolsByTag(...) (page 246)
- New method: Score.ClearTagSymbols(...) (page 247)
- New method: Score.SearchFreeObjectsByTag(...) (page 247)
- New method: Score.ClearTagFreeObjects(...) (page 247)
- New method: Score.SearchPicturesByTag(...) (page 247)
- New method: Score.ClearTagPictures(...) (page 247)
- New method: Staff.SearchSymbolsByTag(...) (page 296)
- New method: Staff.ClearTagSymbols(...) (page 296)
- New method: Staff.SearchPicturesByTag(...) (page 296)
- New method: Staff.ClearTagPictures(...) (page 296)
- New method: Score.SearchOrnamentsByTag(...) (page 247)
- New method: Score.ClearTagOrnaments(...) (page 247)
- New method: Score.AdjustSlurs(...) (page 236)
- New value: Ornament.Symbol (page 342)
- New value: Application.RGBSelectColor (page 181)
- New method: Palette.Magnetize(...) (page 573)
- New method: Palette.GetScale(...) (page 574)
- New method: Palette.SetScale(...) (page 574)
- New value: Score.DrumGridTriangleLineSize (page 218)
- New value: Score.DrumGridTimeBarLineSize (page 218)
- New value: Score.DrumGridBarLineSize (page 219)
- New value: Score.DrumGridCellBarLineSize (page 219)
- Changed value: DigitalSound.LoopEnd (page 443) is now a float number
- Changed value: Ornament.GraphicScale (page 343) is now in percent
- New object: Surface (page 535)
- New method: Graph.DrawSurface(...) (page 529)
- New method: Graph.CopyRect(...) (page 523)
- New values: Graph.AreaLeft (page 519), Graph.AreaTop (page 519), Graph.AreaRight (page 519), Graph.AreaBot

- New parameter: idle time in Dialog.Select(...) (page 503)
- New parameter: Staff.Duplicate(...) (page 290)
- New value: GlobalBarSetting.IsBeamEighthsByFour (page 319)
- New parameters for Palette.KeyPressed(...) (page 575) fingering and color

1.2.29. Version 1.4.7

October 2016

- New value: Symbol.IsInvisibleDot (page 311)
- New value: Symbol.OffsetXDot (page 311)
- New value: Symbol.OffsetYDot (page 312)
- New value: Symbol.IsParenthesis (page 311)
- New value: GraceNote.IsParenthesis (page 350)
- Added a set of methods to use the synthesized human voice.
- New value: Application.IsVirtualSingerActive (page 182)
- New value: Symbol.TupletSymbolCurveTension (page 311)
- New value: FXProcessor.MultibandEQGain (page 465)
- New value: FXProcessor.MultibandEQNumberOfBands (page 465)
- New value: FXProcessor.MultibandEQFrequencies (page 465)
- New value: FXProcessor.MultibandEQLevels (page 465)
- New value: Tablature.Font (page 369)
- New value: Tablature.FontSize (page 369)
- New value: Tablature.FontFace (page 369)
- New value: Tablature.FontColor (page 369)
- New value: Tablature.AdditionalFont (page 370)
- New value: Tablature.AdditionalFontSize (page 370)
- New value: Tablature.AdditionalFontFace (page 370)
- New value: Tablature.AdditionalFontColor (page 370)
- New value: Score.HairPinWidth (page 218)
- New method: Score.JumpToPage(...) (page 238)
- New method: Score.JumpToBar(...) (page 238)
- New value: Score.IsDisplayControlHandles (page 201)
- New value: Score.ExtendParts (page 211)
- New method: Score.WebExport(...) (page 230)
- New method: Application.WebExportSetup(...) (page 195)
- New method: Application.GetWebExportSetup(...) (page 195)
- New value: Staff.MyrWebGroup (page 268)
- New method: Application.ConvertStringToBase64(...) (page 182)
- New method: Application.ConvertFileToBase64(...) (page 182)
- New method: Application.ConvertBase64ToString(...) (page 182)
- New primitive: strifind()
- New value: PageSetup.AddChordDiagrams (page 408)

1.2.30. Version 1.4.8

September 2017

- New method: Application.StringDuration(...) (page 187)
- New value: Tablature.IsDisplaySlurs (page 368)
- New method: DigitalSound.Export(...) (page 443)
- New value: Staff.NeverHide (page 268)
- New value: Score.IsAmbitusEnabled (page 211)
- New value: Staff.IsAmbitusEnabled (page 268)
- New value: Score.AmbitusLineStyle (page 200)
- New value: Score.AmbitusLineThickness (page 200)
- New value: Score.AmbitusHorizontalSize (page 200)
- New value: Score.AmbitusHorizontalOffset (page 200)
- New parameter for: Score.GraphicalExport(...) (page 233)

1.2.31. Version 1.4.9

January 2018

- New method: Application.ConvertPitchListToChord(...) (page 183)

1.2.32. Version 1.5.0

February 2018

- New method: Application.GetChordDiagramFromIndex(...) (page 184)
- New method: Diagram.GetFinger(...) (page 366)
- New method: Diagram.AddFinger(...) (page 365)
- New method: Diagram.GetFingerNumberForString(...) (page 366)
- New method: Diagram.GetCellFromString(...) (page 366)
- New method: Staff.AddChordDiagram(...) (page 279)
- New method: Staff.GetChordDiagramFromName(...) (page 279)
- New method: Staff.GetChordDiagramFromIndex(...) (page 279)
- New method: Staff.DeleteChordDiagrams(...) (page 279)
- New method: Staff.DuplicateChordDiagrams(...) (page 279)
- New method: Staff.NumberOfChordDiagrams (page 257)
- New method: Score.AddChordDiagram(...) (page 225)
- New method: Score.GetChordDiagramFromName(...) (page 225)
- New method: Score.GetChordDiagramFromIndex(...) (page 225)
- New method: Score.DeleteChordDiagrams(...) (page 225)
- New method: Score.DuplicateChordDiagrams(...) (page 225)
- New value: Score.NumberOfChordDiagrams (page 201)
- New value: Diagram.NumberOfStrings (page 364)
- New value: Diagram.NumberOfFingers (page 365)
- New value: Diagram.StringValue (page 365)
- New value: Diagram.ChordName (page 364)
- New value: Diagram.Duplicate(...) (page 366)
- New value: Chord.Diagram (page 387)
- New value: Dynamic.Scale (page 356)

1.2.33. Version 1.5.1

May 2018

- New value: AccordionTab.ComputeRightHandAlongLeftHand (page 380)
- New value: AccordionTab.LeftHandChordDuration (page 380)

1.2.34. Version 1.6.2

January 2020

- New parameter: Application.GetHumanVoiceInformation(...) (page 192)
- New parameter: Application.Say(...) (page 190) New parameter
- New method: Application.SearchHumanVoice(...) (page 193)
- New value: Application.humanVoice (page 179)
- New value: Score.HorizontalMirror (page 211)
- New method: Application.SetHumanVoiceDefault(...) (page 191)
- New value: Staff.SpokenVoiceName (page 269)
- New value: Staff.SpokenVoiceLanguage (page 269)
- New value: Staff.SpokenVoiceGender (page 269)
- New value: Staff.SpokenVoiceRate (page 269)
- New value: Staff.SpokenVoiceMaximumRate (page 269)
- New value: Staff.SpokenVoicePitchOffset (page 269)

1.2.35. Version 1.6.3

February 2020

- New method: Score.SaveEngraverSetup(...) (page 228)
- New method: Score.LoadEngraverSetup(...) (page 228)
- New method: Score.MovieExport(...) (page 231)

1.3. *Scripts* Menu

1.3.1. Overview

The *Scripts* menu is automatically created by the program. It reflect the hierarchy of the program "Scripts" folder, mixed with the user "Scripts" folder.
If a script comes from the user folder, its name is tagged with a little red star.
Is a script is designed to run in background it is tagged with the icon.
If a script exists both in the program "Scripts" folder and the user "Scripts" folder, the user script is taken in priority. (The program's script does not appear.)
It the program's script version is greater than the user's script version, the program's script will be used instead.
You can add new categories by creating new folders in the user's _Scripts_ folder. This kind of sub-folder is shown with the icon.
Standard categories are tagged with the icon in the menu.

Note: The "Scripts" menu is rebuilt each time a script file is saved.

1.3.2. Launching a script from the menu

When you select an item in the *Scripts* menu, the script is launched.
If the Command key is pressed (Apple on Macintosh, Ctrl on Windows) the Script is not launched but loaded in an editor window. You can step it or apply changes and create a new script file from this sample.

1.3.3. Priority of script

When a script is launched and needs to access another script file (for example through the global function "Include"),the file is searched first at the level of the main file, then in the "Includes" folder of the user "Scripts" folder, then in the " Includes" folder of the program "Scripts" folder.

1.3.4. Creating a new script

The *New script file* menu item creates a new empty script window.

1.3.5. Opening and existing script

The *Open...* menu item opens an existing script in a new window or install an archived script (.msa file).

1.3.6. Stopping script execution

The *Stop* menu item stops the currently running script. The user can also press the "ESC" key or "Command+Shift+Dot" to break a script.
In your script you have to call the Application.UserBreak(...) (page 187) method from time to time, in order to test the use request.
If your script falls into a deadlock, the "Stop" item can be very useful.

1.3.7. Downloading the most recent version of this manual

When opening the developer manual from the *Scripts* menu, this manual is displayed in a window.
If the manual is not found, manual file is downloaded from the Myriad Web site.
You can force this download procedure, (for example if you have been informed by the Myriad team that a new version of this manual is available) by selecting this item while Command key is down.
The manual file is located in the "Docs>MyrScript" folder of the "Myriad Documents" folder in your user area.

1.3.8. Script version

You can define the version of your script by using the tag : "--VERSION:" in your script file header.
The version number follows the a.b.c format where a,b,c are decimal values from 0 to 99.
For example :
--VERSION: 1.3.4
Will specify the major version to 1, the minor version to 3, the minor sub-version to 4.
This version will be used to manage automatic update and script priority. For example, if the user has installed a script in version 1.4 and if the new application version installs version 2.0 of the same script in the predefined scripts folder, the most recent version will be taken into account.
In this case the user version of the script will be ignored.

1.3.9. About scripts...

The "About scripts..." item displays the complete information defined in the script file.
The developer has to define some tags in the header of his source file.
Because the script can be used by a non english speaking people, information can be also provided in other languages.
Add the following strings to the tag name :

- --FR for french language
- --IT for italian language
- --DE for german language
- --SP for spanish language
- --JP for japanese language
- --PT for portuguese language
- --NL for dutch language

Here are the tags and their meaning :
--NAME: Complete name of the script in english language
--NAME_IN_MENU: Name displayed in the menu, english language
--CREATOR: Creator of the script in english language
--DATE: Date of creation of the script in english language
--ABSTRACT: Short line describing the script. This line will be displayed in the help window palette.
--INFO: Complete description of the script in english language.
--DIFFUSION_MODE: Diffusion mode of the script. When the user asks information about your script, the diffusion mode that has been set for this script will be displayed in the user's language. You can use the values 1,2, 3 or 4:

- 1 (by default) "The author allows to use, share freely, apply changes or use parts of this script without prior consent."
- 2 "The author allows to use or share freely this script but forbides to apply changes or use parts of it without prior consent."
- 3 "The author allows to use this script but forbides to share, apply changes or use parts of it without prior consent."
- 4 "The author forbides to use, share, apply changes or use parts of this script without prior consent."

Note: A script can be protected by password to avoid to be read by another person

For floating windows only:

--MonoInstance prevent the user to open twice the same floating window
--TransluscentRatio xxx" where xxxx is the transluscent ratio. A ratio of 0 mean "opaque". (Windows only)
--NoPalette If you do not want your floating window to behave like a palette

Some tags can be localized in other languages. Use :
--NAME-xx:
--NAME_IN_MENU-xx:
--CREATOR-xx:
--DATE-xx:
--ABSTRACT-xx:
--INFO-xx:

where "xx" is FR for french language, IT for italian language, DE for german language, SP for spanish language, JP for japanese language.

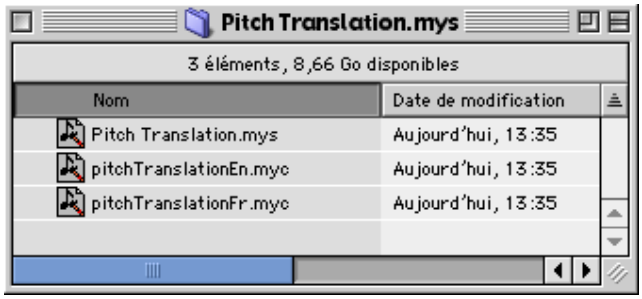
Example of use :
--NAME: Portamento
--CREATOR: Olivier Guillion - Myriad Software
--VERSION: 1.0
--DATE: December, 2002
--DATE-FR: Decembre, 2002
--ABSTRACT: Adds a portamento effect to selected notes
--ABSTRACT-FR: Ajoute un effet de portamento aux notes selectionnees
--DIFFUSION_MODE: 1
--INFO: Portamento is a smooth transition (pitch slide) between notes.
--INFO: This scripts uses invisible HAMMER and BEND ornaments to do the job.
--INFO: Effect on chords is unpredictable :)
--INFO-FR: Le portamento est une transition douce des frequences entre les notes
--INFO-FR: Ce script utilise des effets HAMMER et BEND.
--INFO-FR: Le resultat sur des accords peut etre non conforme.

1.3.10. Hiding a script on a given type of computer

If your script is totally platform-dependent (for example Macintosh only) you can avoid it to appear on other kind of computers by using the
PC_ONLY or MAC_ONLY tag in your file header.

1.3.11. Packaged scripts

You can define a set of scripts files that are designed to run together. This is called a **packaged** script. This enables to split a complex script into several parts.
A packaged script is a folder including multiple scripts and Interface Composer files. The folder name ends with the extension "mys".
It must include a script file with the same name. This script will be considered as the main script.
For example, here is a packaged script made of three files: a script file and two Interface Composer files :



The script will be displayed under the name "Pitch Translation" in the *Scripts* menu.
When you want to send a packaged script to somebody, the best way is to create an archive. Load the main script file and click on the icon. You will create a MyrScript Archive file with the **.msa** extension. This archive includes all files present in the folder. When another user opens this archive (for example using the *"Script>Open..."* menu) the archive will be automaticaly installed at the right location.

1.3.12. Adding scripts to standard menus

A script can appear in some standard menus of the application. In this case, the script will appear twice, once in the Scripts menu, and once at the required location.
Just add the tag:
"--MENU_LOCATION:xxxxx" in your script, xxxxx being one of these string:

"File>Export"	"File>Import"	"File>Info"
"File>Print"	"Edit"	"Edit>Select"
"Edit>Search"	"Edit>Bars"	"Edit>Appearance"
"Edit>Pitch"	"Edit>Quantize"	"Edit>Actions"
"Edit>Mirror"	"Edit>Multi-Voices"	"Edit>Digital audio track"
"Edit>Free objects"	"Score>Page mode"	"Score>Fast Transpose"
"Score>Apply"	"Staff"	"Instruments"
"Options"	"Configuration"	"Windows>Zoom"
"Windows>Palettes"	"Windows>Custom tool"	"Windows>Custom tools"
"Windows>Mixer"		

For example, "--MENU_LOCATION: File>Export" will add your script to the File>Export menu.

Item in the menu is tagged with the symbol if it is a standard script and if it is an user script.

1.3.13. File importer

A script which import a foreign file format, can define itself as an "Importer".
This mean the user will be able to load this format using the File>Open.
Much more, the Load(...) (page 596) command will also use this script to create a document.

To define a script can act as an importer, add the line "--FileImporter" in the header of the script text. When called two global variables will be available:
"GlobalFileNameImport" provide the full name (including path) of the file to import, "GlobalScore" will provide the score to fill.
If the script can't load this file because the file is not the right format, it must return the value -1. This mean Harmony will try to open this file with another importer.
It is recommanded to make a quick check of the file format at the beginning and return -1.
If the file format is recognized, but an error occurs, or the user breaks return the 0 value or more simply quit using the exit() command.
If the file is correctly loaded return the 1 value.

For example, here is the beginning of a script which import a GuitarPro file format :

```
-- Specify the script is an importer
--FileImporter
-- Check if a file name is provided by the caller
if GlobalFileNameImport then
    -- Checking the file extension is ok
    thePath,theName,theExtension=SplitName(GlobalFileNameImport)
    theExtension=strlower(theExtension)
    if theExtension~="gp4" and theExtension~="gp3" and theExtension~="gtp" then
        return(-1)
    end

   -- Open the progress bar, so the user will see the work in progress and can abort the operation
    ProgressBar.Open()
    -- Reading the file, return false if bad file format
    if ReadFile(GlobalFileNameImport)==false then
     return(-1) end
    -- Converting it to the score, return false if error
    if ConvertToScore(GlobalScore)==false then
     return(0) end
    -- Closing the progress bar
    ProgressBar.Close()
    -- Return the OK value
    return(1)
end
-- Asking the user to select a file
theFileName=GetFileNameLoad("gp4,gp3,gtp")
if theFileName ~=nil then
   -- Open the progress bar, so the user will see the work in progress and can abort the operation
   ProgressBar.Open()
   -- Reading the file
   ReadFile(theFileName)
   -- Converting it to the score
   ConvertToScore()
   -- Closing the progress bar
   ProgressBar.Close()
end
```

You can specify the file format name and the extension list handled by the importer. This will act as a filter when the user will apply File>Open.
For example :
--FileImporter Tabledit (*.tef) mean: the Tabledit format support files with .tef extension
--FileImporter Encore (*.enc), Rhapsody (*.rhp), MusicTime (*.mus) mean: this importer 3 differents files format.
--FileImporter GuitarPro (*.gp4,*.gp3,*.gtp) mean: this importer handle a file format with 3 differents extensions.

Please note:
The extensions are always between parenthesis.
Different files format are splitted by ","

1.3.14. Keyboard shorcuts

User can define keyboard key to launch a script.

1.3.15. Searching for scripts

From the *Scripts* menu you can search all source files. Results are displayed in a log window (page 75).

1.3.16. Special considerations

Some sub-folders are not displayed in the *Scripts* menu :
- The "Includes" folders
- All folder with a name begining by "(". This enables to hide some scripts temporarily.
- Packaged scripts : they are displayed as a single script entry.

1.4. Editing scripts

1.4.1. Overview

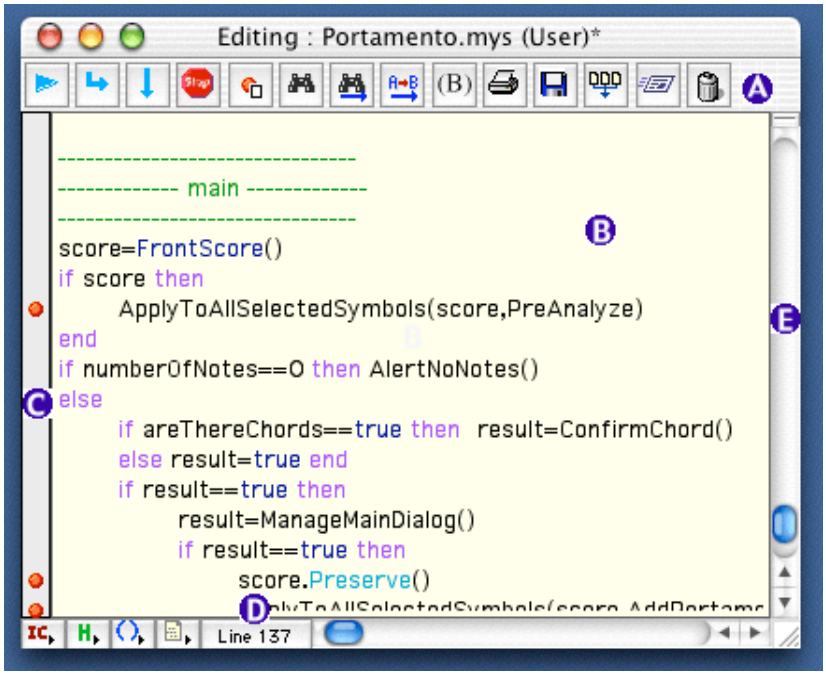
This chapter provides full information about how to create, edit, print, save, run or delete a script file.

1.4.2. What is a Script ?

A script is a file. More precisely a text file. You can edit a script file using any text editor but it is not recommended : the script file is "Macintosh" encoded.
A script **must** be located either in the program "Scripts" folder or in the user "Scripts" folder.
When you create a new script, it will automatically be saved in the user "Scripts" folder.
When you edit a predefined script, a copy of the file will be saved in your "Script" folder. Your script will be displayed in the *Scripts* menu in place of the original program script.

Never change, move or remove a predefined program Script.

Here is a preview of a script while editing :



A Icons enabling to run, debug, search, save, delete, archive, publish... the script file.
B Text currently edited.
C The breakpoints area.
D The placard.
E The scroll bar and the split window area. You can split verticaly the window in four areas.

1.4.3. Creating a new script file

In the *Scripts* menu, select *New*. A new unnamed script will be created into a new window. You can also start from an existing script using *Open* then saving the script under another name using *File>Save as*

1.4.4. Editing a script file

A script file is edited as an usual text file, using the keyboard arrows to move the text cursor, the mouse to select an area etc.
Here is the list of special keybard commands :

Home key : goes to beginning of line
End key : goes to end of line
Insert key: toggles "insert" mode. The blinking text cursor is bolder when "insert" mode is active
Backspace: deletes previous character
Suppr key: deletes next character
Tab: if a selection is active, shifts the whole selection range to the right by inserting a Tab before each line. If no selection, inserts a single Tab.
Shift+Tab: shifts the whole selection to the left by deleting a Tab on each line
Command+'+' : increases text size
Command+'-' : decreases text size
Command+'P' : prints the whole text
Command+'F' : searches the text for a pattern
Command+'G' : searches for next occurrence of pattern in the text
Command+'L' : replaces and searches next pattern in the text
Command+'S' : saves script on disk

Note :The "Command" key is the "Apple" key on Macintosh, "Ctrl" key on windows.

Here is the list of special mouse commands :

Double-click : selects the whole word
Triple-click : selects the whole line
Command+click on the selection: opens a pop-up menu displaying all the related topics. You can directly jump to this topic by selecting it.
The first item of the pop-up menu adds the selected keyword to the global variable window (see the debug (page 76) chapter)
Command+click in the title : opens a pop-up menu showing the file location. You can select an item to open the matching folder.

1.4.5. Autocomplete

Autocomplete (or word completion) allow you to write more easily scripts.
The editor will predict the end of the word you are currently typing.
This can be a member of an object defined in the MyrScript Manual, a member defined in your script, a function defined in your script...
When the proposal is done just press Right arrow key to validate it.
You can also display all possibilities in a contextual menu using Shift+Right arrow key.
As Lua is case sensitive, the autocomplete too.
Autcomplete can be disabled in Harmony Global Setup.

1.4.6. Printing a script

Click the printer icon. See the printing (page 74) chapter for a complete description.

1.4.7. Protecting a script

A protected script can be executed but not read without password. You can protect standard scripts, Interface Composer files or floating window scripts.

To protect/unprotect a script click on the padlock at the top of the edit window.
When a script is switched to protected state, a password is asked. Do not forget this password because it will be asked each time the source file is loaded.
(The password is not asked when a script is launched, only when the script is edited.)
A protected script source file is encrypted on disk to avoid the use of an external editor.

1.4.8. Undo/redo

You can undo/redo changes you applied to a file using the "Command+Z" and "Command+Y" keyboard shorcuts or the *Edit* menus.
Undo and redo must be enabled in the general preferences.

1.4.9. Starting from a predefined script

You cannot edit directly a predefined script but you can use a copy of it and apply changes. Select a predefined script in the menu with the Command key down.
After confirmation, the script will be copied and become an user script. You can now apply changes to it. In the menu, the user script is displayed in place of the predefined script.
When the predefined script is related to other files like Interface Composer files, pictures files or any data files, all these files are copied in the user area.

1.4.10. The placard

The placard is located at the window bottom left corner. There are four areas in the placard.

Interface Composer area

Displayed as a red "IC" . When clicked, the list of all Interface Composer files related to this script pops up(CreateDialog("file name") function).
Selecting a file name in this menu opens the matching Interface Composer file.

Include area

Displayed as a green "H" . When clicked, the list of all included files (Include "file name" function) related to this scr pops up.
Selecting a file name in this menu opens the matching file.

Functions area

Displayed as a blue "()" . When clicked, the list of all functions defined in this script pops up.
Selecting a function name in this menu shows this function.

Command area

Displayed as a little yellow icon. When clicked, a list of miscellaneous commands pops up.

Target area

Displayed as a little red icon. When clicked, a list of recent positions in the text pops up.
Selecting one of them jumps to it.

Line number area

Shows the current cursor position in line. Clicking it changes the current line.

1.4.11. Saving a script file

To save a script, you can use the icon at the top of the script window, or *File>Save*, or Command+'S' keyboard short

When you save a script, if the script file already exists, a backup copy of this file is made with the ".bks" extension. (This option must be activated in the global setup)

1.4.12. Scripts and user's palette

User can add a script in the custom palette by selecting the script with shift key down.
The script is displayed as a button. The content of the button is the name of the script.
A script can substitute this name by a picture : provide a .bmp file or a .pct file with the same name as the script and store it at the same level.

1.4.13. Creating a MyrScript Archive file

Overview

A MyrScript Archive file is a compacted binary file including one or more files. The extension for this kind of file is **.msa**.
Typically, you create an archive file when you want to send to somebody else a complete project including script files, Interface Composer files, custom pictures files, or any custom data files.
When a MyrScript Archive file is opened (for example using the *"Script>Open..."* menu), the files it contains will be automatically copied at the right location, creating folder and sub-folders if necessary.
For example, if you create a new script named "Ukulele.mys" and store it in "Scripts\Notation\Fretted\Ukulele.mys\Ukulele.mys" the path "Scripts\Notation\Fretted\Ukulele.mys\" will be created on the target computer.
If you create an archive from a script file **not** located in the "Scripts" folder, it will be installed in the "Scripts\Unsorted" folder.
If one or more files already exist at the same location on the target computer, confirmation is asked to the user before overwriting.

Creating an archive

When the main script file of a project is opened, the icon allow you to create a MyrScript Archive file including all tl project files.
The MyrScript Archive file name is asked, you can input the name you want. Please note, for Internet tranfer, a short name is recommended (less than 23 characters)

Archiving packaged scripts

When creating an archive of a packaged script (a script located in a folder with the same name) the archive will include all the files present in the folder and the sub-folder except:

- Files outside this folder are ignored. (For example files included with Include function)
- Backup copy (files with **.bks** or **.bkc** extension) are ignored.
- Archive files (files with **.msa** extension) are ignored.

Archiving unpackaged scripts

The main script file is archived plus files located at the same level and used in the main script file:

- Script definition file included with the Include function (only if they are at the same level)
- Script file launched with the Run function (only if they are at the same level)
- Interface Composer file created with the CreateDialog(...) (page 496) function (only if they are at the same level)

Because unpackaged scripts mix all files at the same level and can generate name conflicts, the use of packaged scripts is recommended.

*Note to Macintosh developers : When the archive is built, only the data fork of the file is archived. The resource fork stripped out. So, if you provide picture files with your project, always use the PICT file format and **not** the PICT resource file format.*

Archive file format

Here is the description of the archive file format. All values are Macintosh encoded (big endian encoding).

Archive header (not packed)
- · MAGIC_HEADER, 4 bytes 'MsAr' Magic
- · CHECKSUM, 4 bytes : file checksum : sum of all bytes following this checksum, up to the end of the file, including footer.
- · VERSION, 2 bytes : Archive version (currently 0x0100)
- · COMPACT_MODE, 1 byte : 0 if the archive is unpacked, 1 if the archive is packed (in relation with the packing mode defined in the general setup)
- · SIZEOF_MAIN_FOLDER_NAME, 1 byte : size of string describing the destination folder
- · MAIN_FOLDER_NAME, SIZEOF_MAIN_FOLDER_NAME bytes : string describing the destination folder

For each file (can be packed, depending of COMPACT_MODE):
- · SIZEOF_FILE_NAME, 1 byte : size of file name or 0 if no more file
- · FILE_NAME, SIZEOF_FILE_NAME bytes : string, file name
- · SIZEOF_SUBFOLDER_NAME, 1 byte : size of sub-folder name where the file is located, 0 meaning root of MAIN_FOLDER_NAME
- · SUBFOLDER_NAME, SIZEOF_SUBFOLDER_NAME bytes : string, subfolder name
- · SIZEOF_FILE, 4 bytes, length of unpacked data
- · FILE_DATA, packed data

Archive footer (unpacked)
- · MAGIC_FOOTER 4 bytes 'MsAr' Magic

1.4.14. Submitting a script

If you want to submit a script to be shared with other user or if you have a problem with a script and you want some advices from the Myriad team, just click the icon. You project will be archived in a temporary file and sent to our ema address.

1.4.15. Deleting a script file

To remove a script file, use the "trash" icon at the top of the script window. A confirmation is then asked. You cannot delete a predefined script.

1.4.16. Syntax balance

The syntax balance is an useful tool to check the syntax of your script. Click at any place in your script file then press the icon at the top of the script window. The level will be hightlighted.
You can also use the Command+'B' keyboard shortcut.
A tip: you can use the syntax balance to quickly indent your source : select a level, "balance it" and use the TAB key to increase tabulation or Shift+TAB to decrease tabulation of the whole selected level.

1.4.17. Syntax coloring

Syntax coloring enables to see quickly if the keywords you input are known or not.
Here is the meaning of the miscellaneous default colors.
Dark Green Comment. A comment always begins with '--'
Dark gray String. A string begins and ends with the "'" character
Purple Reserved keyword of Lua language
Dark Blue MyrScript global functions
Medium Blue Methods and fields
Red A syntax error. For example, a method not followed by (), a field not preceded by a dot,etc.

While debugging a script, variables names are colored :
Orange Local variable
Brown Global variable
See the debug (page 76) chapter for more information about debugging mode.

You can disable the syntax coloring mode using the placard (page 52) pop-up menus.
The status of syntax coloring is saved in the script file.

Syntax coloring enables to locate most of the syntax errors while inputing your script.
For example, you want to create a staff in a score. You write :

```
score.NewStaff
```

As you can see "NewStaff" is displayed in black, so it is not a keyword. You should have used "AddStaff" instead.

So you write :

```
score.AddStaff
```

"AddStaff" is displayed in red, this means it is a known keyword but with a bad syntax. In this case, because "AddStaff" is a method it must be followed by ( )

So, you write :

```
score.AddStaff()
```

The keyword "AddStaff" is displayed in blue, the syntax is correct.
Please note however, that syntax coloring does not check the parameters validity...
In the same way, it just checks whether a member name is known or not. It does not ensure this member is used on the right object type.

You can change the default colors in the General Setup, Scripts panel.

1.4.18. Running a script

To run a script, click the blue arrow icon at the top of the script window, or use the Command+'R' keyboard shortcut.
You can also select the script from the menu, but in this case breakpoints (page 76) are not taken into account.
You can stop a running a script by clicking the "Stop" icon in **any** of the opened script window.

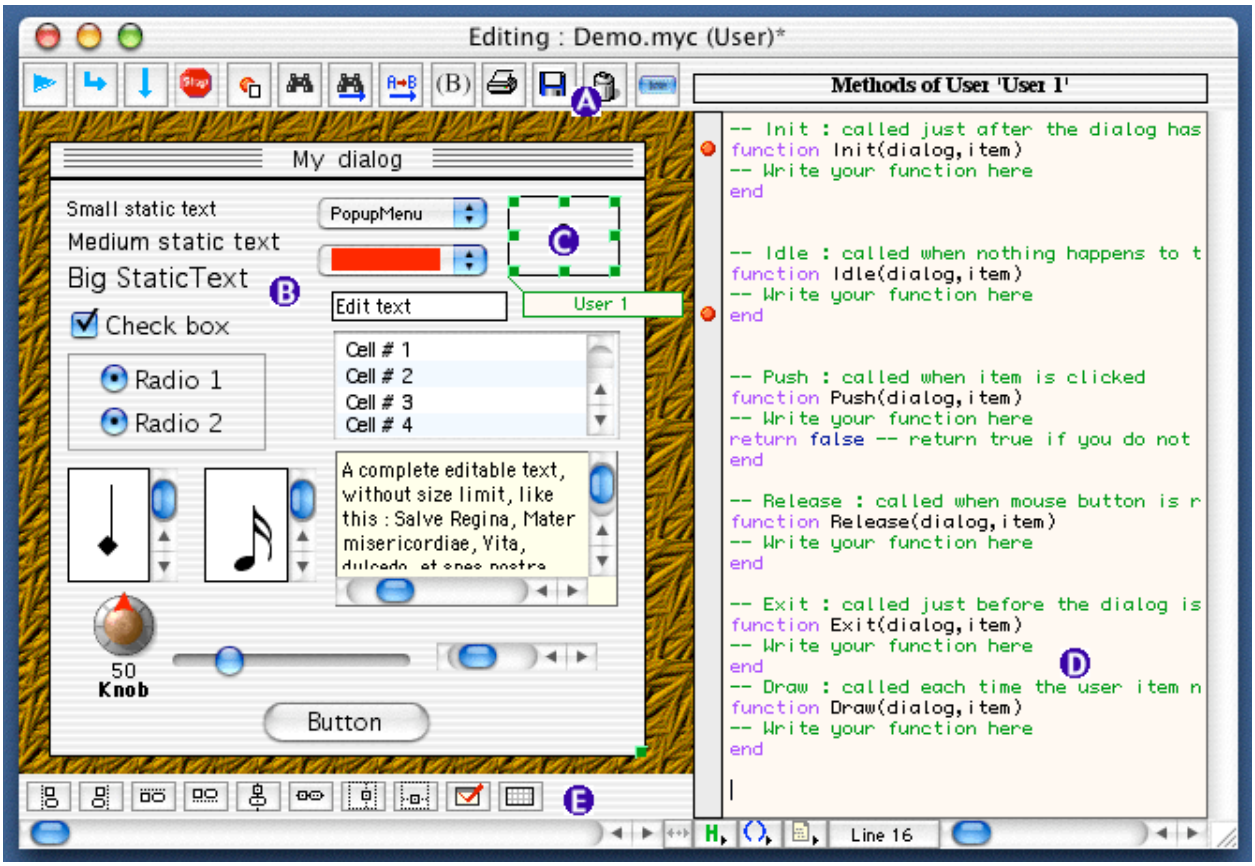
1.4.19. Adding and deleting breakpoints

Click in the left margin to add a breakpoint (red bullet). Click again to delete it.
You can delete all breakpoints by using the "Delete breakpoints" icon . More information about breakpoints is provide
in the debug (page 76) chapter.

1.5. The Interface Composer

1.5.1. Overview

When a script needs the user to input some values or setting, it can use a Dialog object.
Interface Composer enables to define graphically dialog boxes along with the miscellaneous actions to be applied when the user interacts with it. Interface Composer is a built-in module of MyrScript.
An "Interface Composer" file includes all object definitions and their related scripts.
Here is a preview of an Interface Composer Window :



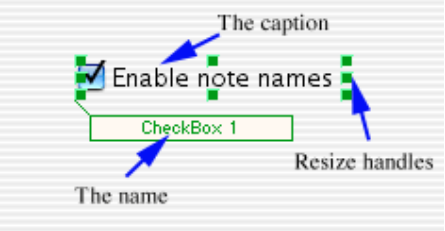
A Icons enabling to run, debug, search, save, delete... the Interface Composer file.
B Preview of the dialog box currently edited.
C Current item. Methods related to it are displayed in the right part of the window.
D Methods related to the current item.
E These buttons enable to align the selected items or to generate a report about the dialog box content.

1.5.2. What is Dialog Box ?

A "Dialog" object is a window with a title and **items** in it. A dialog box can be moved on screen by the user, by dragging its title bar.A dialog box can contain as many items as needed. A dialog box can open another dialog box, and so on. When the script quits, all dialog boxes created by the script are automatically closed. Dialog boxes created by other scripts are not affected.

1.5.3. What is an Item ?

An item is a graphical object that contains values. An item is always embedded into a dialog object.The kind of value stored in an item can change from an item type to another.
For example, a check box item (which can be checked or not) contains a boolean value. It can be true (the item is checked) or false (the item is not checked).



Your main script can change these values by using the dialog item values and methods. Scripts related to items can also change these values, or even other item values.
An item is defined by : its area in the dialog box, the displayed text (if any), its name, its current state... In a same dialog box, two items cannot have the same name, because the name is the only way for your script to identify an item.
When you add a new item to a dialog box, or paste items, a default name is given, but you can change it afterwards.
An item can be active or inactive, hidden or shown.

Note : Do not confuse between the text displayed in the item and the item name. The first one is purely graphical, the second is not shown to the user, and represents the string value the script can use to act on the item.

1.5.4. What is an Interface Composer file ?

An Interface Composer file is a text file with the extension **.myc**. You can load an existing Interface Composer file by using the *Scripts>Open* menu or using the IC icon in the main script placard: all Interface composer files used by the m script are displayed in a pop-up menu.
To create a new Interface Composer file, use *Scripts>New Interface Composer file*. A newly created dialog box includes two pre-defined objects: an "Ok" button named "Ok" and a "Cancel" button named "Cancel".
When loaded or created, the file is displayed in a new window. You can have as many open Interface Composer windows as you want.

1.5.5. Editing

The Interface Composer window is splitted into two parts. On the left, you can see a preview of the dialog box and command buttons. On the right you see the script related to the current selected item.
You can copy, paste, erase items using the standard keyboard shortcuts. The undo/redo commands are available while editing such a dialog box.

Important : this is only a preview. The appearance of the dialog box will change according to the Operating System on which the script is running.

Editing the dialog box

To **select an item**, click it. When selected, six little green marks wrap the item area. Below the current item, you can see a little floating area framed in green : this area shows the item name. You can double-click this area to change the item name.
To **select more than one item**, click with the shift key down or click outside an item and extend the select frame.
To **unselect an item**, click it again.
To **unselect all items** , click outside any item.
To **resize and item or the dialog box**, click the little green handle and extend the frame.
To **edit the text** displayed in an item, double-click it.
To **move an item**, drag it using the mouse. If several items are selected, all of them are moved at the same time.
To **delete an item** drag it outside the dialog box or use the Command+X keyboard shortcut. All Edit keyboard shorcut are available while editing a dialog box: copy, paste, erase. You can copy items or text from a Interface Composer window to another. You can undo or redo the changes.
To **align items**, select them using shift-click, then click the icons at the bottom of the edit window. You can also move the item, vertical and horizontal blue lines show you the possible alignments.
To **add a new item**, select the item type in the palette, then drag the item in the dialog box area with the mouse button down. The palette showing all item types opens automatically when an Interface Composer file is loaded or created. You can also open it using the New icon.
To **change the dialog box title**, double-click it.

Editing the script related to an item

When an item is selected, the script related to this item is displayed in the right part of the Edit window. You can edit it and add breakpoints.

Editing the script related to the dialog box

Click on the dialog box title in the preview, the script related to it will be displayed.

1.5.6. Saving an Interface Composer file

To save a an Interface Composer file, you can use the icon at the top of the script window, or *File>Save*, or Comman S' keyboard shortcut.
When you save an Interface Composer file, if the file already exists, a backup copy of this file is made with the ".bkc" extension. (This option must be activated in the global setup)

1.5.7. Deleting an Interface Composer file

To remove an Interface Composer file, use the "trash" icon at the top of the window. You are asked for a confirmation before proceeding actually.

1.5.8. Debugging

You can debug an Interface Composer dialog box. Use the icon or the Command+R keyboard shortcut.A preview of the dialog box is displayed.When running in debug mode, the "MSDefine (page 100).mys" file is automatically loaded, so that your methods can use any of the predefined constants.
You can click any item. The item name is displayed in the log window.To stop the running mode, use the "Stop" icon.
If breakpoint are added in a script related to an item, they will be taken into account. You can step into a method.
If you press the 'ESC' key while a dialog is running, execution is paused and the current line is displayed.

1.5.9. Opening a Dialog Box

To open an use a dialog box from a script, use the CreateDialog function. For example *myDialog=CreateDialog(" MyDialog.myc")* will create a new dialog box from the "MyDialog.myc" file and store the result in the *myDialog* variable .
The Interface Composer file must be at the same level on hard drive as the script file that uses it. It is recommended to use packaged scripts to avoid conflicts between two scripts which use the same Interface Composer file name.
Much more interesting, storing the script files and all Interface Composer files related to it in the same folder, enables to create easily MyrScript Archives (page 122).

1.5.10. Closing a Dialog Box

Simply use the Dialog.Dispose(...) (page 503) method. Please note if the script ends because of an error, dialog boxes ar automatically closed.

1.5.11. Methods

Methods are functions related to an object which describe the way the object will interact. Some methods are predefined (Init, Exit, Idle, Push, Release...) and are called when some events occur. For example, the Idle method is called several times per second, the Init method is called when dialog opens, etc...
You will in this chapter a precise description of methods.

To edit methods related to an item, click it. The script related to this item is displayed on the right of the window.
To edit methods related to the dialog itself, click its title.
You can add and remove breakpoints in methods.

Some tips :

· We saw there are default methods, but you can define you own methods in an object an call them.

So, in the script related to an object you can insert :

```
function myTest()
    print ("Test")
end
```

But, for example this will not work in a script related to an object:

```
function myTest()
    print ("Test")
end
function Idle(dialog,item)
    myTest()
end
```

You will get the following error:
error: attempt to call global `myTest' (a nil value)

Because **myTest** is a method which is owned by the item object and can't be called without the object reference.
You must write instead:

```
function myTest()
    print ("Test")
end
function Idle(dialog,item)
    item.myTest()
end
```

· For the same reason, you can't begin a script definition with :

```
    Include "MSDefine"
```

because this will never be called.

You have to write the "include" command into the Init method of the dialog (or in the Init method of an object) like this :

```
function Init(dialog)
    Include "MSDefine"
end
```

· But you can also call the method of an object from another object. For example, this script will call a method defined in an object called "Item1" from an object called "Item2"

In Item2 script:

```
function Idle(dialog,item)
    dialog.GetItem("Item1").myTest()
end
```

· if you want to declare global variables, the best way is to store them in the Dialog object "Init" method.
For example :

```
function Init(dialog)
   myVar=0
end
```

will declare a global variable nammed "myVar"

· A more complexe case is when you want to send a method to a function. This is used for example with some predefined functions in MSLibrary.

For example, the function "BrowseFolder" needs a function (a callback) which will be called each time a file is found.

The script:

```
function CallBackProcessFile(filename)
   print (filename)
end

function Release(dialog,item)
   BrowseFolder(folderName,"*.*",CallBackProcessFile,true)
end
```

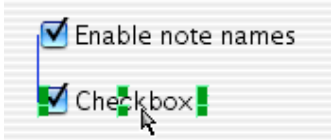
will not work.

You must write instead:

```
function CallBackProcessFile(filename)
   print (filename)
end

function Release(dialog,item)
   BrowseFolder(folderName,"*.*",function (fileName) return(GlobalDialog.CallBackProcessFile(fileName)) end,true)
end
```

1.5.12. Item alignment

To align items, select more than one item and use the icons at the bottom of the window : , , , , , ...
You can also move or resize the item : a vertical blue line shows the closest item alignment.



If "auto alignment mode" is active, the item is automatically aligned. You can setup the alignment accuracy or disable it using the icon.

1.5.13. Generating report

At the bottom of the window you can see an icon with a little frame checked : . By clicking it, your generate a report about your dialog, including a diagnosis about possible errors.

1.5.14. Optimizing

When you add a new item to a dialog, default empty methods are created for this item. Many of these methods will not be used and will remain empty.
This is particulary annoying for Idle methods. Idle methods are called **several times by second and can slow down the application**, especially when used in a floating window.
The icon enables to remove all unused methods from your dialog.

1.5.15. Items types

You will find here a description of the different item types you can add to a dialog box.

When a dialog box is opened, methods of the objects are called when the related event occurs. Some methods are present in all objects (Init,Exit,Idle...) some methods are specific to some item types (KeyDown).

All objects including items and dialog can have their own private methods.

Here is a description of the methods :

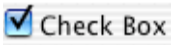
Method name	Method call	Objects	Called when
Init	Init(dialog,item)	All	One time, when the dialog box opens.
Exit	Exit(dialog,item)	All	One time, when the dialog box closes.
Idle	Idle(dialog,item)	All	Very regularly, when no event happens.
Push	Push(dialog,item)	Items	When mouse button is down on the user area.
Release	Release(dialog,item)	Items	When mouse button is released.
KeyDown	KeyDown(dialog,item,key)	Text item, Edit text item, Dialog	When a key is pressed on keyboard and the item has the focus the item's method is called, then the dialog's method is called.
Draw	Draw(dialog,item)	User item or Dialog	The Draw method is called to draw the item content. In a dialog, this method is called before any other Draw method.
Track	Track(dialog,item)	Scrollbar item, Slider_item, Knob item	Called repetitively while user moves the thumb or clicks on an arrow.

Button item type

Preview :

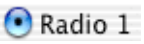
Description : A push button is a rounded rectangle that is labeled with text. Push buttons usually perform instantaneous actions, such as completing
operations defined by a dialog box or acknowledging an error message. Clicking a push button initiates the action described in the button caption.
For example, a dialog box should have a "Ok" button and a "Cancel" button.
Caption : This item has a caption and you can change it using DialogItem.Caption (page 508).
Value : This item does not have a value.
Extra values : This item does not have any extra value.
Methods : This item supports methods: Init, Exit, Idle, Push, Release.

Checkbox item type

Preview : 

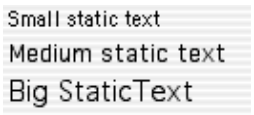
Description : A checkbox is displayed as a little frame that can be checked or not, followed by a displayed string. The user can change the state of a checkbox by clicking on it.
Caption : This item has a caption and you can change it using DialogItem.Caption (page 508) value.
Value : Boolean, true if selected, false otherwise.
Extra values : This item does not have any extra value.
Methods : This item supports methods : Init, Exit, Idle, Push, Release.

Radio item type

Preview : 

Description : A radio button is displayed as a little circle followed by a string. A radio button is never alone in a dialog but always included in a group of radio buttons. To define a group, just frame the radio buttons with a FrameRect object. To activate a radio button, the user can click the button itself or its text label. Only one button in a group can be "on" at the same time. The "on" state is indicated by a dot in the middle of the button. Activating a radio button turns off any button that was previously on in that group.
Caption : This item has a caption and you can change it using DialogItem.Caption (page 508) value.
Value : Boolean, true if selected, false otherwise.
Extra values : This item does not have any extra value.
Methods : This item supports methods : Init, Exit, Idle, Push, Release.

Static Text item type

Preview : 

Description : A statText is a static text : a text the user cannot edit. Three text sizes are available : small, medium and big. Generally the user does not have to interact with it, but if you want, you can apply operations when the user clicks it. However this is not recommended.
Caption : This item has a caption and you can change it using DialogItem.Caption (page 508) value.
Value : This item does not have a value.
Extra values : This item does not have any extra value.
Methods : This item supports methods : Init, Exit, Idle, Push, Release.

Edit Text item type

Preview :

Description : A EditText is an editable text : a text the user can edit. To deferenciate a StatText from an EditText, the EditText is framed. The maximum string length is 240 characters. If you want the user to edit a complete text, without limit in size, use a Text item type (page 68) instead.
Caption : This item type does not have caption.
Value : String, edit text current content.
Extra values : This item does not have any extra value.
Methods : This item supports methods : Init, Exit, Idle, Push, Release, KeyDown.

Pop-up Menu item type

Preview :

Description : A pop-up menu button displays a list of items the user can select. The left side of the button contains text that shows the current selection. When the user clicks it, a menu opens and the user can change this current value. The different values must be defined in the Init method of the object.
Caption : This item type does not have caption.
Value : Number, selected menu item number from 1 to DialogItem.NumberOfExtraValues (page 510).
Extra values : Indexed strings, menu item titles. If the string begins with the character '(' the item is dimmed (not selectable)
Methods : This item supports methods : Init, Exit, Idle, Push, Release.

Color item type

Preview :
Description : A color item displays the currently selected color. When the user clicks it, a palette opens and he can select a color. A color varies from 0 to 255.You can use the IndexToRGB and IndexToRGB graphic functions to convert back and forth from an index to the RGB (reg-green-blue) color.
If the user shift-clicks this kind of item, the color list is restricted to 16 "standard" colors.
Caption : This item type does not have caption.
Value : Number, selected color index from 0 to 255, 0 meaning default color.
Extra values : This item does not have any extra value.
Methods : This item supports methods : Init, Exit, Idle, Push, Release.

Style item type

Preview :
Description : A style item is displayed as a little blue "A". When the user clicks it, he can select font, size, face and color.
Caption : This item type does not have caption.
Value : TextStyle (page 414), selected text style.
Extra values : This item does not have any extra value.
Methods : This item supports methods : Init, Exit, Idle, Push, Release.

HeadShape item type

Preview :

Description : A HeadShape object displays a list of note head shapes. The user can select a shape by using the related scroll bar.
Caption : This item type does not have caption.
Value : Number, selected head shape (see constants in MSdefine (page 100)).

SHAPE_NORMAL=0
SHAPE_TRIANGLE=1
SHAPE_DIAMOND=2
SHAPE_CROSS=3
SHAPE_PLUS=4
SHAPE_SQUARE=5
SHAPE_BEAM=6
SHAPE_NONE=7
SHAPE_FLAG=8
SHAPE_HALF_CIRCLE=9
SHAPE_DROP=10
SHAPE_SAND_GLASS=11
SHAPE_ARC=12
SHAPE_EMPTY_DIAMOND=13
SHAPE_CONE=14
SHAPE_CRESCENT=15
SHAPE_LEFT_TRIANGLE=16
SHAPE_BELL=17
SHAPE_CONCAVE_TRIANGLE=18
SHAPE_CONVEX_TRIANGLE=19
SHAPE_JOHNSON_TRIANGLE=20
SHAPE_JOHNSON_SQUARE=21
SHAPE_CIRCLE_BOLD_CROSS=22
SHAPE_BOLD_CROSS=23
SHAPE_THIN_CROSS=24
SHAPE_HALF_CIRCLE_THIN_CROSS=25
SHAPE_EMPTY=25

Extra values : This item does not have any extra value.
Methods : This item supports methods : Init, Exit, Idle, Push, Release.

Duration item type

Preview :

Description : A Duration object displays a list of note duration. The user can select a duration by using the related scroll bar. You can define the list of note duration using the Init method of this object.
Caption : This item type does not have caption.
Value : Number, selected duration.
Extra values : This item has extra values from 1 to DialogItem.NumberOfExtraValues (page 510). Each extra value is duration.
By default a standard set of durations is defined, but you can change it.

For example, here is a custom init that defines a duration selector with three value : DURATION_QUARTER, DURATION_HALF,DURATION_WHOLE, the DURATION_HALF duration being the default value.

```
-- Init : called just after the dialog has been displayed for the first time
function Init(dialog,item)
item.DeleteExtraValues()
item.SetExtraValue(1,DURATION_QUARTER)
item.SetExtraValue(2,DURATION_HALF)
item.SetExtraValue(3,DURATION_WHOLE)
item.Value=DURATION_HALF
end
```

Methods : This item supports methods : Init, Exit, Idle, Push, Release.

Scroll bar item type

Preview :

Description : A scroll bar object is displayed as two arrows and a thumb area. The scroll bar can be vertical or horizontal.
The thumb position shows the current value. By default, the scroll bar value varies from 0 to 100 but you can change the minimum and maximum values in the Init method of this object for example.
Caption : This item type does not have caption.
Value : Number, current scroll bar value from DialogItem.GetExtraValue(1) to DialogItem.GetExtraValue(2).
Extra values :

This item type has two extra values :
1 : Minimal value the item can provide. (by default 0)
2 : Maximal value the item can provide. (by default 100)

For example to change the minimal and maximal value to [10,1000] you can write :

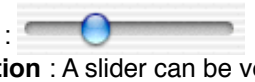
```
item.SetExtraValue(1,10)
item.SetExtraValue(2,1000)
```

To get the current value use for example :

```
currentValue=Item.Value
```

Methods : This item supports methods : Init, Exit, Idle, Push, Release, Track.

Slider item type

Preview : 
Description : A slider can be vertical or horizontal. The thumb position shows the current value. By default, the scroll bar value varies from 0 to 100 but you can change the minimum and maximum values in the Init method of this object for example.
Caption : This item type does not have caption.
Value : Number, current slider value from DialogItem.GetExtraValue(1) to DialogItem.GetExtraValue(2).
Extra values :

This item type has two extra values :
1 : Minimal value the item can provide. (by default 0)
2 : Maximal value the item can provide. (by default 100)

For example to change the minimal and maximal value to [10,1000] you can write :
item.SetExtraValue(1,10)
item.SetExtraValue(2,1000)

To get the current value use for example :
currentValue=Item.Value
Methods : This item supports methods : Init, Exit, Idle, Push, Release, Track.

Knob item type

Preview :

Description : This enables the user to select a value through knob.
Caption : This item has a caption and you can change it using DialogItem.Caption (page 508) value.
Value : Number, current knob value from DialogItem.GetExtraValue(1) to DialogItem.GetExtraValue(2).
Extra values :

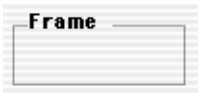
This item type has two extra values :
1 : Minimal value the item can provide. (by default 0)
2 : Maximal value the item can provide. (by default 100)

For example to change the minimal and maximal value to [10,1000] you can write :
item.SetExtraValue(1,10)
item.SetExtraValue(2,1000)

To get the current value use for example :
currentValue=Item.Value

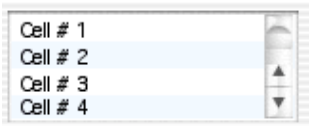
Methods : This item supports methods : Init, Exit, Idle, Push, Release, Track.

Frame item type

Preview : 

Description : A frame rectangle is a frame. It can be purely graphic, but when a frame rectangle includes radio buttons, it then defines a group of radio buttons. A name can be related to the frame.
Caption : This item has a caption and you can change it using DialogItem.Caption (page 508) value.
Value : DialogItem, selected radio button in this group (if any).
Extra values : This item does not have any extra value.
Methods : This item supports methods : Init, Exit, Idle, Push, Release.

List item type

Preview : 

Description : This is a list of cell. Each cell shows a string. The user can select one of them. A scroll bar is automatically related to the list to allow the user to scroll into the list.
Caption : This item type does not have caption.
Value : Number, selected cell in the list from 1 to DialogItem.NumberOfExtraValues (page 510).
Extra values : Indexed strings, list cell titles.
You can release all extra values using the DialogItem.DeleteExtraValues(...) (page 515) method.

Example:

```
-- Init : called just after the dialog has been displayed for the first time
function Init(dialog,item)
-- Filling the list with strings value
   for i=1,20 do
      item.SetExtraValue(i,"Choice "..i);
   end
end

-- Release : called when mouse button is released
function Release(dialog,item)
--- When an element is selected we display its value
print(item.GetExtraValue(item.Value))
end
```

The list object type supports special select mode. By default, when the user clicks on a cell, the current cell is deselected and the new cell is selected.
You can define, for example, another select mode using :
item.SetExtraValue(0,"SelectionMode",mode)
the "0" index value mean : "for all the list"
"SelectionMode" is the keyword to use
"mode" determines how selection is performed. This value can be a combination of constants defined in MSDefine :

- LIST_EXTENDDRAG= 64 Drag extends without shift key down
- LIST_NODISJOINT = 32 Click deselects all previous selection
- LIST_NOEXTEND=16 Shift won't extend selection
- LIST_NORECT=8 Don't expand selection as a rectangle
- LIST_USESENSE=4 Shidt senses state of initial cell
- LIST_NONILHILITE=1 Don't highlight empty cell
- LIST_MULTIPLE_SELECT=LIST_EXTENDDRAG+LIST_USESENSE The user can select more than one cell
- LIST_SINGLE_SELECT=0 The default mode

When multiple select is active, the GetExtraValue method, applied to one cell, provides two values : the string in the cell, the state of the cell (a Boolean, selected or not)
For example :
str,select=item.GetExtraValue(index)
Methods : This item supports methods : Init, Exit, Idle, Push, Release.

Tab View item type

Preview :

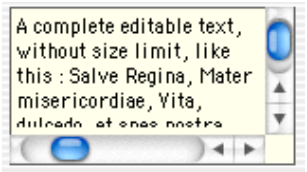
Description : This is a list of selectable pannel. Each pannel shows a string. The user can select one of them.
Caption : This item type does not have caption.
Value : Number, selected pannel in the tab view from 1 to DialogItem.NumberOfExtraValues (page 510).
Extra values : Indexed strings, list pannel titles.
You can release all extra values using the DialogItem.DeleteExtraValues(...) (page 515) method.

Example:

```
-- Init : called just after the dialog has been displayed for the first time
function Init(dialog,item)
-- Filling the list with strings value
   for i=1,4 do
      item.SetExtraValue(i,"Choice "..i);
   end
end

-- Release : called when mouse button is released
function Release(dialog,item)
--- When an element is selected we display its value
print(item.GetExtraValue(item.Value))
end
```

Methods : This item supports methods : Init, Exit, Idle, Push, Release.

Text item type

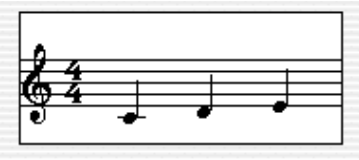
Preview : 

Description : This is a full text without size limit. Two scroll bars are automatically related, enabling the user to scroll into the text.
When a key is pressed on computer keyboard, and the focus is set to the text, the "KeyDown" method of this object is called.
The object can process this key down event and return the value "false" to specify that the standard key down processing won't be performed.
For example, you can use this "KeyDown" method to apply filters on characters.
Caption : This item type does not have caption.
Value : String, the content of the text.
Extra values : One extra value : Text (page 578) object that enables to apply operations on the content. You can only read this extra value.
Methods : This item supports methods : Init, Exit, Idle, Push, Release, KeyDown.

User item type

Description : This item is drawn using the "Draw" method of the object. When this method is called, the drawing area is clipped to the item area (you cannot draw outside the item) and the origin is set to the item area (coordinate 0,0 is the top left of the item's area). The item area is erased before the Draw method is called. You have to use the Graph (p ge 516) methods to draw inside the user item area.
Caption : This item type does not have caption.
Value : This item has no value.
Extra values : This item does not have any extra value.

For example we will define methods of an user item which display a staff with three notes :

Preview :

```
-- Init : called just after the dialog has been displayed for the first time
function Init(dialog,item)
-- Creating new score
item.UserTable["Score"]=NewScore(false)
score=item.UserTable["Score"]
if(score~=nil) then
   item.UserTable["Staff"]=item.UserTable["Score"].AddStaff()
   ---Creating new staff in score
   staff=item.UserTable["Staff"]
   if(staff~=nil) then
      staff.InsertNote(0,DURATION_QUARTER,5*12)
      staff.InsertNote(DURATION_QUARTER,DURATION_QUARTER,5*12+2)
      staff.InsertNote(2*DURATION_QUARTER,DURATION_QUARTER,5*12+4)
      end
   end
end
-- Draw : called each time the user item need to be redrawn
function Draw(dialog,item)
   Graph.FrameRect(0,0,item.AreaWidth,item.AreaHeight)
   Graph.DrawStaff(0,0,item.AreaWidth,item.AreaHeight,item.UserTable["Staff"])
end
-- Exit : called just before the dialog is closed
function Exit(dialog,item)
-- Disposing score
   item.UserTable["Score"].Dispose()
end
```

Methods : This item supports methods : Init, Exit, Idle, Push, Release, Draw.

1.6. Floating windows

1.6.1. Overview

A floating window is a special window kind you can create.
We saw in previous chapters that a script can open and close a dialog box defined in an Interface Composer file.
When a script open a dialog box, the dialog box is modal. This mean the user cannot perform any other action, like opening a music file or launching another script, until the script ends.
These limitations are not true with floating windows:

- Several floating window can be opened at the same time
- When a floating window is open, all actions remain possible

From many point of vue, floating windows are very similar to application palettes.

1.6.2. What is a Floating window ?

A floating window is a window with a title and **items** in it. A floating window can contain as many item objects as needed, of any type except Edit text items.

You can consider that a floating window is a stand alone Interface Composer dialog object which can be used without a script.
All items type you can use in an Interface Object dialog can be used too in a floating window.

A floating window is stored in a file with the **myf** extension.
Floating window are displayed in the *"Script"* menu. Floating window provided with the program are tagged with , user floating window with .
As scripts, floating window can be packaged : just store the floating window main file in a folder with the same name.

1.6.3. How to create a Floating window ?

In the *"Script"* menu select "New floating window". This will open the Interface Composer editor.
All operations we saw on Interface Composer editing are available.
You can comment a floating window in the same way you comment a script file.

1.6.4. Special considerations

Because the floating window will be run without the need of an external script (remember it is a stand-alone Interface Composer dialog), all initializations must be performed in the "Init" method of the dialog object.
For example, if your floating window needs to use some constants or functions defined in an predefined include file, the "Init" method of the floating window will be something like :

```
function Init(dialog)
Include "MSDefine"
Include "MSSounds"
end
```

If you define a local function in a floating window, you will be able to call it as a new method.
For example, this function :

```
function DisplayContent(value)
 ...
end
```

declared in the dialog object body.
To use it you will have to write : "dialog.DisplayContent(1)".

1.6.5. High-level events

Some high-level events are automatically sent to all open **floating windows**. This enables to perform special actions when they occur.
For example, a high-level event is sent when music starts or stops, or when the score is printed.
Methods defined in the main script of the dialog object are called. Some of these methods can return a boolean value.
If the result is "false", the high-level event is handled exclusively by the script and not processed by the program.
If no result value is provided, the value "true" is returned.

Method name	Method call	Called when	Can break the event
OnStartPlay	OnStartPlay(dialog,score)	Called when music starts.	Yes
OnStopPlay	OnStopPlay(dialog,score)	Called when music stops.	No
OnStartPrint	OnStartPrint(dialog,score)	Called when the score will be printed.	Yes
OnStopPrint	OnStopPrint(dialog,score)	Called when printing process ends.	No
OnInsertSymbol	OnInsertSymbol(dialog,score,symbol)	Called after a symbol is inserted.	No
OnWillInjectSymbol	OnWillInjectSymbol(dialog,score,symbol)	Called before a symbol will be changed through the injector tool.	Yes
OnInjectSymbol	OnInjectSymbol(dialog,score,symbol)	Called after a symbol has be changed through the injector tool.	No
OnInjectScore	OnInjectScore(dialog,score,staff,xPosition,yPosition,timePositionw)	Called when the inject tool is clicked outside an object.	Yes
OnLoad	OnLoad(dialog,score)	Called after a file is loaded.	No
OnViewsChange	OnViewsChange(dialog,score)	Called after views are edited.	No
OnViewChange	OnViewChange(dialog,score)	Called after the current view is changed.	No
OnMouseWheel	OnMouseWheel(dialog,score,amount)	Called each time the mouse wheel is moved.	No
OnScoreChange	OnScoreChange(dialog,score)	Called when a change has been made to the score by the user. If your script keeps score objects in global variables, you have to	No

Important: When you debug a floating window, High-level events are NOT called.

For example, this enables to create a temporary staff each time music starts, and to delete it when music stops.
--- Called when music starts

```
function OnStartPlay(dialog,score)
   -- We create a staff in the score
   staff=score.AddTemporaryStaff()
      if staff ~= nil then
      -- We insert notes on it
         for i=0,20 do
            staff.InsertNote(DURATION_QUARTER*i,DURATION_QUARTER,5*12+i)
         end
      end
   return true
end
--- Called when music stops
function OnStopPlay(dialog,score)
   score.DeleteTemporaryStaves()
end
```

The OnInjectScore high-level event is called each time the user click on a staff with the inject tool outside any symbol. The event receive the dialog, the score, the staff, the absolute pixel position (you can convert it written time position using score.AbsolutePixelToWrittenTime), the relative y position to the staff base line.

Here is an example of a floating window which add a black circle as a picture when the user click on a staff.

```
function OnInjectScore(dialog,score,staff,xPosition,yPosition)
   staff.BeginRecord(32,32)
   Graph.PenNormal()
   Graph.PaintOval(0,0,32,32)
   staff.EndRecord(xPosition,yPosition,true)
   Application.UpdateScores()
   return false
end
```

1.6.6. Floating window from the user point of view

When the user opens one (or more) floating window, he can move it on screen, click its items, etc...
The floating window will respond to the user actions.
When the floating window uses an "text edit" object and the user hits a key on keyboard, the frontmost floating window will receive this event.
Your floating window can run in background, using the items or dialog "Idle" methods which are called regularly by the program.

1.6.7. Multiple Floating windows consideration

We saw the user can open as many floating window as he wants, and keep them open while he works. This means he can open many times the same floating window. You can, if you want, prevent the user to open twice the same floating window: add the text : "--MonoInstance" somewhere, for example at the beginning of the text related to the dialog.

1.6.8. Palette or dialog

When a floating window does not include any "text edit" item, it acts as a palette. A palette is always displayed in front of all document windows.
If you do not want your floating window to behave like a palette, insert the text "--NoPalette" somewhere, for example at the beginning of the text related to the dialog.

1.6.9. Translucent ratio

On Windows, a palette can be translucent. You can define the transluscent ratio by inserting :
"--TransluscentRatio xxx" where xxxx is the transluscent ratio. A ratio of 0 mean "opaque".

1.6.10. Optimizing

The idle methods being called for each item several times per second, it is recommended to remove unused methods.

1.7. The Search Tool

1.7.1. Overview

This tool enables to search and substitute strings in text windows or files.
There are different ways to open the search tool dialog box :

- By clicking the icon in an edit text window.
- By using the Command+F keyboard shortcut.
- By using the "Search" command in the *Scripts* menu.

1.7.2. Searching

You can make a **case sensitive search** and/or a **whole word** search.
You can search the front window only, all opened windows, all files, all files including the predefined scripts.
If you search using the "Find All" command, results will be displayed into a text window. By clicking on a line, the related line in the script file is displayed.

1.7.3. Searching again

When you do not use the "Find all" command, the first found occurence is displayed. To jump to the next one, use the Command+G keyboard shortcut or the icon.

1.7.4. Replacing

To replace a string by another one, input the search string and the replacement string in the dialog box, then select " Replace" or "Replace all".
When a replacement string is defined and you apply a "Find" command, you can substitute the currently found target through the "Replace" command or by using the Command+L keyboard shortcut. The next target is automatically found.
To skip to the next target without replacing, use Command+G keyboard shortcut.

1.7.5. Fast Search

To search quickly for a word, click it with Command key down (or right mouse button on PC) and select "Search this..." in the menu.

1.8. Printing texts

1.8.1. Overview

You can print either a script or this whole manual. A printing preview is displayed if the option is not disabled in the global setup.
Use the Command+'P' keyboard shortcut or the matching command in the placard (page 52).

1.8.2. Selecting a printing scale

There are different ways to change the size of text on paper. You can edit the printer setup (File menu) and adjust the printing scale of your printer driver.
You can increase/decrease the text size using the Command+"+" and Command+"-" or choosing a new text size in the placard (page 52) related to the window.

1.8.3. Syntax coloring

If syntax coloring is active and the window is a script window, text is printed in color. You can deactivate syntax coloring using the placard (page 52) popup menu.

1.9. The Log Window

The log window is a text window. There is only one log window open at the same time. Each time a script is launched, the log window is erased.The log window records the 1000 last lines. When this limit is reached, the first line is erased.The log window will receive :

- All the results of MyrScript *print* function
- The errors generated by a script. Clicking on the line will open the matching script source file or Interface Composer file.

Please note you can define you own text window, using the TextWindow (page 560) object type.

1.10. Debugging a script

1.10.1. Overview

The best way to locate problems in a script and to fix them is to use the debug mode.In debug mode, you can set breakpoints to stop the script execution, running in step-by-step mode, show local and global variables.

1.10.2. Debugging Interface

At the bottom of the script window, you can see some icons that are used in running and debugging mode.
When you are running in step-by-step mode a script that applies changes to a score, the score window is automatically updated on screen but you cannot apply any change manually to the score window while the script is running.

Here is a description of the miscellaneous commands :

Run Icon

Keyboard shortcut for this command is Command+R.
The Run icon starts script execution. If a breakpoint is encountered, the script is halted and the current line is displayed.
Variable window is automatically updated, showing the current local variable values and the global variable values you added to the variable window.
You can break the script execution by hitting ESC key on keyboard. Current line is then displayed. After this, you can restart exeuction.

Step into Icon

Keyboard shortcut for this command is Command+D.
This icon executes one line. If the script calls a function, the step process continues in this function.\
If the script includes another script file (with the *Include* function) the new script is loaded into another window.

Step outside Icon

Keyboard shortcut for this command is Command+E.
This icon executes one line. If the script calls a function or includes another script file, the function or the external script is not shown.

Stop Icon

This command ends the script execution. The current MyrScript status is displayed in the log window. After this yo cannot continue the script execution.

Delete all breakpoints Icon

This command deletes all breakpoints that have been set in the script.

1.10.3. Breakpoints

Breakpoints are displayed in the script left margin as a little red bullet. You can set or remove a breakpoint by clicking in the margin.If lines are inserted or deleted in the script, breakpoints are moved accordingly. Breakpoints are saved along with the script file itself.

Note : if you run a script from the menu, breakpoints are not taken into account.

1.10.4. Data browser

The variable window shows the current type and value of all local variables (variables that have been defined using the *local* keyword) and global variables you added to the variable window. You can edit the text displayed in the variable window to change a local variable value or add a new global variable. (You cannot change the value of a global variable).

To add quickly a global variable, double-click its name in the script window with the Command (Ctrl) key down.
To remove, add or delete all global variables, Command (Ctrl) + click on the Variable windows.
When a variable owns several members a "+" is displayed in front of this name. Click it show all members values. The "-" symbol hide the members.

1.10.5. Performances

Due to breakpoints and other internal management, a script runs from 50% to 75% slower in debug mode.

1.10.6. Crashes

All the MyrScript methods and functions are protected against illegal data entries, but in some case a script can crash the program.So, before sharing a script with other users, please check it in depth...
For example, here is a script with errors :

```
local score
-- Getting front score
score=FrontScore()
-- First error : using score without testing if it is valid
if(score.Dispose()==true) then
-- Second error : using score after disposed it.
print("The score : "..score.PathName..score.FileName.." is now closed.")
end
```

The script should provide a simple runtime error but can also crash the program in some cases.

1.11. The Lua language

1.11.1. Overview

The core of MyrScript is **Lua** : a free open source language. We have been amazed by the power, the speed and the stability of this library. Hats off to Mr R. Lerusalimschy, L. H. de Figueiredo and W. Celes from the **Lua Team** for this wonderful work !

You will find in this chapter all the essential information about the **Lua** language syntax.

If you want more information about **Lua**, including a full documentation please go to : http://www.lua.org/

You can download the complete Lua manual in PDF format here :
"Reference manual of the programming language Lua 4.0", PDF format : http://www.lua.org/ftp/refman-4.0.pdf

1.11.2. Types and Tags

Lua is a dynamically typed language. This means that variables do not have types; only values do.
Therefore, there are no type definitions in the language. All values carry their own type. Besides a type, all values also have a tag.
There are six basic types in **Lua**: nil, number, string, function, and table.

·Nil is the type of the value nil, whose main property is to be different from any other value.

·Number represents real (double-precision floating-point) numbers, while string has the usual meaning. **Lua** is 8-bit clean, and so strings may contain any 8-bit character, including embedded zeros ('\0').

·The type function returns a string describing the type of a given value.
Functions are considered first-class values in **Lua**. This means that functions can be stored in variables, passed as arguments to other functions, and returned as results.
The type table implements associative arrays, that is, arrays that can be indexed not only with numbers, but with any value (except nil). Therefore, this type may be used not only to represent ordinary arrays, but also symbol tables, sets, records, graphs, trees, etc. Tables are the main data structuring mechanism in **Lua**. To represent records, **Lua** uses the field name as an index. The language supports this representation by providing a.name as syntactic sugar for a["name"]. Tables may also carry methods: Because functions are first class values, table fields may contain functions. The form t:f(x) is syntactic sugar for t.f(t,x), which calls the method f from the table t passing the table itself as the first parameter.

Note that tables are objects, and not values. Variables do not contain tables, only references to them. Assignment, parameter passing, and returns always manipulate references to tables, and do not imply any kind of copy. Moreover, tables must be explicitly created before used.
Each of the types nil, number, and string has a different tag. All values of each of these types have the same pre-defined tag. Finally, values of type table can have variable tags, assigned by the programmer . The tag function returns the tag of a given value. User tags are created with the function newtag. The settag function is used to change the tag of a table . Tags are mainly used to select tag methods when some events occur. Tag methods are the main mechanism for extending the semantics of **Lua** .

1.11.3. The Language

This section describes the lexis, the syntax, and the semantics of **Lua**.

Lexical Conventions

Identifiers in **Lua** can be any string of letters, digits, and underscores, not beginning with a digit. This coincides with the definition of identifiers in most languages, except that the definition of letter depends on the current locale: Any character considered alphabetic by the current locale can be used in an identifier. The following words are reserved, and cannot be used as identifiers:

and	break	do	else	elseif
end	for	function	if	in
local	nil	not	or	repeat
return	then	until	while	

Lua is a case-sensitive language: and is a reserved word, but And and ánd (if the locale permits) are two different, valid identifiers. As a convention, identifiers starting with underscore followed by uppercase letters (such as _INPUT) are reserved for internal variables.

```
myVariable=2
myvariable=3
print(myVariable," not equal to ",myvariable)
```

The following strings denote other tokens:

~=	<=	>=	<	>	==	=	+	-	*	/	[
(	)	{	}	[	]	;	,	.	..	...	

Literal strings can be delimited by matching single or double quotes, and can contain the C-like escape sequences `\a' ( bell), `\b' (backspace),`\f' (form feed), `\n' (newline), `\r' (carriage return), `\t' (horizontal tab), `\v' (vertical tab), `\\' (backslash), `\"' (double quote), `\'' (single quote), and `\newline' (that is, a backslash followed by a real newline, which results in a newline in the string). A character in a string may also be specified by its numerical value, through the escape sequence `\ddd', where ddd is a sequence of up to three decimal digits. Strings in **Lua** may contain any 8-bit value, including embedded zeros, which can be specified as `\000'.
Literal strings can also be delimited by matching [[...]]. Literals in this bracketed form may run for several lines, may contain nested [[...]] pairs, and do not interpret escape sequences. This form is specially convenient for writing strings that contain program pieces or other quoted strings. As an example, in a system using ASCII, the following three literals are equivalent:

```
1)  "alo\n123\""
2)  '\97lo\10\04923"'
3)  [[alo
    123"]]
```

Comments start anywhere outside a string with a double hyphen (--) and run until the end of the line.

```
-- This is a comment
-- This program does nothing...
```

Numerical constants may be written with an optional decimal part and an optional decimal exponent. Examples of valid numerical constants are

```
3   3.0   3.1416  314.16e-2   0.31416E1
```

Coercion

Lua provides some automatic conversions between values at run time. Any arithmetic operation applied to a string tries to convert that string to a number, following the usual rules. Conversely, whenever a number is used when a string is expected, that number is converted to a string, in a reasonable format. The format is chosen so that a conversion from number to string then back to number reproduces the original number exactly. Thus, the conversion does not necessarily produces nice-looking text for some numbers. For complete control of how numbers are converted to strings, use the format function .

```
myString="100"
myString=myString+1
-- Will display value 101
print(myString)
```

Adjustment

Functions in **Lua** can return many values. Because there are no type declarations, when a function is called the system does not know how many values the function will return, or how many parameters it needs. Therefore, sometimes, a list of values must be adjusted, at run time, to a given length. If there are more values than are needed, then the excess values are thrown away. If there are less values than are needed, then the list is extended with as many nil's as needed. This adjustment occurs in multiple assignments and in function calls.

```
-- function myTest returns two values
function myTest()
return 1,2
end
local a
local b
a,b=myTest()
print("a= "..a.."", b= "..b)
```

Statements

Lua supports an almost conventional set of statements, similar to those in Pascal or C. The conventional commands include assignment, control structures, and procedure calls. Non-conventional commands include table constructors and local variable declarations.

Blocks

A block is a list of statements; syntactically, a block is equal to a chunk:

block ::= chunk

A block may be explicitly delimited:

stat ::= do block end

Explicit blocks are useful to control the scope of local variables. Explicit blocks are also sometimes used to add a return or break statement in the middle of another block.

Assignment

Lua allows multiple assignment. Therefore, the syntax for assignment defines a list of variables on the left side and a list of expressions on the right side. The elements in both lists are separated by commas:

stat ::= varlist1 `=' explist1
varlist1 ::= var {`,' var}

This statement first evaluates all values on the right side and eventual indices on the left side, and then makes the assignments. So, the code

x = 3
x, a[x] = 4, 20

sets a[3] to 20, but does not affect a[4] because the x in a[x] is evaluated before it is assigned 4.
Multiple assignment can be used to exchange two values, as in

x, y = y, x

The two lists in a multiple assignment may have different lengths. Before the assignment, the list of values is adjusted to the length of the list of variables.
A single name can denote a global variable, a local variable, or a formal parameter:

var ::= name

```
a,b,c=1,2,3
print("a= "..a.., b= "..b..",c= "..c)
```

Square brackets are used to index a table:

```
var ::= varorfunc `[' exp1 `]'
varorfunc ::= var | functioncall
```

The varorfunc should result in a table value, from where the field indexed by the expression exp1 value gets the assigned value.
The syntax var.NAME is just syntactic sugar for var["NAME"]:

```
var ::= varorfunc `.' name
```

The meaning of assignments and evaluations of global variables and indexed variables can be changed by tag methods. Actually, an assignment x = val, where x is a global variable, is equivalent to a call setglobal("x", val) and an assignment t[x] = val is equivalent to settable_event(t,x,val).

Control Structures

The control structures if, while, and repeat have the usual meaning and familiar syntax

```
stat ::= while exp1 do block end
```

```
local a
a=0
while a<20 do
   print(a)
   a=a+1
end
```

stat ::= repeat block until exp1

```
local a
a=0
repeat
   print(a)
   a=a+1
until a>=20
```

stat ::= if exp1 then block {elseif exp1 then block} [else block] end

The condition expression exp1 of a control structure may return any value. All values different from nil are considered true; only nil is considered false.

```
local a
a=0
while a<20 do
   if a<10 then
      print(a)
   end
   a=a+1
end
```

The return statement is used to return values from a function or from a chunk. Because functions or chunks may return more than one value, the syntax for the return statement is

stat ::= return [explist1]

The break statement can be used to terminate the execution of a loop, skipping to the next statement after the loop:

stat ::= break

A break ends the innermost enclosing loop (while, repeat, or for).
For syntactic reasons, return and break statements can only be written as the last statements of a block. If it is really necessary to return or break in the middle of a block, an explicit inner block can used, as in the idiom `do return end', because now return is last statement in the inner block.

For Statement

The for statement has two forms, one for numbers and one for tables. The numerical for loop has the following syntax:

stat ::= for name `=' exp1 `,' exp1 [`,' exp1] do block end

```
local a
for a=0,20 do
   print(a)
end
```

A for statement like

```
for var = e1 ,e2, e3 do block end
```

is equivalent to the code:

```
do
  local var, _limit, _step = tonumber(e1), tonumber(e2), tonumber(e3)
  if not (var and _limit and _step) then error() end
  while (_step>0 and var<=_limit) or (_step<=0 and var>=_limit) do
    block
    var = var+_step
  end
end
```

Note the following:
* sep=0pt
* _limit and _step are invisible variables. The names are here for explanatory purposes only.
* The behavior is undefined if you assign to var inside the block.
* If the third expression (the step) is absent, then a step of_1 is used.
* Both the limit and the step are evaluated only once, before the loop starts.
* The variable var is local to the statement; you cannot use its value after the for ends.
* You can use break to exit a for. If you need the value of the index, assign it to another variable before breaking.

The table for statement traverses all pairs (index,value) of a given table. It has the following syntax:

```
stat ::= for name `,' name in exp1 do block end
```

A for statement like

```
for index, value in exp do block end
```

is equivalent to the code:

```
do
  local _t = exp
  local index, value = next(t, nil)
  while index do
    block
    index, value = next(t, index)
  end
end
```

Note the following:
* sep=0pt
* _t is an invisible variable. The name is here for explanatory purposes only.
* The behavior is undefined if you assign to index inside the block.
* The behavior is undefined if you change the table _t during the traversal.
* The variables index and value are local to the statement; you cannot use their values after the for ends.
* You can use break to exit a for. If you need the value of index or value, assign them to other variables before breaking.
* The order that table elements are traversed is undefined, even for numerical indices. If you want to traverse indices in numerical order, use a numerical for.

Function Calls as Statements

Because of possible side-effects, function calls can be executed as statements:

```
stat ::= functioncall
```

In this case, all returned values are thrown away.

Local Declarations

Local variables may be declared anywhere inside a block. The declaration may include an initial assignment:

```
stat ::= local declist [init]
declist ::= name {`,' name}
init ::= `=' explist1
```

If present, an initial assignment has the same semantics of a multiple assignment. Otherwise, all variables are initialized with nil.
A chunk is also a block, and so local variables can be declared outside any explicit block.
The scope of local variables begins after the declaration and lasts until the end of the block. Thus, the code local print= print creates a local variable called print whose initial value is that of the global variable of the same name.

```
for index=0,20 do
-- Declare 'var' as local
   local var
   var=index*2
end
-- Will print 'nil' because var is local to the loop
-- and does not have value outside it
print(var)
```

Expressions

·Basic Expressions
The basic expressions in **Lua** are

```
exp ::= `(' exp `)'
exp ::= nil
exp ::= number
exp ::= literal
exp ::= var
exp ::= upvalue
exp ::= function
exp ::= functioncall
exp ::= tableconstructor
```

An access to a global variable x is equivalent to a call getglobal("x") and an access to an indexed variable t[x] is equivalent to a call gettable_event(t,x).
The non-terminal exp1 is used to indicate that the values returned by an expression must be adjusted to one single value:

```
exp1 ::= exp
```

·Arithmetic Operators
Lua supports the usual arithmetic operators: the binary + (addition), - (subtraction), * (multiplication), / (division), and ^ (exponentiation); and unary - (negation). If the operands are numbers, or strings that can be converted to numbers, then all operations except exponentiation have the usual meaning. Otherwise, an appropriate tag method is called. An exponentiation always calls a tag method. The standard mathematical library redefines this method for numbers, giving the expected meaning to exponentiation.

- Relational Operators
The relational operators in **Lua** are

```
==   ~=   <   >   <=   >=
```

These operators return nil as false and a value different from nil as true.
Equality (==) first compares the tags of its operands. If they are different, then the result is nil. Otherwise, their values are compared. Numbers and strings are compared in the usual way. Tables and functions are compared by reference, that is, two tables are considered equal only if they are the same table (* (page 99)). The operator ~= is exactly the negation of equality (==).
The conversion rules do not apply to equality comparisons. Thus, "0"==0 evaluates to false, and t[0] and t["0"] denote different entries in a table.
The order operators work as follows. If both arguments are numbers, then they are compared as such. Otherwise, if both arguments are strings, then their values are compared using lexicographical order. Otherwise, the ``lt'' tag method is called.

·Logical Operators
The logical operators in **Lua** are

```
and   or   not
```

Like the control structures, all logical operators consider nil as false and anything else as true.
The conjunction operator and returns nil if its first argument is nil; otherwise, it returns its second argument. The disjunction operator or returns its first argument if it is different from nil; otherwise, it returns its second argument. Both and and or use short-cut evaluation, that is, the second operand is evaluated only if necessary.
There are two useful **Lua** idioms that use logical operators. The first idiom is

```
x = x or v
```

which is equivalent to

```
if x == nil then x = v end
```

This idiom sets x to a default value v when x is not set.
The second idiom is

```
x = a and b or c
```

which should be read as x = (a and b) or c. This idiom is equivalent to

```
if a then x = b else x = c end
```

provided that b is not nil.

·Concatenation
The string concatenation operator in **Lua** is denoted by two dots (`..'). If both operands are strings or numbers, then they are converted to strings according to the rules. Otherwise, the ``concat'' tag method is called.

·Precedence
Operator precedence in **Lua** follows the table below, from the lower to the higher priority:

```
and   or
<    >    <=   >=   ~=   ==
..
+     -
*     /
not   - (unary)
^
```

All binary operators are left associative, except for ^ (exponentiation), which is right associative. The pre-compiler may rearrange the order of evaluation of associative operators (such as_.. or_+), as long as these optimizations do not change normal results. However, these optimizations may change some results if you define non-associative tag methods for these operators.

·Table Constructors
Table constructors are expressions that create tables; every time a constructor is evaluated, a new table is created. Constructors can be used to create empty tables, or to create a table and initialize some of its fields. The general syntax for constructors is

```
tableconstructor ::= `{' fieldlist `}'
fieldlist ::= lfieldlist | ffieldlist | lfieldlist `;' ffieldlist | ffieldlist `;' lfieldlist
lfieldlist ::= [lfieldlist1]
ffieldlist ::= [ffieldlist1]
```

The form lfieldlist1 is used to initialize lists:

```
lfieldlist1 ::= exp {`,' exp} [`,']
```

The expressions in the list are assigned to consecutive numerical indices, starting with_1. For example,

```
a = {"v1", "v2", 34}
```

is equivalent to

```
do
  local temp = {}
  temp[1] = "v1"
  temp[2] = "v2"
  temp[3] = 34
  a = temp
end
```

The form ffieldlist1 initializes other fields in a table:

```
ffieldlist1 ::= ffield {`,' ffield} [`,']
ffield ::= `[' exp `]' `=' exp | name `=' exp
```

For example,

```
a = {[f(k)] = g(y), x = 1, y = 3, [0] = b+c}
```

is equivalent to

```
do
  local temp = {}
  temp[f(k)] = g(y)
  temp.x = 1    -- or temp["x"] = 1
  temp.y = 3    -- or temp["y"] = 3
  temp[0] = b+c
  a = temp
end
```

An expression like {x = 1, y = 4} is in fact syntactic sugar for {["x"] = 1, ["y"] = 4}.
Both forms may have an optional trailing comma, and can be used in the same constructor separated by a semi-colon. For example, all forms below are correct.

```
x = {;}
x = {"a", "b",}
x = {type="list"; "a", "b"}
x = {f(0), f(1), f(2),; n=3,}
```

```
-- To use this script :
-- Run it, then examine the output on log window,
-- and compare it to the matching source code
-- to understand how tables can be used.

-- This function displays a full table content
function tableDump(table,indentStr)
local index,val
    -- Browses all table elements
    for index,val in table do
        -- recursive call if table element itself is a table
        if type(val)=="table" then
            print(indentStr.."["..index.."] :")
            tableDump(table[index],indentStr..    ")
        -- otherwise, displays on log window
        else print(indentStr.."["..index.."] :",val)
        end
    end
end

-- 1- Simple table (numerical indexes)
c={} -- c is an empty table
c[1]=1 -- fills in with values
c[2]=4
c[3]=9
c[4]=16
c[5]=c[3]+c[4]
print("Displaying table (1)")
tableDump(c,"")

-- 2- Simple table (word indexes)
c={} -- c is an empty table
c["One"]=1 -- fills in with values
c["Two"]=4
c.Three=9 -- Equivalent to c["Three"]=9
print("\rDisplaying table (2)")
tableDump(c,"")

-- 3- Table of table
c={} -- c is an empty table
c[1]={}  -- c[1] is a table too
c[1].double=2
c[1].triple=3
c[2]={}  -- c[2] is a table too
c[2].double=4
c[2].triple=6
print("\rDisplaying table (3)")
tableDump(c,"")

-- 4- Pre-inited table
c={"Hello","World",34}
print("\rDisplaying table (4)")
tableDump(c,"")

-- 5- Pre-inited table
c={First="Hello",Second="World",[3]=34}
print("\rDisplaying table (5)")
tableDump(c,"")

-- 6- Complex Pre-inited table
c={"One","Two",{cell=1,line=2},"Four"}
print("\rDisplaying table (6)")
tableDump(c,"")
```

·**Function Calls**
A function call in **Lua** has the following syntax:

functioncall ::= varorfunc args

First, varorfunc is evaluated. If its value has type function, then this function is called, with the given arguments. Otherwise, the ``function'' tag method is called, having as first parameter the value of varorfunc, and then the original call arguments.

The form

functioncall ::= varorfunc `:' name args

can be used to call ``methods''. A call v:name(...) is syntactic sugar for v.name(v, ...), except that v is evaluated only once.

Arguments have the following syntax:

args ::= `(' [explist1] `)'
args ::= tableconstructor
args ::= literal
explist1 ::= {exp1 `,'} exp

All argument expressions are evaluated before the call. A call of the form f{...} is syntactic sugar for f({...}), that is, the argument list is a single new table. A call of the form f'...' (or f"..." or f[[...]]) is syntactic sugar for f('...'), that is, the argument list is a single literal string.
Because a function can return any number of results, the number of results must be adjusted before they are used. If the function is called as a statement, then its return list is adjusted to_0, thus discarding all returned values. If the function is called in a place that needs a single value (syntactically denoted by the non-terminal exp1), then its return list is adjusted to_1, thus discarding all returned values but the first one. If the function is called in a place that can hold many values (syntactically denoted by the non-terminal exp), then no adjustment is made. The only places that can hold many values is the last (or the only) expression in an assignment, in an argument list, or in the return statement. Here are some examples:

```
f()                -- adjusted to 0 results
g(f(), x)          -- f() is adjusted to 1 result
g(x, f())          -- g gets x plus all values returned by f()
a,b,c = f(), x     -- f() is adjusted to 1 result (and c gets nil)
a,b,c = x, f()     -- f() is adjusted to 2
a,b,c = f()        -- f() is adjusted to 3
return f()         -- returns all values returned by f()
return x,y,f()     -- returns a, b, and all values returned by f()
```

·Function Definitions

The syntax for function definition is

function ::= function `(' [parlist1] `)' block end
stat ::= function funcname `(' [parlist1] `)' block end
funcname ::= name | name `.' name | name `:' name

The statement

function f () ... end

is just syntactic sugar for

f = function () ... end

and the statement

function v.f () ... end

is syntactic sugar for

v.f = function () ... end

A function definition is an executable expression, whose value has type function. When **Lua** pre-compiles a chunk, all its function bodies are pre-compiled too. Then, whenever **Lua** executes the function definition, its upvalues are fixed, and the function is instantiated (or closed). This function instance (or closure) is the final value of the expression. Different instances of the same function may have different upvalues.
Parameters act as local variables, initialized with the argument values:

parlist1 ::= `...'
parlist1 ::= name {`,' name} [`,' `...']

When a function is called, the list of arguments is adjusted to the length of the list of parameters, unless the function is a vararg function, which is indicated by three dots (`...') at the end of its parameter list. A vararg function does not adjust its argument list; instead, it collects all extra arguments into an implicit parameter, called arg. The value of arg is a table, with a field_n whose value is the number of extra arguments, and the extra arguments at positions 1,_2,_...,_n.
As an example, consider the following definitions:

```
function f(a, b) end
function g(a, b, ...) end
function r() return 1,2,3 end
```

Then, we have the following mapping from arguments to parameters:

CALL PARAMETERS

```
f(3)            a=3, b=nil
f(3, 4)         a=3, b=4
f(3, 4, 5)      a=3, b=4
f(r(), 10)      a=1, b=10
f(r())          a=1, b=2

g(3)            a=3, b=nil, arg={n=0}
g(3, 4)         a=3, b=4, arg={n=0}
g(3, 4, 5, 8)   a=3, b=4, arg={5, 8; n=2}
g(5, r())       a=5, b=1, arg={2, 3; n=2}
```

Results are returned using the return statement. If control reaches the end of a function without encountering a return statement, then the function returns with no results.
The syntax

```
funcname ::= name `:' name
```

is used for defining methods, that is, functions that have an implicit extra parameter self.
The statement

```
function v:f (...) ... end
```

is just syntactic sugar for

```
v.f = function (self, ...) ... end
```

Note that the function gets an extra formal parameter called self.

Visibility and Upvalues

A function body may refer to its own local variables (which include its parameters) and to global variables, as long as they are not shadowed by local variables with the same name from enclosing functions. A function cannot access a local variable from an enclosing function, since such variables may no longer exist when the function is called. However, a function may access the value of a local variable from an enclosing function, using upvalues, whose syntax is

```
upvalue ::= `%' name
```

An upvalue is somewhat similar to a variable expression, but whose value is frozen when the function wherein it appears is instantiated. The name used in an upvalue may be the name of any variable visible at the point where the function is defined, that is, global variables and local variables from the immediately enclosing function. Note that when the upvalue is a table, only the reference to that table (which is the value of the upvalue) is frozen; the table contents can be changed at will. Using table values as upvalues is a technique for having writable but private state attached to functions. Here are some examples:

```
a,b,c = 1,2,3   -- global variables
local d
function f (x)
  local b = {}  -- x and b are local to f; b shadows the global b
  local g = function (a)
    local y     -- a and y are local to g
    p = a       -- OK, access local `a'
    p = c       -- OK, access global `c'
    p = b       -- ERROR: cannot access a variable in outer scope
    p = %b      -- OK, access frozen value of `b' (local to `f')
    %b = 3      -- ERROR: cannot change an upvalue
    %b.x = 3    -- OK, change the table contents
    p = %c      -- OK, access frozen value of global `c'
    p = %y      -- ERROR: `y' is not visible where `g' is defined
    p = %d      -- ERROR: `d' is not visible where `g' is defined
  end           -- g
end             -- f
```

1.11.4. Standard Libraries

The standard libraries provide useful functions that are implemented directly through the standard API.

- basic library;
- string manipulation;
- mathematical functions (sin, log, etc);

Basic Functions

The basic library provides some core functions to **Lua**. Therefore, if you do not include this library in your application, you should check carefully whether you need to provide some alternative implementation for some facilities. (For instance, without function _ERRORMESSAGE, **Lua** is unable to show error messages.)

_**ALERT** (message)
Prints its only string argument to stderr. All error messages in **Lua** are printed through the function stored in the _ALERT global variable. Therefore, a program may assign another function to this variable to change the way such messages are shown (for instance, for systems without stderr).

```
-- New __ALERT function, to print script error to an alert box
function _ALERT(message)
message=gsub(message,"\n","\r") -- needs \r to end lines
Message("Script internal error : ",message)
end
-- error example
a=10/0
```

assert (v [, message])
Issues an ``assertion failed!'' error when its argument v is nil. This function is equivalent to the following **Lua** function:

```
function assert (v, m)
  if not v then
    m = m or ""
    error("assertion failed!  " .. m)
  end
end
```

call (func, arg [, mode [, errhandler]])
Calls function func with the arguments given by the table arg. The call is equivalent to

```
func(arg[1], arg[2], ..., arg[n])
```

where n is the result of getn(arg). All results from func are simply returned by call.
By default, if an error occurs during the call to func, the error is propagated. If the string mode contains "x", then the call is protected. In this mode, function call does not propagate an error, regardless of what happens during the call. Instead, it returns nil to signal the error (besides calling the appropriated error handler).
If errhandler is provided, the error function _ERRORMESSAGE is temporarily set to errhandler, while func runs. In particular, if errhandler is nil, no error messages will be issued during the execution of the called function.

collectgarbage ([limit])
Sets the garbage-collection threshold for the given limit (in Kbytes), and checks it against the byte counter. If the new threshold is smaller than the byte counter, then **Lua** immediately runs the garbage collector. If limit is absent, it defaults to zero (thus forcing a garbage-collection cycle).

copytagmethods (tagto, tagfrom)
Copies all tag methods from one tag to another; returns tagto.

dostring (string [, chunkname])
Executes a given string as a **Lua** chunk. If there is any error executing the string, then dostring returns nil. Otherwise, it returns the values returned by the chunk, or a non-nil value if the chunk returns no values. The optional parameter chunkname is the ``name of the chunk'', used in error messages and debug information.

error (message)
Calls the error handler and then terminates the last protected function called (in_C: lua_dofile, lua_dostring, lua_dobuffer, or lua_callfunction; in **Lua**: dostring, or call in protected mode). If message is nil, then the error handler is not called. Function error never returns.

foreach (table, func)
Executes the given func over all elements of table. For each element, the function is called with the index and respective value as arguments. If the function returns any non-nil value, then the loop is broken, and this value is returned as the final value of foreach. This function could be defined in **Lua**:

```
function foreach (t, f)
  for i, v in t do
    local res = f(i, v)
    if res then return res end
  end
end
```

The behavior of foreach is undefined if you change the table t during the traversal.

foreachi (table, func)
Executes the given func over the numerical indices of table. For each index, the function is called with the index and respective value as arguments. Indices are visited in sequential order, from_1 to n, where n is the result of getn(table). If the function returns any non-nil value, then the loop is broken, and this value is returned as the final value of foreachi. This function could be defined in **Lua**:

```
function foreachi (t, f)
  for x=1,getn(t) do
    local res = f(x, t[x])
```

```
    if res then return res end
  end
end
```

getglobal (name)
Gets the value of a global variable, or calls a tag method for ``getglobal''. The string name does not need to be a syntactically valid variable name.

getn (table)
Returns the ``size'' of a table, when seen as a list. If the table has an n field with a numeric value, this value is the ``size'' of the table. Otherwise, the ``size'' is the largest numerical index with a non-nil value in the table. This function could be defined in **Lua**:

```
function getn (t)
  if type(t.n) == "number" then return t.n end
  local max = 0
  for i, _ in t do
    if type(i) == "number" and i>max then max=i end
  end
  return max
end
```

gettagmethod (tag, event)
Returns the current tag method for a given pair (tag, event). This function cannot be used to get a tag method for the `` gc'' event. (Such tag methods can only be manipulated by C_code.)

globals ([table])
Returns the current table of globals. If the argument table is given, then it also sets this table as the table of globals.

newtag ()
Returns a new tag.

next (table, [index])
Allows a program to traverse all fields of a table. Its first argument is a table and its second argument is an index in this table. next returns the next index of the table and the value associated with the index. When called with nil as its second argument, next returns the first index of the table and its associated value. When called with the last index, or with nil in an empty table, next returns nil. If the second argument is absent, then it is interpreted as nil.
Lua has no declaration of fields; semantically, there is no difference between a field not present in a table or a field with value nil. Therefore, next only considers fields with non-nil values. The order in which the indices are enumerated is not specified, even for numeric indices (to traverse a table in numeric order, use a numerical for or the function foreachi). The behavior of next is undefined if you change the table during the traversal.

print (e1, e2, ...)
Receives any number of arguments, and prints their values using the strings returned by tostring. This function is not intended for formatted output, but only as a quick way to show a value, for instance for debugging.

rawget (table, index)
Gets the real value of table[index], without invoking any tag method. table must be a table, and index is any value different from nil.

rawset (table, index, value)
Sets the real value of table[index] to value, without invoking any tag method. table must be a table, index is any value different from nil, and value is any **Lua** value.

setglobal (name, value)
Sets the named global variable to the given value, or calls a tag method for ``setglobal''. The string name does not need to be a syntactically valid variable name.

settag (t, tag)
Sets the tag of a given table . tag must be a value created with newtag . settag returns the value of its first argument (the table).

settagmethod (tag, event, newmethod)
Sets a new tag method to the given pair (tag, event) and returns the old method. If newmethod is nil, then settagmethod restores the default behavior for the given event. This function cannot be used to set a tag method for the ``gc'' event. (Such tag methods can only be manipulated by C_code.)

sort (table [, comp])
Sorts table elements in a given order, in-place, from table[1] to table[n], where n is the result of getn(table) . If comp is given, then it must be a function that receives two table elements, and returns true (that is, a value different from nil) when the first is less than the second (so that not comp(a[x+1], a[x]) will be true after the sort). If comp is not given, then the standard **Lua** operator < is used instead.

The sort algorithm is not stable (that is, elements considered equal by the given order may have their relative positions changed by the sort).

tag (v)
Allows **Lua** programs to test the tag of a value. It receives one argument, and returns its tag (a number).

tonumber (e [, base])
Tries to convert its argument to a number. If the argument is already a number or a string convertible to a number, then tonumber returns that number; otherwise, it returns nil.
An optional argument specifies the base to interpret the numeral. The base may be any integer between 2 and 36, inclusive. In bases above_10, the letter `A' (either upper or lower case) represents_10, `B' represents_11, and so forth, with `Z' representing 35. In base 10 (the default), the number may have a decimal part, as well as an optional exponent part. In other bases, only unsigned integers are accepted.

tostring (e)
Receives an argument of any type and converts it to a string in a reasonable format. For complete control of how numbers are converted, use function format.

tinsert (table [, pos] , value)
Inserts element value at table position pos, shifting other elements to open space, if necessary. The default value for pos is n+1, where n is the result of getn(table), so that a call tinsert(t,x) inserts x at the end of table t. This function also sets or increments the field n of the table to n+1. This function is equivalent to the following **Lua** function, except that the table accesses are all raw (that is, without tag methods):

```
function tinsert (t, ...)
  local pos, value
  local n = getn(t)
  if arg.n == 1 then
    pos, value = n+1, arg[1]
  else
    pos, value = arg[1], arg[2]
  end
  t.n = n+1;
  for i=n,pos,-1 do
    t[x+1] = t[x]
  end
  t[pos] = value
end
```

tremove (table [, pos])
Removes from table the element at position pos, shifting other elements to close the space, if necessary. Returns the value of the removed element. The default value for pos is n, where n is the result of getn(table), so that a call tremove(t) removes the last element of table t. This function also sets or decrements the field n of the table to n-1.
This function is equivalent to the following **Lua** function, except that the table accesses are all raw (that is, without tag methods):

```
function tremove (t, pos)
  local n = getn(t)
  if n<=0 then return end
  pos = pos or n
  local value = t[pos]
  for x=pos,n-1 do
    t[x] = t[x+1]
  end
  t[n] = nil
  t.n = n-1
  return value
end
```

type (v)
Allows **Lua** programs to test the type of a value. It receives one argument, and returns its type, coded as a string. The possible results of this function are "nil" (a string, not the value nil), "number", "string", "table" and "function" (* (page 99)

String Manipulation

This library provides generic functions for string manipulation, such as finding and extracting substrings and pattern matching. When indexing a string in **Lua**, the first character is at position_1 (not at_0, as in C). Also, indices are allowed to be negative and are intepreted as indexing backwards, from the end of the string. Thus, the last character is at position -1, and so on.

strbyte (s [, i])
Returns the internal numerical code of the i-th character of s. If i is absent, then it is assumed to be_1. i may be negative.

Numerical codes are not necessarily portable across platforms.

strchar (i1, i2, ...)
Receives 0 or more integers. Returns a string with length equal to the number of arguments, wherein each character has the internal numerical code equal to its correspondent argument.
Numerical codes are not necessarily portable across platforms.

strfind (s, pattern [, init [, plain]])
Looks for the first match of pattern in s. If it finds one, then strfind returns the indices of s where this occurrence starts and ends; otherwise, it returns nil. If the pattern specifies captures (see gsub below), the captured strings are returned as extra results.
A third, optional numerical argument init specifies where to start the search; its default value is 1, and may be negative. A value of 1 as a fourth, optional argument plain turns off the pattern matching facilities, so the function does a plain `` find substring'' operation, with no characters in pattern being considered ``magic''. Note that if plain is given, then init must be given too.

strifind (s, pattern [, init [, plain]])
Same as strfind but case insensitive.

strlen (s)
Receives a string and returns its length. The empty string "" has length 0. Embedded zeros are counted, and so "a\000b\ 000c" has length 5.

strlower (s)
Receives a string and returns a copy of that string with all upper case letters changed to lower case. All other characters are left unchanged. The definition of what is an upper-case letter depends on the current locale.

strrep (s, n)
Returns a string that is the concatenation of n copies of the string s.
ll i
strsub (s, i [, j])
Returns another string, which is a substring of s, starting at i and running until j; i and j may be negative, If j is absent, then it is assumed to be equal to -1 (which is the same as the string length). In particular, the call strsub(s,1,j) returns a prefix of s with length j, and the call strsub(s, -i) returns a suffix of s with length i.

strupper (s)
Receives a string and returns a copy of that string with all lower case letters changed to upper case. All other characters are left unchanged. The definition of what is a lower case letter depends on the current locale.

format (formatstring, e1, e2, ...)
Returns a formatted version of its variable number of arguments following the description given in its first argument (which must be a string). The format string follows the same rules as the printf family of standard C_functions. The only differences are that the options/modifiers *, l, L, n, p, and h are not supported, and there is an extra option, q. The q option formats a string in a form suitable to be safely read back by the **Lua** interpreter: The string is written between double quotes, and all double quotes, returns, and backslashes in the string are correctly escaped when written. For instance, the call

```
format('%q', 'a string with "quotes" and \n new line')
```

will produce the string:

"a string with \"quotes\" and new line"

Conversions can be applied to the n-th argument in the argument list, rather than the next unused argument. In this case the conversion character % is replaced by the sequence %d$, where d is a decimal digit in the range [1,9], giving the position of the argument in the argument list. For instance, the call format("%2$d -> %1$03d", 1, 34) will result in "34 -> 001". The same argument can be used in more than one conversion.
The options c, d, E, e, f, g, G, i, o, u, X, and x all expect a number as argument, whereas q and s expect a string. The * modifier can be simulated by building the appropriate format string. For example, "%*g" can be simulated with "%"..width. ."g".
Neither the format string nor the string values to be formatted with %s can contain embedded zeros. %q handles string values with embedded zeros.

gsub (s, pat, repl [, n])
Returns a copy of s in which all occurrences of the pattern pat have been replaced by a replacement string specified by repl. gsub also returns, as a second value, the total number of substitutions made.
If repl is a string, then its value is used for replacement. Any sequence in repl of the form %n with n between 1 and 9 stands for the value of the n-th captured substring.
If repl is a function, then this function is called every time a match occurs, with all captured substrings passed as arguments, in order (see below). If the value returned by this function is a string, then it is used as the replacement string otherwise, the replacement string is the empty string.
The last, optional parameter n limits the maximum number of substitutions to occur. For instance, when n is 1 only the first occurrence of pat is replaced.
Here are some examples:

```
x = gsub("hello world", "(%w+)", "%1 %1")
--> x="hello hello world world"

x = gsub("hello world", "(%w+)", "%1 %1", 1)
--> x="hello hello world"

x = gsub("hello world from Lua", "(%w+)%s*(%w+)", "%2 %1")
--> x="world hello Lua from"

x = gsub("home = $HOME, user = $USER", "%$(%w+)", getenv)
--> x="home = /home/roberto, user = roberto"  (for instance)

x = gsub("4+5 = $return 4+5$", "%$(.-)%$", dostring)
--> x="4+5 = 9"

local t = {name="lua", version="4.0"}
x = gsub("$name - $version", "%$(%w+)", function (v) return %t[v] end)
--> x="lua - 4.0"

t = {n=0}
gsub("first second word", "(%w+)", function (w) tinsert(%t, w) end)
--> t={"first", "second", "word"; n=3}
```

Patterns

Character Class:
a character class is used to represent a set of characters. The following combinations are allowed in describing a character class:
x
(where x is any magic characters ^$()%.[]*+-?) - represents the character x itself.
.
- (a dot) represents all characters.
%a
- represents all letters.
%c
- represents all control characters.
%d
- represents all digits.
%l
- represents all lower case letters.
%p
- represents all punctuation characters.
%s
- represents all space characters.
%u
- represents all upper case letters.
%w
- represents all alphanumeric characters.
%x
- represents all hexadecimal digits.
%z
- represents the character with representation 0.
%x
(where x is any non-alphanumeric character) - represents the character x. This is the standard way to escape the magic characters. We recommend that any punctuation character (even the non magic) should be preceded by a % when used to represent itself in a pattern.
[char-set]
- represents the class which is the union of all characters in char-set. A range of characters may be specified by

separating the end characters of the range with a -. All classes %x described above may also be used as components in a char-set. All other characters in char-set represent themselves. For example, [%w_] (or [_%w]) represents all alphanumeric characters plus the underscore, [0-7] represents the octal digits, and [0-7%l%-] represents the octal digits plus the lower case letters plus the - character.
The interaction between ranges and classes is not defined. Therefore, patterns like [%a-z] or [a-%%] have no meaning.
[^char-set]
- represents the complement of char-set, where char-set is interpreted as above.
For all classes represented by single letters (%a, %c, ...), the corresponding upper-case letter represents the complement of the class. For instance, %S represents all non-space characters.
The definitions of letter, space, etc. depend on the current locale. In particular, the class [a-z] may not be equivalent to %l. The second form should be preferred for portability.

Pattern Item:
a pattern item may be
* a single character class, which matches any single character in the class;
* a single character class followed by *, which matches 0 or more repetitions of characters in the class. These repetition items will always match the longest possible sequence;
* a single character class followed by +, which matches 1 or more repetitions of characters in the class. These repetition items will always match the longest possible sequence;
* a single character class followed by -, which also matches 0 or more repetitions of characters in the class. Unlike *, these repetition items will always match the shortest possible sequence;
* a single character class followed by ?, which matches 0 or 1 occurrence of a character in the class;
* %n, for n between 1 and 9; such item matches a sub-string equal to the n-th captured string (see below);
* %bxy, where x and y are two distinct characters; such item matches strings that start with_x, end with_y, and where the x and y are balanced. This means that, if one reads the string from left to right, counting +1 for an x and -1 for a y, the ending y is the first y where the count reaches 0. For instance, the item %b() matches expressions with balanced parentheses.

Pattern:
a pattern is a sequence of pattern items. A ^ at the beginning of a pattern anchors the match at the beginning of the subject string. A $ at the end of a pattern anchors the match at the end of the subject string. At other positions, ^ and $ have no special meaning and represent themselves.

Captures:
A pattern may contain sub-patterns enclosed in parentheses, they describe captures. When a match succeeds, the sub-strings of the subject string that match captures are stored (captured) for future use. Captures are numbered according to their left parentheses. For instance, in the pattern "(a*(.)%w(%s*))", the part of the string matching "a*(.)%w(%s*)" is stored as the first capture (and therefore has number_1); the character matching . is captured with number_2, and the part matching %s* has number_3.
A pattern cannot contain embedded zeros. Use %z instead.

Mathematical Functions

This library is an interface to some functions of the standard C_math library. In addition, it registers a tag method for the binary operator ^ that returns x^y when applied to numbers x^y.
The library provides the following functions:

```
abs  acos  asin  atan  atan2  ceil  cos  deg   exp   floor  log  log10
max  min   mod   rad   sin    sqrt  tan  frexp  ldexp  random  randomseed
```

plus a global variable PI. Most of them are only interfaces to the homonymous functions in the C_library, except that, for the trigonometric functions,all angles are expressed in degrees, not radians. The functions deg and rad can be used to convert between radians and degrees.
The function max returns the maximum value of its numeric arguments. Similarly, min computes the minimum. Both can be used with 1, 2, or more arguments.
The functions random and randomseed are interfaces to the simple random generator functions rand and srand, provided by ANSI C. (No guarantees can be given for their statistical properties.) The function random, when called without arguments, returns a pseudo-random real number in the range [0,1). When called with a number n, random returns a pseudo-random integer in the range [1,n]. When called with two arguments, l and u, random returns a pseudo-random integer in the range [l,u].

1.11.5. The Complete Syntax of Lua

```
chunk ::= {stat [`;']}

block ::= chunk

stat ::=  varlist1 `=' explist1
| functioncall
| do block end
| while exp1 do block end
| repeat block until exp1
| if exp1 then block {elseif exp1 then block} [else block] end
| return [explist1]
| break
| for `name' `=' exp1 `,' exp1 [`,' exp1] do block end
| for `name' `,' `name' in exp1 do block end
| function funcname `(' [parlist1] `)' block end
| local declist [init]

funcname ::=  `name' | `name' `.' `name' | `name' `:' `name'

varlist1 ::= var {`,' var}

var ::=  `name' | varorfunc `[' exp1 `]' | varorfunc `.' `name'

varorfunc ::= var | functioncall

declist ::= `name' {`,' `name'}

init ::= `=' explist1

explist1 ::= {exp1 `,'} exp

exp1 ::= exp

exp ::=  nil | `number' | `literal' | var | function | upvalue
| functioncall | tableconstructor | `(' exp `)' | exp binop exp | unop exp


functioncall ::=  varorfunc args | varorfunc `:' `name' args

args ::=  `(' [explist1] `)' | tableconstructor | `literal'

function ::= function `(' [parlist1] `)' block end

parlist1 ::=  `...' | `name' {`,' `name'} [`,' `...']

upvalue ::= `%' `name'

tableconstructor ::= `{' fieldlist `}'
fieldlist ::=  lfieldlist | ffieldlist | lfieldlist `;' ffieldlist | ffieldlist `;' lfieldlist
lfieldlist ::= [lfieldlist1]
ffieldlist ::= [ffieldlist1]
lfieldlist1 ::= exp {`,' exp} [`,']
ffieldlist1 ::= ffield {`,' ffield} [`,']
ffield ::=  `[' exp `]' `=' exp | `name' `=' exp

binop ::= `+' | `-' | `*' | `/' | `\^{ ' | `..'
| `<' | `<=' | `>' | `>=' | `==' | `\_{ '=}
| and | or}

unop ::= `-' | not
```

1.11.6. Notes from the Myriad Team

Here are remarks, warnings and tips about Lua, we found while writing our first scripts in **Lua** language. Please note however, these remarks are directly related to our own programming experience. Maybe we found problems that you won't consider as such...

Division by zero

This is a big problem in **Lua** : on some operating systems, dividing by zero exits the application.
The Lua library has been modified to provide a Lua standard error instead of a crash when a divide by zero occurs.

```
-- This program will write an error message
-- to log window instead of crashing.
print(10/0)
```

Checking false and true or nil

In many programming languages, nil (or NULL) represents a value of 0. So does the Boolean 'false' value.
In such languages, you can write either

if (myVariable==true) then ...
or
if (myVariable) then ...

These two syntaxes having the same effect.

It is NOT the case in Lua. Nil is a type, not a value.
It means :

if (myVariable==true) then ...
will pass if myVariable is defined (non-nil) AND its value is true, as expected, while :
if (myVariable) then ...
will only check the existence of myVariable, so it will pass if this variable exits, whatever its value is (true or false).
So, in Lua, this last syntax is in fact equivalent to :
if myVariable~=nil then ...

```
-- To test, change the following line to either :
-- variable = true
-- variable = false
-- variable = nil
-- or no line at all
variable=false

if variable then print("this variable exists")
else print("this variable does not exist (nil)")
end
if variable==true then print("this variable is true") end
if variable==false then print("this variable is false") end
```

Non-integer table (array) index

In many programming languages, array index are automatically integer values. In Lua, an array (table) index can be either an integer, a floating point number or a string.
It means:

myTable[10] is the same element as myTable[20/2], but not the same as myTable[10.5], myTable[21/2] or myTable["10"].

As a side note, here is an example showing how to build and use a bidimensional array :

```
T={}
for line=1,10 do
   T[line]={}
   for col=1,10 do
      T[line][col]=line*col
   end
end
print("6x9="..T[6][9])
```

1.11.7. Deviation from the Lua standard syntax

In order to implement the object types that enable a script to manage program objects, and to simplify the task of the script programmer, some changes have been made to the usual Lua syntax.
These changes only apply to the predefined object types.
If you create your own kind of objects in your script, they will follow the standard syntax as explained in the Lua documentation.

Object types

All new object types have been implemented as Lua tables. However, they are not **really** lua tables.
When setting a field by "object.field=value", you set in fact the matching field in the internal object of Harmony Assistant.
When reading a field by "value=object.field", you get information about the matching field in the internal object of Harmony Assistant.
It means, in some special cases, some fields can be read but not written (read-only information about an object), or several fields can be different ways to get or set the same internal value,so setting one of them will change the value of all.

Calling methods

When calling a method for such objects, the Lua syntax "object:method(parameters)" must not to be used anymore.
For compatibility with most of the currently used object-oriented languages, a method has to be called through "object. method(parameters)" instead.

Type values

The possible values for object types have been extended. The standard Lua "type" function can now return the object type name as defined in this manual, such as "Score","Staff","Symbol"...

Object content

Objects of these new types are references to the actual object. It means, if you transfer such a variable into another (for example, score2=score1), you won't make a copy of a score, but make score2 refer internally to the same score as score1.
If, after this, you perform any change to score1, it will change the value into score2 as well (and vice-versa)

Object comparison

Such new objects can be compared easily, without the restrictions that usually apply to standard Lua tables. If two objects refer to the same internal object, they are equal. If not, they are different.

1.12. MSDefine file

1.12.1. Overview

The MSDefine file provides constant definitions to be used in script files. This file is provided by default in the application.

1.12.2. Using it

Add the following line: **Include "MSDefine.mys"** at the beginning of your script file.
You can omit the extension and write **Include "MSDefine"**.
After this line, all constants defined in the MSDefine.mys file are available for your script.
The MSDefine file is searched first at the same level as the script, if not found, in the user "Includes" folder, if not found in the application "Includes" folder.

1.12.3. Modifying it

You are not allowed to apply changes to this file.
If you want to add your own constant definitions, we recommend to create a new file and to save it in the user's "Scripts\Includes" folder.
You will be able to use this file the same way as MSDefine.
For example, you create a file "MyDefinitions.mys" and you save it in the "Scripts\Includes" folder. Now, you can use Include "MyDefinitions" in your scripts.

Warning ! If you send your script file to somebody else, do not forget to provide the definition file too...

1.12.4. Content

Here is the content of this file.

```
            ------------------------------------------------
--NAME: MSDefine.mys
--CREATOR: Didier Guillion - Myriad Software
--VERSION: 1.3.7
--DATE: December 12 2002- April 2018
--ABSTRACT: Standard constants for MyrScript. For script developers only.
--INFO: This file includes all standard constant definitions to be used
--INFO: Just write : Include "MSDefine" at the begining of your script file.
------------------------------------------------

------------------- MISCELLANEOUS ----------------
--- Text face (bits)
FACE_NONE=0
FACE_BOLD=1
FACE_ITALIC=2
FACE_UNDERLINE=4
FACE_OUTLINE=8
FACE_SHADOW=16
FACE_CONDENSE=32
FACE_EXTEND=64
--- Line cap and join for curve drawing
CAP_ROUND=0
CAP_SQUARE=1
CAP_BUTT=2
JOIN_MITER=0
JOIN_BEVEL=1
JOIN_ROUND=2
--- Some useful color indexes
IDX_COLOR_CURRENT=0
IDX_COLOR_BLACK=1
IDX_COLOR_LIGHT_YELLOW=2
IDX_COLOR_YELLOW=3
IDX_COLOR_LIGHT_ORANGE=11
IDX_COLOR_ORANGE=17
IDX_COLOR_PINK=13
IDX_COLOR_RED=35
IDX_COLOR_LIGHT_GREEN=110
IDX_COLOR_GREEN=225
IDX_COLOR_DARK_GREEN=229
IDX_COLOR_LIGHT_BLUE=36
IDX_COLOR_BLUE=235
IDX_COLOR_DARK_BLUE=239
IDX_COLOR_LIGHT_GRAY=247
IDX_COLOR_MEDIUM_GRAY=249
IDX_COLOR_DARK_GRAY=251
IDX_COLOR_MAGENTA=96
IDX_COLOR_MEDIUM_BLUE=193
IDX_COLOR_BROWN=95
IDX_COLOR_WHITE=255

--- Color indexes that can be used in rules
IDX_COLOR_RULE_NONE=1
IDX_COLOR_RULE_DARK_BROWN=131
IDX_COLOR_RULE_DARK_RED=220
IDX_COLOR_RULE_DARK_GREEN=203
IDX_COLOR_RULE_DARK_BLUE=237
IDX_COLOR_RULE_DARK_PURPLE=104
IDX_COLOR_RULE_BROWN=251
IDX_COLOR_RULE_YELLOW=47
IDX_COLOR_RULE_RED=35
IDX_COLOR_RULE_GREEN=197
IDX_COLOR_RULE_BLUE=204
IDX_COLOR_RULE_PURPLE=30
IDX_COLOR_RULE_GRAY=250
IDX_COLOR_RULE_LIGHT_YELLOW=5
IDX_COLOR_RULE_LIGHT_RED=23
IDX_COLOR_RULE_LIGHT_GREEN=226
IDX_COLOR_RULE_LIGHT_BLUE=126
IDX_COLOR_RULE_LIGHT_PURPLE=18
IDX_COLOR_RULE_LIGHT_GRAY=248

--- Maximum number of bars
BAR_MAX=9999
--- Text alignment
ALIGN_RIGHT=-1
```

ALIGN_LEFT=0
ALIGN_CENTER=1
---- Files
ASK_FOR_NAME=1
AUCUN_FORMAT=-1
FORMAT_STANDARD=1
FORMAT_EXPORT=2
FORMAT_MU3=3
FORMAT_MIDI0=4
FORMAT_MIDI1=5
FORMAT_KAR=6
FORMAT_RMTF=7
FORMAT_CMF=8
FORMAT_ABC=9
FORMAT_TAB=10
FORMAT_MSF=11 --Load only
FORMAT_BAK=12 --Load only
FORMAT_MUT=13
FORMAT_AIFF=14
FORMAT_WAV=15
FORMAT_BAND_IN_A_BOX=16 --Load only
FORMAT_SND=17
FORMAT_RAW=18
FORMAT_MOD=19 --Load only
FORMAT_S3M=20 --Load only
FORMAT_MP3=21
FORMAT_TEMPLATE=22
FORMAT_ANY=23
FORMAT_OGG=24
FORMAT_STAFF_TEMPLATE=25
FORMAT_XML=26
FORMAT_XML_COMPRESSED=27
FORMAT_MXL=28
FORMAT_MYRWEB=29
FORMAT_HTML=30
FORMAT_PDF=1000

--- Graphical Export formats
EXPORT_EPS=1
EXPORT_FPCT=2
EXPORT_PICT=3
EXPORT_BMP=4
EXPORT_WMF=5
EXPORT_EMF=6
EXPORT_PNG=7
EXPORT_JPG=8
EXPORT_GIF=9
EXPORT_SVG=10
EXPORT_SVG_MULTI_PAGES=11
EXPORT_PDF=12
---------- APPLICATION ----------
--- Constant returned by Application.Language
LANGUAGE_FRENCH=0
LANGUAGE_ENGLISH=1
LANGUAGE_ITALIAN=2
LANGUAGE_GERMAN=3
LANGUAGE_SPANISH=4
LANGUAGE_JAPANESE=5
LANGUAGE_PORTUGUESE=6
LANGUAGE_DUTCH=7
---------- SCORE ----------
------ BarAction : BARLINES AND BREAK SYMBOLS ----
CUSTOM_BREAK_SYMBOL=-1 -- Cannot be used to create a new BarAction.

PIECE_START_BARLINE=1
PIECE_END_BARLINE=2
REPEAT_START_BARLINE=3
REPEAT_END_BARLINE=4
INVISIBLE_BARLINE=5
DOUBLE_BARLINE=6
BOLD_BARLINE=7
DOTTED_BARLINE=8
PART_START_BREAK_SYMBOL=9
PART_END_BREAK_SYMBOL=10
SEGNO_BREAK_SYMBOL=11
CODA_BREAK_SYMBOL=12
FINE_BREAK_SYMBOL=13
AL_CODA_BREAK_SYMBOL=14
DA_CAPO_BREAK_SYMBOL=15
DA_SEGNO_BREAK_SYMBOL=16
DA_CAPO_AL_FINE_BREAK_SYMBOL=17

```
DA_SEGNO_AL_FINE_BREAK_SYMBOL=18
DA_CAPO_AL_CODA_BREAK_SYMBOL=19
DA_SEGNO_AL_CODA_BREAK_SYMBOL=20
SIMPLE_BARLINE=21

-- BarAction : Position---
BARLINE_POSITION_BAR_START=0
BARLINE_POSITION_AFTER_KEY=1
BARLINE_POSITION_AFTER_MARGIN=2
BARLINE_POSITION_BAR_END=3

-- BarAction : Triggering --
TRIGGER_ALWAYS=0
TRIGGER_X_TIMES=1
TRIGGER_EACH_X_TIMES=2
TRIGGER_SELECTED_TIMES=3
TRIGGER_LAST_TIME=4

-- BarAction : Activation --
ACTIVATE_NOTHING=0
ACTIVATE_CODA=1
ACTIVATE_FINE=2

-- BarAction : Action --
BREAK_SYMBOL_NO_ACTION=0
BREAK_SYMBOL_STOP=1
BREAK_SYMBOL_JUMP_TO_BAR=2
BREAK_SYMBOL_JUMP_TO_WRITTEN_START=3
BREAK_SYMBOL_JUMP_TO_PLAYED_START=4
BREAK_SYMBOL_JUMP_TO_SEGNO=5
BREAK_SYMBOL_JUMP_TO_CODA=6
BREAK_SYMBOL_DEFINE_SEGNO=7
BREAK_SYMBOL_DEFINE_CODA=8
BREAK_SYMBOL_DEFINE_START=9
BREAK_SYMBOL_DEFINE_END=10
BREAK_SYMBOL_DEFINE_PART_START=11
BREAK_SYMBOL_DEFINE_PART_END=12
BREAK_SYMBOL_DEFINE_REPEAT_START=13
BREAK_SYMBOL_DEFINE_REPEAT_END=14

------------------- INSTRUMENTS -----------------
--- Output devices ---
NO_OUTPUT_DEVICE=-1
MIDI1_OUTPUT_DEVICE=9
MIDI2_OUTPUT_DEVICE=10
MIDI3_OUTPUT_DEVICE=11
MIDI4_OUTPUT_DEVICE=12
MIDI5_OUTPUT_DEVICE=13
MIDI6_OUTPUT_DEVICE=14
MIDI7_OUTPUT_DEVICE=15
MIDI8_OUTPUT_DEVICE=16
MIDI9_OUTPUT_DEVICE=17
MIDI10_OUTPUT_DEVICE=18
MIDI11_OUTPUT_DEVICE=19
MIDI12_OUTPUT_DEVICE=20
DIGITAL_OUTPUT_DEVICE=6
MYRSYNTH_OUTPUT_DEVICE=2
EXTERN_OUTPUT_DEVICE=3
QUICKTIME_OUTPUT_DEVICE=7 -- Mac Only
-- Layering --
NO_LAYERING=0
VELOCITY_LAYERING=1

---------- FX PROCESSORS ------------------
--- Resonator
FX_INACTIVE_RESONATOR=0
FX_STOPPED_RESONATOR=1
FX_FIXED_RESONATOR=2
FX_LFO_RESONATOR=3
FX_WAH_RESONATOR=4
--- Distortion
FX_INACTIVE_DISTORTION=0
FX_STOPPED_DISTORTION=1
FX_REGULAR_DISTORTION=2
FX_OVERDRIVEN_DISTORTION=3
--- Flanger/Chorus
FX_INACTIVE_FLANGER=0
FX_STOPPED_FLANGER=1
FX_REGULAR_FLANGER=2
FX_CHORUSED_FLANGER=3
--- Equalizer
FX_INACTIVE_EQUALIZER=0
FX_STOPPED_EQUALIZER=1
```

```
FX_ACTIVE_EQUALIZER=2
FX_ACTIVE_MULTIBAND_EQ=3
--- Panning
FX_INACTIVE_PANNING=0
FX_STOPPED_PANNING=1
FX_FIXED_PANNING=2
FX_LFO_PANNING=3
--- Reverb
FX_INACTIVE_REVERB=0
FX_STOPPED_REVERB=1
FX_DELAY_REVERB=2
FX_MULTIPLE_DELAY_REVERB=3
FX_NATURAL_REVERB=4
--- Bracketing/bracing
SYMBOL_BRACE=0
SYMBOL_BRACKET=1
SYMBOL_BRACKET_UP=2
SYMBOL_BRACKET_DOWN=3
SYMBOL_LINE=4
SYMBOL_NONE=5
SYMBOL_SUB_BRACKET=6
JOIN_LINE_NONE=0
JOIN_LINE_STANDARD=1
JOIN_LINE_MENSURSTRICH=2

---------- STAVES ----------
--- Title Print mode
NEVER_PRINT_NAME=0
PRINT_NAME_ONCE=1
ALWAYS_PRINT_NAME=2

--- Bar numbering
BAR_NUMBERING_NONE=0
BAR_NUMBERING_NUMBER=1
BAR_NUMBERING_LETTER=2
--- Shape note
NO_SHAPE_NOTE=0
SHAPE_4_NOTE=1
SHAPE_7_NOTE_AIKEN=2
SHAPE_7_NOTE_SWAN=2
SHAPE_7_NOTE_WALKER=3
SHAPE_7_NOTE_FUNK=4
SHAPE_7_NOTE_JOHNSON=5
--- Staff type
STAFF_TYPE_STANDARD=0
STAFF_TYPE_DRUM_GRID=2
STAFF_TYPE_TEXT=3
STAFF_TYPE_TEMPO=4
STAFF_TYPE_TABLATURE=7
STAFF_TYPE_DRUM=8
STAFF_TYPE_GREGORIAN=9
STAFF_TYPE_DIGITAL_TRACK=10
--- Clef types
CLEF_TYPE_NONE=0
CLEF_TYPE_TREBLE=1
CLEF_TYPE_BASS=2
CLEF_TYPE_UT=3
CLEF_TYPE_TENOR=3 -- same as above
-- Relative key
ABSOLUTE_KEY=0
PARTIAL_RELATIVE_KEY=1
FULL_RELATIVE_KEY=2
--- Chord/Lyrics frequency
CELL_EVERY_N_BARS=0
CELL_N_TIMES_PER_BAR=1
CELL_EACH_BEAT=2
CELL_EACH_NOTE=3
CELL_EACH_DIVISION=4
--- TRANSPOSITION
TRANSPOSE_PITCH_UNCHANGED=0
TRANSPOSE_UP=1
TRANSPOSE_DOWN=2
TRANSPOSE_DISPLAY_UNCHANGED=3
----------TABLATURES ----------
GUITAR_TABLATURE=0
HARMONICA_TABLATURE=1
ACCORDION_TABLATURE=2
---------- DYNAMICS ----------
-- TYPE --
-- crescendi
DYNAMIC_CRESCENDO=1
DYNAMIC_DECRESCENDO=2
-- power
```

```
DYNAMIC_PPPPPP=26
DYNAMIC_PPPPP=27
DYNAMIC_PPPP=28
DYNAMIC_PPP=3
DYNAMIC_PP=4
DYNAMIC_P=5
DYNAMIC_MP=6
DYNAMIC_MF=7
DYNAMIC_F=8
DYNAMIC_FF=9
DYNAMIC_FFF=10
DYNAMIC_FFFF=31
DYNAMIC_FFFFF=30
DYNAMIC_FFFFFF=29
DYNAMIC_SF=11
DYNAMIC_FZ=12
DYNAMIC_FP=13
DYNAMIC_FPP=14
DYNAMIC_SFZ=15
DYNAMIC_SFFZ=16
DYNAMIC_SFP=25
DYNAMIC_SFPP=17
DYNAMIC_RF=32
DYNAMIC_RFZ=33
-- octave
DYNAMIC_OCTAVE_UP=18          -- 8va
DYNAMIC_OCTAVE_DOWN=19     -- 8vb
DYNAMIC_DOUBLE_OCTAVE_UP=20    --15ma
DYNAMIC_DOUBLE_OCTAVE_DOWN=21   --15mb
-- tempo
DYNAMIC_TEMPO=22
-- pedal
DYNAMIC_PEDAL_ON=23
DYNAMIC_PEDAL_OFF=24
-- TEMPO ALIGNMENT --
TEMPO_HIDDEN=0
TEMPO_DISPLAYED_LEFT=1
TEMPO_DISPLAYED_RIGHT=2
---------- RULES -----------------
RULE_DISPLAY_NEVER=0        -- never displayed
RULE_DISPLAY_ONCE=1         -- only once in the whole score
RULE_DISPLAY_EACH_SYSTEM=2  -- each system (line)
RULE_DISPLAY_ALWAYS=3       -- each time a note matches

------------- STAFF TEXT -----------
COMMENT_STAFF_TEXT=0
LYRICS_STAFF_TEXT=1
CHORD_STAFF_TEXT=2

----------- CURVES -------------
VOLUME_CURVE="Volu"
PANNING_CURVE="Ster"
FREQUENCY_CURVE="Freq"
RESONATOR_RESONANCE_CURVE="Resr"
RESONATOR_FREQUENCY_CURVE="Resf"
OVERDRIVE_POWER_CURVE="Ovep"
OVERDRIVE_TONE_CURVE="Ovet"
FLANGER_POWER_CURVE="Flap"
EQUALIZER_TREBLE_CURVE="Equa"
EQUALIZER_BASS_CURVE="Equg"
REVERBERATION_POWER_CURVE="Revp"
MIDI1_CURVE="Mid1"
MIDI2_CURVE="Mid2"
MIDI3_CURVE="Mid3"
MIDI4_CURVE="Mid4"
MIDI5_CURVE="Mid5"

---- VIRTUAL SINGER LANGUAGE -----
SINGER_LANGUAGE_UNKNOWN=0
SINGER_LANGUAGE_NORTH_FRENCH=1
SINGER_LANGUAGE_SOUTH_FRENCH=2
SINGER_LANGUAGE_UK_ENGLISH=3
SINGER_LANGUAGE_US_ENGLISH=4
SINGER_LANGUAGE_LATIN=5
SINGER_LANGUAGE_SPANISH=6
SINGER_LANGUAGE_ITALIAN=7
SINGER_LANGUAGE_FINISH=8
SINGER_LANGUAGE_JAPANESE=9
SINGER_LANGUAGE_GERMAN=10
SINGER_LANGUAGE_OCCITAN=11

---------- SYMBOLS ----------
--- Accidentals
```

```
NATURAL=0
SHARP=1
AUTO=2
FLAT=3
DOUBLE_SHARP=4
DOUBLE_FLAT=5
--- Shapes
SHAPE_NORMAL=0
SHAPE_TRIANGLE=1
SHAPE_DIAMOND=2
SHAPE_CROSS=3
SHAPE_PLUS=4
SHAPE_SQUARE=5
SHAPE_BEAM=6
SHAPE_NONE=7
SHAPE_FLAG=8
SHAPE_HALF_CIRCLE=9
SHAPE_DROP=10
SHAPE_SAND_GLASS=11
SHAPE_ARC=12
SHAPE_EMPTY_DIAMOND=13
SHAPE_CONE=14
SHAPE_CRESCENT=15
SHAPE_LEFT_TRIANGLE=16
SHAPE_BELL=17
SHAPE_CONCAVE_TRIANGLE=18
SHAPE_CONVEX_TRIANGLE=19
SHAPE_JOHNSON_TRIANGLE=20
SHAPE_JOHNSON_SQUARE=21
SHAPE_CIRCLE_BOLD_CROSS=22
SHAPE_BOLD_CROSS=23
SHAPE_THIN_CROSS=24
SHAPE_HALF_CIRCLE_THIN_CROSS=25
SHAPE_EMPTY=26
--- Stem directions
STEM_AUTO=0
STEM_DOWN=1
STEM_UP=2
---- Note durations
DURATION_64TH=2520
DURATION_32ND=DURATION_64TH*2
DURATION_16TH=DURATION_32ND*2
DURATION_EIGHTH=DURATION_16TH*2
DURATION_QUARTER=DURATION_EIGHTH*2
DURATION_HALF=DURATION_QUARTER*2
DURATION_WHOLE=DURATION_HALF*2
DURATION_BREVE=DURATION_WHOLE*2
DURATION_DOUBLE_BREVE=DURATION_BREVE*2
--- Duration mutipliers
DOTTED=1.5
DOUBLE_DOTTED=1.75
TRIPLET=2/3
--- Slide kinds
SLIDE_NONE=0
SLIDE_SEMITONE=1
SLIDE_SMOOTH=2
SLIDE_CHROMATIC=3
---------- Ornaments ----------
ORNAMENT_HAMMER_ON=1
ORNAMENT_PULLING_OFF=2
ORNAMENT_SLIDE=3
ORNAMENT_SLIDE_UP=4
ORNAMENT_SLIDE_DOWN=5
ORNAMENT_DADDI_PICKING=6
ORNAMENT_CHOKE=7
ORNAMENT_BEND=8
ORNAMENT_RELEASE=9
ORNAMENT_BEND_AND_RELEASE=10
ORNAMENT_MUTED_NOTE=11
ORNAMENT_PALM_MUTE=12
ORNAMENT_VIBRATO_HIT=13
ORNAMENT_UP_MEDIATOR=14
ORNAMENT_DOWN_MEDIATOR=15
ORNAMENT_TAPPING=16
ORNAMENT_SLAP=17
ORNAMENT_POP=18
ORNAMENT_HARMONICS=19
ORNAMENT_ARTIFICIAL_HARMONICS=20
ORNAMENT_TREMOLO=21
ORNAMENT_0=22
ORNAMENT_1=23
ORNAMENT_2=24
ORNAMENT_3=25
```

ORNAMENT_4=26
ORNAMENT_5=27
ORNAMENT_TILDE=28
ORNAMENT_UP_GRUPPETTO=29
ORNAMENT_DOWN_GRUPPETTO=30
ORNAMENT_TRILL=31
ORNAMENT_FLAT_TRILL=32
ORNAMENT_SHARP_TRILL=33
ORNAMENT_NATURAL_TRILL=34
ORNAMENT_VIBRATO=35
ORNAMENT_UP_MORDENT=36
ORNAMENT_DOWN_MORDENT=37
ORNAMENT_UP_DOUBLE_MORDENT=38
ORNAMENT_DOWN_DOUBLE_MORDENT=39
ORNAMENT_THIN_PLUS=40
ORNAMENT_BOLD_PLUS=41
ORNAMENT_CIRCLE=42
ORNAMENT_VOLUME_UP_AND_DOWN=43
ORNAMENT_BRIDGE=44
ORNAMENT_SLUR_ALONE_UP=45
ORNAMENT_SLUR_ALONE_DOWN=46
ORNAMENT_PEDAL_ON=47 -- obsolete
ORNAMENT_PEDAL_OFF=48 -- obsolete
ORNAMENT_FERMATA_UP=49
ORNAMENT_FERMATA_DOWN=50
ORNAMENT_BEAM_1=51
ORNAMENT_BEAM_2=52
ORNAMENT_BEAM_3=53
ORNAMENT_BEAM_4=54
ORNAMENT_SFORZANDO=55
ORNAMENT_SFORZANDO_INVERTED=56
ORNAMENT_VOLUME_UP=57
ORNAMENT_VOLUME_DOWN=58
ORNAMENT_TENUTO=59
ORNAMENT_STACCATO=60
ORNAMENT_MEZZO_STACCATO_UP=61
ORNAMENT_MEZZO_STACCATO_DOWN=62
ORNAMENT_STACCATISSIMO_UP=63
ORNAMENT_STACCATISSIMO_DOWN=64
ORNAMENT_MARCATO_TENUTO_UP=65
ORNAMENT_MARCATO_TENUTO_DOWN=66
ORNAMENT_MARCATO_STACCATO_UP=67
ORNAMENT_MARCATO_STACCATO_DOWN=68
ORNAMENT_MARTELLATO_TENUTO_UP=69
ORNAMENT_MARTELLATO_TENUTO_DOWN=70
ORNAMENT_MARTELLATO_STACCATO_UP=71
ORNAMENT_MARTELLATO_STACCATO_DOWN=72
ORNAMENT_STACCATISSIMO_TENUTO_UP=73
ORNAMENT_STACCATISSIMO_TENUTO_DOWN=74
ORNAMENT_STACCATISSIMO_MARTELLATO_UP=75
ORNAMENT_STACCATISSIMO_MARTELLATO_DOWN=76
ORNAMENT_COMMA_1_SHARP=77
ORNAMENT_COMMA_1_FLAT=78
ORNAMENT_COMMA_4_FLAT=79
ORNAMENT_COMMA_5_SHARP=80
ORNAMENT_COMMA_8_SHARP=81
ORNAMENT_COMMA_8_FLAT=82
ORNAMENT_THUMB=83
ORNAMENT_INDEX=84
ORNAMENT_MIDDLE=85
ORNAMENT_RING=86
ORNAMENT_STRING_0=87
ORNAMENT_STRING_1=88
ORNAMENT_STRING_2=89
ORNAMENT_STRING_3=90
ORNAMENT_STRING_4=91
ORNAMENT_STRING_5=92
ORNAMENT_STRING_6=93
ORNAMENT_MARCATO=94
ORNAMENT_PAUSE=95
ORNAMENT_LONG_PAUSE=96
ORNAMENT_HEEL_PEDAL=97
ORNAMENT_ARPEGGIO=98
ORNAMENT_TEXT=99
ORNAMENT_RULE_TEXT=100
ORNAMENT_STACCATISSIMO_DROP_DOWN=101
ORNAMENT_STACCATISSIMO_DROP_UP=102
ORNAMENT_GRAPHIC=106
ORNAMENT_GOLPE=107
ORNAMENT_ARROW_UP=108
ORNAMENT_ARROW_DOWN=109
ORNAMENT_SLIDE=110

```
--- Ornaments position
ORNAMENT_TOP_STAFF=128
ORNAMENT_BOTTOM_STAFF=128+64
ORNAMENT_TOP_SYMBOL=0
ORNAMENT_BOTTOM_SYMBOL=64
ORNAMENT_LEFT_SYMBOL=32
ORNAMENT_RIGHT_SYMBOL=32+64

-- Slur type --
SLUR_APPEARANCE_STANDARD = 0
SLUR_APPEARANCE_BRACKET=1
SLUR_APPEARANCE_DOTTED_BRACKET=2
SLUR_APPEARANCE_WAVY=3
SLUR_APPEARANCE_DOTTED_CURVE=4
SLUR_APPEARANCE_LINE=5
-----------------------------------------------
-- Column type  for jukebox
JUKEBOX_COLUMN_FILE_NAME=1
JUKEBOX_COLUMN_FOLDER_NAME=2
JUKEBOX_COLUMN_CATEGORIES=3
JUKEBOX_COLUMN_DATE=4
JUKEBOX_COLUMN_LYRICS=5
JUKEBOX_COLUMN_DURATION=6
JUKEBOX_COLUMN_WEB_PAGE=7
JUKEBOX_COLUMN_EMAIL=8
JUKEBOX_COLUMN_FILE_SIZE=9
JUKEBOX_COLUMN_TITLE=10
JUKEBOX_COLUMN_COMPOSER=11
JUKEBOX_COLUMN_COMMENT=12
JUKEBOX_COLUMN_SECONDS_WORKING_ON=13
JUKEBOX_COLUMN_PROTECT_MODE=14
JUKEBOX_COLUMN_ICON=15
JUKEBOX_COLUMN_SOUND_BASE=16
JUKEBOX_COLUMN_ANNOTATION=17
JUKEBOX_COLUMN_PLAYED_BARS=18
JUKEBOX_COLUMN_FILE_KIND=19
JUKEBOX_COLUMN_FILE_VERSION=20
--Icons types for jukebox
JUKEBOX_ICON_STAR=0
JUKEBOX_ICON_SMILEY_LAUGH=1
JUKEBOX_ICON_SMILEY_COOL=2
JUKEBOX_ICON_SMILEY_HAPPY=3
JUKEBOX_ICON_SMILEY_SMILE=4
JUKEBOX_ICON_SMILEY_EMBARRASED=5
JUKEBOX_ICON_SMILEY_ANGRY=6
--- Chord construction
SEMITONE_OFFSET_DOUBLE_DIMINISHED=-2
SEMITONE_OFFSET_DIMINISHED=-1
SEMITONE_OFFSET_MAJOR=0
SEMITONE_OFFSET_AUGMENTED=1
SEMITONE_OFFSET_NONE=2
-- Chord inversion
INVERSION_FUNDAMENTAL=0
INVERSION_1ST=1
INVERSION_2ND=2
INVERSION_3RD=3
INVERSION_4TH=4
INVERSION_1ST_BOTTOM=5
INVERSION_2ND_BOTTOM=6
INVERSION_3RD_BOTTOM=7
INVERSION_4TH_BOTTOM=8
--- Selections modes for List object
LIST_EXTENDDRAG= 64
LIST_NODISJOINT = 32
LIST_NOEXTEND=16
LIST_NORECT=8
LIST_USESENSE=4
LIST_NONILHILITE=1
LIST_MULTIPLE_SELECT=LIST_EXTENDDRAG+LIST_USESENSE
LIST_SINGLE_SELECT=0
----------------- RawTables ---------------
-- Type
RAW_TABLE_FLOAT=0
RAW_TABLE_LONG=1
-- Operations
OP_COPY=0
OP_ADD=1
OP_MULT=2
OP_CONVOLUTION=3
-- Windowing functions
WINDOWING_NONE=0  -- No windowing
WINDOWING_TRIANGLE=1 -- Triangle window
WINDOWING_WELCH=2 -- Welch window
```

WINDOWING_HANNING=3 -- Hanning window
WINDOWING_HAMMING=4 -- Hamming window
---- String encoding ---
NO_STR_ENCODING=-2
UNKNOWN_STR_ENCODING=-1
MS_DOS_STR_ENCODING=0
WINDOWS_STR_ENCODING=1
MACINTOSH_STR_ENCODING=2
HTML_STR_ENCODING=3
UTF8_STR_ENCODING=4
MAC_SHIFTJIS_STR_ENCODING=5
UTF16_BE_STR_ENCODING=6
UTF16_LE_STR_ENCODING=7
-- Carriage return types ---
MAC_CR=0
LINUX_CR=32
WIN_CR=64
UNKNOWN_CR=32+64+128
--- FreeObjects
FREE_OBJECT_TEXT_TYPE=0
FREE_OBJECT_PICTURE_TYPE=1
FREE_OBJECT_SCORE_TYPE=2
FREE_OBJECT_FRAME_TYPE=3
FREE_OBJECT_FILLED_FRAME_TYPE=4
FREE_OBJECT_CIRCLE_TYPE=5
FREE_OBJECT_FILLED_CIRCLE_TYPE=6
FREE_OBJECT_ROUND_FRAME_TYPE=7
FREE_OBJECT_FILLED_ROUND_FRAME_TYPE=8
FREE_OBJECT_HORIZONTAL_LINE_TYPE=9
FREE_OBJECT_VERTICAL_LINE_TYPE=10
FREE_OBJECT_LINE_DOWN_TYPE=11
FREE_OBJECT_LINE_UP_TYPE=12
FREE_OBJECT_CURVED_LINE_UP_TYPE=13
FREE_OBJECT_CURVED_LINE_DOWN_TYPE=14
FREE_OBJECT_CURVED_LINE_LEFT_TYPE=15
FREE_OBJECT_CURVED_LINE_RIGHT_TYPE=16
FREE_OBJECT_BRACE_TYPE=17

FREE_OBJECT_NO_FILE_RELATED=0
FREE_OBJECT_FILE_UP_TO_DATE=1
FREE_OBJECT_FILE_NOT_UP_TO_DATE=2
FREE_OBJECT_FILE_LINK_BROKEN=3

FREE_OBJECT_FROM_START=0
FREE_OBJECT_FROM_END=1
FREE_OBJECT_ALL_PAGES=2
FREE_OBJECT_EVEN_PAGES=3
FREE_OBJECT_ODD_PAGES=4
FREE_OBJECT_AND_NEXT=5
FREE_OBJECT_AND_PREVIOUS=6
FREE_OBJECT_AFTER_LAST=7

FREE_OBJECT_FROM_TOP_OF_PAGE=0
FREE_OBJECT_FROM_BOTTOM_OF_PAGE=1
FREE_OBJECT_FROM_RIGHT_OF_PAGE=2
FREE_OBJECT_FROM_BOTTOM_RIGHT_OF_PAGE=3
-- Pictures
PICTURE_TYPE_STANDARD=0
PICTURE_TYPE_EMBEDDED_PICT=1
PICTURE_TYPE_EMBEDDED_BMP=2
--Lines
SOLID_LINE=0
DASHED_LINE=1
DOTTED_LINE=2
WAVY_LINE=3
SOLID_CURVE=4
DOTTED_CURVE=5

END_NONE=0
END_JOG_UP=1
END_JOG_DOWN=2
END_JOG_BOTH=3
END_ARROW=4

1.13. MSSounds file

1.13.1. MSSounds file Overview

The MSSounds file provides constant definitions of sound ID. This file is provided by default in the application.

```
--- This script display all the sound definitions in the log window
Include "MSSounds"
foreachi(SoundArray,function (i,val) if val then print(i,val) end end)
```

1.13.2. Using MSSounds definitions

Add the following line: **Include "MSSounds.mys"** at the beginning of your script file.
You can omit the extension and write **Include "MSSounds"**.
You can convert an ID to the matching sound name by using : SoundArray[id] where id is the unique ID (number).

```
--- This script display  the name of sounds ID 6000
Include "MSSounds"
Message("Sound ID 6000 name is : "..SoundArray[6000])
```

You can convert a sound name to its matching ID by using : SoundArray[name] where name is the sound name (string).

```
--- This script displays ID of the sound named 'Piano-Detuned EP2'
Include "MSSounds"
Message("sound 'Piano-Detuned EP2' id is : "..SoundArray["Piano-Detuned EP2"])
```

1.13.3. Modifying it

You are not allowed to apply change to this file.

1.14. MSLibrary file

1.14.1. Overview

The MSLibrary file provide MyrScript function definitions to be used in script files. This file is provided by default in the application.

1.14.2. Using it

Add the following line: **Include "MSLibrary.mys"** at the beginning of your script file.
You can omit the extension and write **Include "MSLibrary"**.
After this line, all functions defined in the MSLibrary.mys file are available for your script.
The MSLibrary file is searched first at the same level of the script, then, if not found, in the user's "Includes" folder, then, if not found, in the application "Includes" folder.

1.14.3. Modifying it

You are not allowed to apply change to this file.

1.14.4. Content

From there, you will be able to use the following global functions :

```
-------------------------------------------------
ApplyToAllStaves
In the given score, calls the function for each staff
 Parameters : Score, function(staff)
 Returns :  Number of processed staves
-------------------------------------------------
ApplyToAllSelectedSymbols
In the given score, calls the function for each selected symbol
Symbols are processed from end to beginning
Parameters : Score, function(symbol)
Returns : Number of processed symbols
-------------------------------------------------
ApplyToAllSymbols
In the given score, calls the function for each symbol
Symbols are processed from end to beginning
if a selection exist, only selected symbols are processed
if no selection, the whole score
Parameters : Score, function(symbol)
Returns : number of processed symbols
-------------------------------------------------
ApplyToAllSymbolsOfChosenStaves
For the given score, call the function for each symbol
in the staves chosen by the user
Symbols are processed from end to beginning
Parameters : Score, function(symbol)
Returns : Number of processed symbols
-------------------------------------------------
CopyBarArea
Copies a range of bars from a staff to another (or to the same staff)
Parameters : Source staff, first bar, last bar, target staff, target bar number
Returns : Nothing
-------------------------------------------------
ProcessChord
Calls the note processing function for each symbol in a chord
If you need to insert notes in the currently browsed chord,
it is recommended to insert them in the temporary staff that
is sent as parameter to the processing function, in order not
to process the notes you just inserted.

Notes inserted to the temporary staff are copied into the
original chord once all notes of this chord have been processed.
The time location in the temporary staff does not matter. All notes
of this staff are inserted at the chord position in the original staff
Parameters : first note, processing function
   The remaining parameters are sent "as is" to the processing function
Returns : first symbol that does not belong to this chord
```

1.15. Guidelines

1.15.1. Overview

This section provides hints and tips about how to write a script program. You are free to follow these advices or not.

This chapter is not written yet.

1.16. FAQ

You will find in this chapter frequently asked questions and related answer.

Using DLL calls

Using DLL calls on Windows XP Home Edition would be IMHO nice for MyrScript...
Is it planned, to call external functions from within MyrScript?

By principle, MyrScript is platform independent. A script will work the same way on Windows, Mac OS and UNIX (MacOS X).
So, MyrScript cannot call "dll" entry points, which are Windows-specific.
But, a script can call an external program which will be able to do what needed, by using the OpenFileWithDefaultApplication(...) (page 596) global function.
In this case, of course, the script will be specific to the Windows platform.

2. Manuel Francais

2.1. Manuel du programmeur

2.1.1. Généralités

Harmony Assistant ouvre désormais ses sources pour répondre encore mieux à vos besoins.
Il manque une fonction ? Développez-la, tout simplement.
Le langage MyrScript fournit tout ce dont vous avez besoin pour contrôler les routines internes d'Harmony Assistant.
Vous trouverez ci-dessous tous les renseignements nécessaires pour construire vos propres scripts.
Une première section explique les bases de l'interface graphique.
Vous trouverez une version de ce manuel destinée à l'impression, au format PDF sur
http://www.myriad-online.com/fr/docs/preprinted.htm.
Une seconde section commente tous les objets-types, les valeurs, méthodes et fonctions que vous pouvez utiliser dans vos scripts.
Traduction française : Jean-Pierre Butin.
Relecture : Didier Guillion, Yannick Marchegay.

2.1.2. Comment utiliser le manuel en hypertexte

L'index

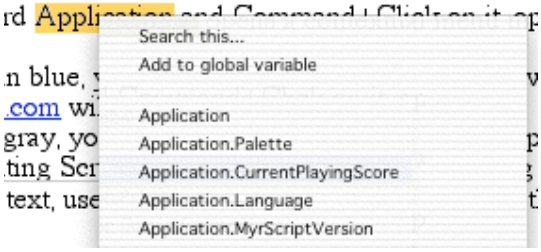
À gauche de cette fenêtre, vous pouvez voir l'intégralité de l'index. Cliquez sur une flèche pour ouvrir ou fermer la section qui lui est associée. Cliquez sur un sujet pour y accéder. La position courante est affichée en bleu.
Quand vous parcourez le manuel, vous pouvez utiliser Commande+I pour fermer toutes les sections ouvertes dans l' index et ouvrir la section en cours.

Recherche dans le manuel

Pour trouver un mot ou une phrase spécifique, utilisez la recherche (page 143).

Aide contextuelle

Pour ouvrir un menu contextuel sur un mot et ainsi afficher les sujets associés, cliquez sur ce mot (avec la touche Commande enfoncée). Vous pouvez également sélectionner un groupe de mots puis faire Commande +clic sur la sélection. Sélectionnez un sujet pour y aller directement.
Par exemple, Commande+clic sur le mot "Application" ouvre un menu contextuel qui affiche tous les mots-clefs liés à ' Application".



Lien internet

Quand un mot est souligné s'affiche en bleu, c'est un lien sur lequel vous pouvez cliquer pour l'ouvrir dans votre navigateur.
Par exemple, http://www.myriad-online.com va ouvrir notre site web.

Saut vers une cible

Quand un mot est souligné et s'affiche en gris, vous pouvez cliquer dessus pour obtenir l'explication de ce mot.
Par exemple : Cliquez ici pour voir le chapitre sur l'édition des scripts (page 120) va sauter à la section "Édition des scripts".
Pour retourner à un emplacement plus haut dans le texte, utilisez l'icône pour ouvrir un menu déroulant qui montre l'historique de vos recherches.
Pour retourner à l'emplacement précédent, faites Commande+Effacement arrière.

Exemples interactifs

Certains exemples MyrScript fournis dans le manuel peuvent être lancés directement. Cliquez simplement sur la flèche bleue .
Par exemple, si vous cliquez sur la flèche bleue ci-dessous, le script va se lancer et va afficher "Bonjour et bienvenue sur MyrScript.".

```
print("Bonjour et bienvenue sur MyrScript.")
```

A noter : si vous cliquez sur la flèche bleue tout en maintenant la touche Majuscule enfoncée, le script ne sera pas lancé, mais chargé dans une nouvelle fenêtre d'édition.

2.1.3. Imprimer le manuel

Pour imprimer ce manuel, faites Commande+"P" (Ctrl+P sous Windows, Pomme+"P" sous Macinstosh)

2.1.4. Dépendance de l'environnement

Un script ne dépend pas de la plate-forme sur laquelle il est utilisé. Le même script peut fonctionner sans modification sous Windows ou Mac OS ou UNIX (MacOS X).
Très peu de caractéristiques sont réservées à une plate-forme en particulier. Par exemple, l'exportation de fichiers graphiques .WMF n'est disponible que sous Windows.

Ces caractéristiques spéciales sont marquées avec cette icône

,

et .

2.1.5. Notes à l'intention du programmeur MyrScript

Merci d'avoir écrit un script. Veuillez garder à l'esprit qu'il peut être utile à beaucoup de monde dans de nombreux pays différents.
Donc, si vous voulez partager votre script, n'oubliez pas de le rendre lisible : utilisez l'anglais comme langue principale puis appelez les fonctions MyrScript pour tester la langue de l'utilisateur.
Vous pouvez alors afficher l'interface graphique de votre script dans la langue demandée. Essayez au moins de fournir une version anglaise de votre script.
Et surtout, n'oubliez pas d'insérer de nombreux commentaires dans votre fichier source. Si vous avez besoin d'aide sur un problème spécifique, l'équipe Myriad vous demandera un script bien commenté.

Enfin, quand votre script fonctionnera bien et que vous voulez le partager avec d'autres utilisateurs, envoyez-le nous en utilisant cette icône .

Voici quelques règles que nous recommandons :

- Comme dans n'importe quel type de programme, un script doit être composé de nombreux commentaires avec quelques instructions. Commentez méticuleusement votre source.
- Utilisez des noms de variables et de fonctions clairs et explicites
- Quand vous déclarez une variable, utilisez le mot-clef local si la variable est locale. Ce genre de variable est directement affichée dans la fenêtre des variables pendant le débogage.
- Enregistrez toujours votre fichier source avant de le tester.

2.2. Le menu *Scripts*

2.2.1. Introduction

Le menu *Scripts* est automatiquement créé par le programme. Il reflète la hiérarchie du dossier "Scripts" du programme en incluant le dossier "Scripts" de l'utilisateur.
Si un script est issu du dossier utilisateur, son nom est signalé par un astérisque rouge.
Si un script est conçu pour fonctionner en arrière-plan, il est indiqué par l'icône.
Si un script existe à la fois dans le dossier utilisateur et le dossier programme, c'est le script "utilisateur" qui est choisi en priorité (le script du dossier programme n'apparaît pas).
Si la version du script "programme" est plus récente que celle du script "utilisateur", c'est le script programme qui sera utilisé.
On peut ajouter de nouvelles catégories en créant des sous-dossiers dans le dossier des scripts "utilisateur". Ce type de sous-dossier est indiqué par l'icône.
Les catégories standard sont indiquées dans le menu par l'icône.

À noter : le menu "Scripts" est reconstruit chaque fois qu'un script est sauvegardé.

2.2.2. Lancement d'un script depuis le menu

Quand on sélectionne un élément du menu *Scripts*, le script est lancé. Si la touche commande est enfoncée (touche Pomme sur MacIntosh, touche Ctrl sous Windows) le script n'est pas lancé, mais seulement chargé dans la fenêtre d'édition. On peut le tester pas à pas ou y appliquer des modifications et créer un nouveau fichier script à partir de cet échantillon.
Quand un script est lancé et qu'il a besoin d'accéder à un autre fichier script (par exemple à l'aide de la fonction globale "Include"), la recherche est faite d'abord au niveau du fichier principal, puis dans le dossier "Includes" du dossier "Scripts" de l'utilisateur et enfin dans le dossier "Includes" du dossier "Scripts" du programme.

2.2.3. Création d'un nouveau script

Dans le menu "Scripts", l'élément *Nouveau Script* crée une fenêtre de script vide.

2.2.4. Ouverture d'un script existant

L'option *Ouvrir* du menu "Scripts" ouvre un script existant dans une nouvelle fenêtre ou installe un script archivé (fichier d'extension .msa).

2.2.5. Arrêt de l'exécution d'un script

L'option *Stop* du menu arrête le script en cours d'exécution. L'utilisateur peut aussi appuyer sur la touche Esc ou faire Commande+Maj+point pour arrêter l'exécution d'un script.
Dans votre script, il vous faut appeler la méthode Application.UserBreak(...) (page 187) de temps en temps, de façon à tester les demandes de l'utilisateur.
Si votre script se bloque, le bouton "Stop" peut s'avérer très utile.

2.2.6. Téléchargement de la dernière version de ce manuel

Ce manuel apparaît dans une fenêtre à l'ouverture du manuel du développeur depuis le menu *Scripts*.
Si le manuel est introuvable, le fichier est téléchargé depuis le site de Myriad.
Il est possible de forcer le téléchargement (par exemple si vous avez été informé par l'équipe Myriad qu'une nouvelle version du manuel est disponible) en sélectionnant la ligne avec la touche Commande enfoncée.
Le fichier du manuel est situé dans le dossier "Docs>Myrscript" du dossier des documents utilisateurs "Myriad Documents".

2.2.7. Version d'un script

Vous pouvez définir la version de votre script en utilisant la balise "-- VERSION :" dans l'en-tête de votre fichier de script.
Le format de version est au format a.b.c dans lequel a, b et c sont des valeurs décimales allant de 0 à 99.
Par exemple :
-- VERSION 1.3.4 signifie que la version principale est 1, la sous-version est 3 et la deuxième sous-version est 4.
Cette version sera utilisée pour gérer la mise à jour et la priorité d'exécution des scripts. Par exemple, si l'utilisateur a installé la version 1.4 d'un script et que la nouvelle version de l'application installe la version 2.0 du même script dans le dossier des scripts par défaut, c'est la version la plus récente qui sera prise en compte. Dans ce cas, la version utilisateur du script sera ignorée.

2.2.8. À propos des scripts...

La ligne "A propos des scripts..." affiche tous les renseignements définis dans le fichier script.
Le développeur doit définir certaines balises dans l'en-tête du fichier source.
Comme les scripts peuvent être utilisés par un non-anglophone, on peut aussi fournir des renseignement dans une autre langue.
Ajoutez les chaînes suivantes au nom de la balise :

- --FR pour le français
- --IT pour l'italien
- --DE pour l'allemand
- --SP pour l'espagnol
- --JP pour le japonais
- --PT pour le portugais
- --NL pour le néerlandais

Voici les balises et leur signification :
--NAME: nom complet du script en anglais
--NAME_IN_MENU: nom affiché dans le menu, en anglais
--CREATOR: nom du créateur du script, en anglais
--DATE: date de création du script, en anglais
--ABSTRACT: une ligne décrivant le script : cette ligne s'affichera dans la fenêtre d'aide
--INFO: description complète du script. Il peut y avoir autant de lignes d'info que vous voulez.
--DIFFUSION_MODE: quand l'utilisateur demande des renseignements sur votre script, le mode de diffusion mis en place pour ce script est affiché dans la langue de l'utilisateur. On peut utiliser les valeurs 1, 2, 3 ou 4

- 1 (par défaut) : l'auteur autorise l'utilisation libre, le partage, les modifications de ce script sans consentement préalable
- 2 : l'auteur autorise l'utilisation libre et le partage du script mais interdit la modification ou l'utilisation d'éléments sans son consentement préalable.
- 3 : l'auteur autorise l'utilisation du script mais interdit le partage, la modification ou l'utilisation d'éléments sans son consentement préalable.
- 4 : l'auteur interdit l'utilisation, le partage ou la modification d'éléments sans son consentement préalable.

A noter qu'un script peut être protégé par mot de passe pour empêcher la consultation de sa source.

Pour les palettes flottantes uniquement:

--MonoInstance empeche l'utilisateur d'ouvrir deux fois la même palette
--TransluscentRatio xxx" où xxxx est le niveau de transparence. 0 signifie opaque. (Windows uniquement)
--NoPalette Si vous voulez que votre palette flottante ne réagisse pas comme une vrai palette.

Certaines balises peuvent être localisées dans d'autres langues. Utilisez :
--NAME -xx
--NAME_IN_MENU(xx
--CREATOR -xx
--DATE-xx
--ABSTRACT-xx
--INFO-xx

dans lesquels "xx" est : FR pour le français, IT pour l'italien, DE pour l'allemand, SP pour l'espagnol, etc.

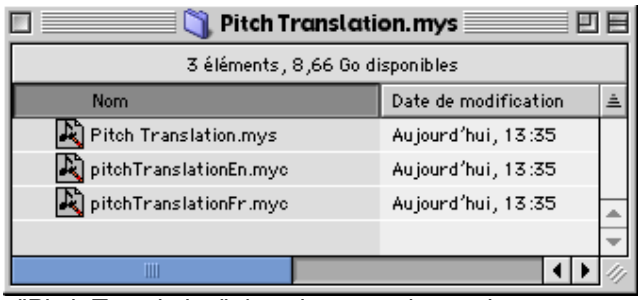
Exemple d'utilisation :
--NAME: Portamento
--CREATOR: Olivier Guillion - Myriad Software
--VERSION: 1.0
--DATE: December, 2002
--DATE-FR: Décembre, 2002
--ABSTRACT: Adds a portamento effect to selected notes
--ABSTRACT-FR: Ajoute un effet de portamento aux notes sélectionnées
--DIFFUSION_MODE: 1
--INFO: Portamento is a smooth transition (pitch slide) between notes.
--INFO: This scripts uses invisible HAMMER and BEND ornaments to do the job.
--INFO: Effect on chords is unpredictable :)
--INFO-FR: Le portamento est une transition douce des fréquences entre les notes
--INFO-FR: Ce script utilise des effets HAMMER et BEND.
--INFO-FR: Le résultat sur des accords peut être non conforme.

2.2.9. Cacher un script sur un type donné d'ordinateur

Si votre script dépend totalement de la plate-forme d'utilisation (par ex. Macintosh uniquement), vous pouvez éviter qu'il apparaisse sur un autre type d'ordinateur en utilisant la balise PC_ONLY ou MAC_ONLY dans votre en-tête de script.

2.2.10. Scripts empaquetés

Vous pouvez définir un jeu de fichiers scripts qui sont conçus pour travailler ensemble. C'est ce qu'on appelle un **script empaqueté**. Cela permet d'éclater le script en plusieurs parties.
Un script empaqueté est un dossier qui contient plusieurs fichiers de scripts et d'Interface Composer.
Le nom du dossier se termine par l'extension ".mys".
Il doit contenir un fichier script de même nom. Ce script sera considéré comme le script principal.
Par exemple, voici un script empaqueté élaboré à partir de 3 fichiers scripts et de 2 fichiers Interface Composer.



Il sera affiché sous le nom de "Pitch Translation" dans le menu des *scripts*.
Quand vous voulez envoyer un script empaqueté à quelqu'un, le meilleur moyen est de créer une archive. Chargez le fichier script principal et cliquez sur l'icône .Vous créez ainsi un fichier Archive Myrscript qui porte l'extension **.msa**. Cette archive inclut tous les fichiers présents dans le dossier.
Quand un autre utilisateur ouvre cette archive (par exemple en utilisant le menu *"Script>Ouvrir..."*) l'archive est automatiquement installée au bon endroit.

2.2.11. Ajout de scripts dans les menus standard

Un script peut apparaître dans certains menus standard de l'application. Dans ce cas, le script apparaîtra deux fois, une fois dans le menu Scripts et une fois à l'emplacement choisi.
Il suffit d'ajouter la balise :
"-- MENU_LOCATION : xxxxx" dans votre script, xxxxx étant l'une des chaînes suivantes :

"File>Export"	"File>Import"	"File>Info"
"File>Print"	"Edit"	"Edit>Select"
"Edit>Search"	"Edit>Bars"	"Edit>Appearance"
"Edit>Pitch"	"Edit>Quantize"	"Edit>Actions"
"Edit>Mirror"	"Edit>Multi-Voices"	"Edit>Digital audio track"
"Edit>Free objects"	"Score"	"Score>Page mode"
"Score>Apply"	"Staff"	"Instruments"
"Options"	"Configuration"	"Windows>Zoom"
"Windows>Palettes"	"Windows>Custom tool"	"Windows>Custom tools"
"Windows>Mixer"		

Par exemple, "--MENU_LOCATION : File>Export" ajoutera votre script dans le menu Fichier>Exporter

La ligne du menu est marquée avec le symbole s'il s'agit d'un script standard ou s'il s'agit d'un script utilisateur.

2.2.12. Raccourci clavier

L'utilisateur peut définir une touche du clavier pour lancer un script.

2.2.13. Recherche de scripts

Depuis le menu *Scripts*, vous pouvez lancer une recherche sur tous les fichiers sources. Les résultats sont affichés dans une fenêtre log (page 145)

2.2.14. À noter

Certains sous-dossiers ne sont pas affichés dans le menu *Scripts* :
- les dossiers "Includes",
- tout dossier dont le nom commence par "(", ce qui permet de cacher temporairement certains scripts,
- les scripts empaquetés : ils sont affichés comme un script simple.

2.3. Édition de scripts

2.3.1. Généralités

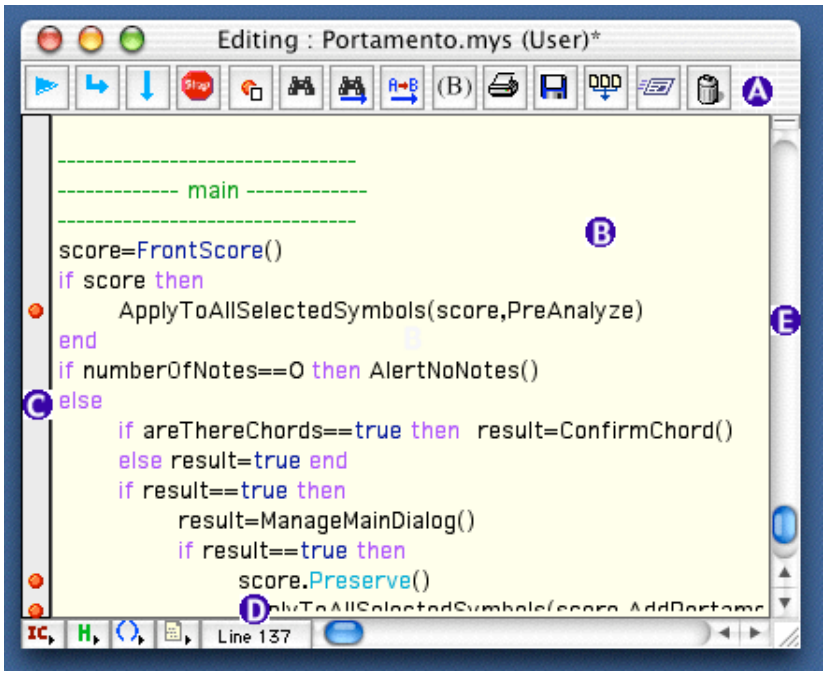
Ce chapitre fournit tous les renseignements sur la manière de créer, éditer, imprimer, sauvegarder, lancer ou effacer un fichier script.

2.3.2. Qu'est-ce qu'un script ?

Un script est un fichier ; plus précisément, un fichier texte. Vous pouvez éditer un fichier script avec n'importe quel éditeur de texte, mais ce n'est pas recommandé : le script est encodé au standard Macintosh.
Un script **doit** se trouver soit dans le dossier "Scripts" du programme, soit dans le dossier *Scripts* de l'utilisateur. Quand vous créez un script, il est automatiquement sauvegardé dans le dossier *Scripts* de l'utilisateur. Quand vous éditez un script prédéfini, une copie sera enregistrée dans votre dossier *Scripts*. Votre script s'affichera dans le menu Scripts à la place du script original.

Ne modifiez jamais, (ni déplacez ou supprimez) un script programme par défaut.

Voici un aperçu d'un script en cours d'édition :



A Icônes permettant de lancer, déboguer, rechercher, enregistrer, effacer, archiver, publier le fichier script.
B Texte en cours d'édition
C Zone des points d'arrêt
D Le placard
E La barre de défilement et la zone de partage de la fenêtre. Vous pouvez diviser la fenêtre verticalement en quatre zones.

2.3.3. Création d'un script

Dans le menu *Scripts*, choisir *Nouveau*. Un script sans nom est créé dans une nouvelle fenêtre. On peut également démarrer à partir d'un script existant en utilisant *Ouvrir* puis en enregistrant sous un autre nom avec *Fichier>Enregistrer sous...*

2.3.4. Édition d'un nouveau fichier script

Un fichier script s'édite comme un fichier texte normal, en utilisant les flèches du clavier pour bouger le curseur, la souris pour sélectionner une zone, etc.
Voici la liste des commandes clavier spéciales :

Début : va en début de ligne
Fin : va en fin de ligne
Insert : bascule entre les modes remplacement/insertion. Le curseur est gras quand le mode insertion est actif.
Effacement arrière : efface le caractère précédent
Suppr : efface le caractère suivant
Maj+flèche gauche : étend la sélection vers la gauche
Maj+flèche droite : étend la sélection vers la droite
Maj+flèche haute : étend la sélection vers le haut
Maj+flèche basse : étend la sélection vers le bas
Tab : si une sélection est en cours, décale tout le bloc sélectionné vers la droite en insérant une tabulation avant chaque ligne. Si rien n'est sélectionné, Tab insère une seule tabulation.
Maj+Tab : décale toute sa sélection d'une tabulation vers la gauche en enlevant un taquet à chaque ligne
Commande+'+' : augmente la taille de la police de caractères
Commande+'-' : diminue la taille de la police de caractères
Commande+'P' : imprime l'ensemble du texte
Commande+'F' : recherche une chaîne dans le texte
Commande+'G' : recherche l'occurrence suivante d'une chaîne dans le texte
Commande+'L' : remplace et recherche l'occurrence suivante de la chaîne
Commande+'S' : enregistre le nouveau script sur le disque

Note : Commande désigne la touche Ctrl sous Windows et la touche Pomme sur Macintosh

Voici la liste des commandes souris spéciales :
Double-clic : sélectionne le mot entier
Triple-clic : sélectionne la ligne entière
Commande+clic sur la sélection : ouvre un menu déroulant qui affiche tous les sujets apparentés. On peut alors aller directement à un sujet en le sélectionnant.
Le premier élément du menu déroulant ajoute le mot-clef sélectionné dans la fenêtre des variables globales (voir le chapitre frdebug (page 146))
Commande+clic dans le titre : ouvre un menu déroulant qui indique l'emplacement du fichier. Il est possible de sélectionner une entrée pour ouvrir le dossier correspondant.

2.3.5. Impression d'un script

Cliquer sur l'icône Imprimante . Lisez le chapitre impression (page 144) pour une description complète)

2.3.6. Protection d'un script

Un script protégé peut être exécuté mais pas lu sans mot de passe. On peut protéger des scripts standard, des fichiers Interface Composer ou des scripts de fenêtres flottantes.

Pour protéger/déprotéger un script, cliquer sur le cadenas en haut de la fenêtre d'édition.
Quand un script est basculé en mode protégé, un mot de passe est demandé. N'oubliez pas ce mot de passe car il sera demandé à chaque chargement du fichier source.
Le mot de passe n'est pas demandé quand un script est lancé. Il ne l'est que si on tente d'éditer le script.
Le fichier source d'un script protégé est crypté sur le disque, pour interdire l'emploi d'un éditeur de texte externe.

2.3.7. Démarrage à partir d'un script existant

Vous ne pouvez pas éditer directement un script prédéfini, mais vous pouvez en utiliser une copie et y faire des changements. Sélectionnez un script prédéfini dans le menu Scripts en laissant la touche Commande enfoncée.
Après un message de confirmation, le script est recopié et devient un script utilisateur. Il est alors possible de le modifier. Dans le menu, le script utilisateur prend la place du script prédéfini.
Quand le script prédéfini concerne d'autres fichiers, comme les fichiers de l'Interface Composer, des fichiers images ou des fichiers de données, tous ces fichiers sont également copiés dans la zone utilisateur.

2.3.8. Le placard

Il se situe dans le coin gauche, au bas de la fenêtre. Il y a quatre zones dans le placard.

Zone d'Interface Composer

Affichée comme un "IC" rouge . Lorsqu'on clique dessus, la liste de tous les fichiers de l'Interface Composer liés à script s'affiche (fonction CreateDialog("nom de fichier"))
La sélection d'une ligne de ce menu ouvre le fichier Interface Composer correspondant.

Zone d'inclusion
Affichée comme un "H" vert . Lorsqu'on clique dessus, la liste de tous les fichiers inclus (fonction 'Include "nom fichier") liés au script en cours s'affiche.
La sélection d'une ligne de ce menu ouvre le fichier correspondant.

Zone de fonctions
Affichée comme des "()" bleues . Lorsqu'on clique dessus, la liste de toutes les fonctions définies dans le script s'affiche.
Pour afficher une fonction, sélectionnez-la dans ce menu.

Zone de commandes
Affichée comme une icône jaune . Lorsqu'on clique dessus, une liste des diverses commandes s'affiche.

Zone cible
Affichée comme une icône rouge . Lorsqu'on clique dessus, la liste des dernières positions dans le texte s'affich
La sélection d'une position cale l'éditeur sur la ligne concernée.

Zone de numéro de ligne
Montre la position actuelle du curseur. En cliquant dessus, on change la position du curseur.

2.3.9. Enregistrement d'un script

Pour enregistrer un fichier script, on peut utiliser soit l'icône en haut de la fenêtre du script, soit *Fichier>enregistrer* o encore Commande+"S".
Quand on enregistre un script, si le fichier existe déjà, une copie est générée avec l'extension ".bks" (il faut avoir activé cette fonction dans le paramétrage général).

2.3.10. Scripts et palette utilisateur

L'utilisateur peut ajouter un script dans la palette utilisateur en le sélectionnant avec la touche Maj enfoncée. Le script est affiché à l'aide d'un bouton dont le contenu est le nom du script. Une image peut être substituée à ce nom : il faut alors fournir un fichier .bmp ou .pct qui porte le même nom que le script et l'enregistrer au même emplacement que le script.

2.3.11. Création d'une archive MyrScript

Généralités

Un fichier MyrScript est un fichier binaire compacté qui contient un ou plusieurs fichiers. L'extension de ce fichier est **.msa**
Vous créez un fichier archive quand vous voulez envoyer à quelqu'un un projet complet, comprenant des fichiers scripts, des fichiers Interface Composer, des fichiers images ou n'importe quel fichier de données.
Quand on ouvre une archive MyrScript (par exemple avec *Scripts>Ouvrir...*), les fichiers qu'elle contient sont automatiquement copiés au bon endroit, en créant le dossier et les sous-dossiers si nécessaire.
Par exemple, si vous créez un nouveau script appelé "Ukulele.mys" et que vous l'enregistrez dans "Scripts\Notation\Fretted\Ukulele.mys\Ukulele.mys" le chemin "Scripts\Notation\Fretted\Ukulele.mys\" sera créé sur l'ordinateur.
Si vous créez une archive à partir d'un fichier script qui **n'est pas** situé dans le dossier "Scripts", il sera installé dans le dossier "Scripts\Unsorted".
Si un ou plusieurs fichiers existent déjà au même emplacement sur l'ordinateur, une confirmation est demandée à l'utilisateur avant d'écraser les fichiers.

Création d'une archive

Quand le fichier archive principal d'un projet est ouvert, l'icône vous permet de créer un fichier Archive MyrScript c contient tous les fichiers du projet.
Il vous est demandé un nom pour l'archive MyrScript. Vous pouvez entrer le nom que vous voulez.
A noter : pour un transfert via internet, il est recommandé d'utiliser un nom court (moins de 23 caractères).

Archivage des scripts empaquetés

Quand vous créez une archive d'un fichier empaqueté (un dossier script portant le même nom) l'archive inclut tous les fichiers présents dans le dossier et le sous-dossier sauf :

- les fichiers extérieurs au dossier sont ignorés (par exemple les fichiers inclus avec la fonction Include),
- les copies de sécurité (backup) qui portent l'extension **.bks** ou **.bkc** sont ignorées,
- les fichiers archives (portant l'extension **.msa**) sont ignorés.

Archivage de scripts non-empaquetés

Le fichier principal est archivé ainsi que les fichiers situés au même emplacement et utilisés dans le fichier script principal.

- Le fichier de définition de script inclus avec la fonction Include (seulement si situé au même emplacement).
- Le fichier script lancé par la fonction CreateDialog(...) (page 496) (seulement si situé au même emplacement).
- Le fichier Interface Composer (seulement si situé au même emplacement).

Étant donné que les fichiers non-empaquetés mélangent tous les fichiers situés au même emplacement et peuvent ainsi générer des conflits de noms, l'utilisation de fichiers empaquetés est recommandé.

Note à l'attention des programmeurs sur Macintosh : quand l'archive est construite, seule la fourchette de données archivée et la fourchette ressources est exclue. Si vous fournissez des fichiers images avec votre projet, utilisez donc toujours le format fichier PICT et **non pas** le format ressource PICT.

Format du fichier archive

Voici la description du format archive. Toutes les valeurs sont codées au format Macintosh (codage étendu)

En-tête d'archive
- ·MAGIC_HEADER, 4 octets, 'MsAr' Magic
- ·CHECKSUM, 4 octets : checksum (somme de tous les octets qui suivent la checksum jusqu'à la fin de l' archive, y compris le pied de page
- ·VERSION : 2 octets : version de l'archive (0x0100)
- ·COMPACT_MODE, 1 octet : 0 si l'archive n'est pas empaquetée, 1 si elle l'est

(lié au mode d'archivage défini dans le paramétrage général)
- ·SIZEOF_MAIN_FOLDER_NAME, 1 octet : taille de la chaîne qui décrit le dossier de destination
- ·MAIN_FOLDER_NAME, SIZEOF_MAIN_FOLDER_NAME , octets : chaîne décrivant le dossier de destination.

Pour chaque fichier (peuvent être empaqueté selon le mode d'empaquetage COMPACT_MODE)
- · SIZEOF_FILE_NAME, 1 octet: taille du nom de fichier ou 0 s'il n'y en a pas d'autre
- · FILE_NAME, SIZEOF_FILE_NAME , octets : chaîne, nom de fichier
- · SIZEOF_SUBFOLDER_NAME, 1 octet: taille du nom du sous-dossier où se trouve le fichier, 0 signifiant la racine du dossier principal (MAIN_FOLDER_NAME)
- · SUBFOLDER_NAME, SIZEOF_SUBFOLDER_NAME , octets : chaîne, nom du sous-dossier
- · SIZEOF_FILE, 4 octets, longueur des données non empaquetées
- · FILE_DATA, données empaquetées

Pied de page d'archive (non-empaqueté)
- ·MAGIC_FOOTER 4 octets, 'MsAr' Magic

2.3.12. Proposition de script

Si vous voulez proposer un script pour le partager avec d'autres utilisateurs ou si vous avez un problème avec ce script et que vous désirez les conseils de l'équipe Myriad, cliquez sur l'icône . Votre projet sera alors archivé dans un fichier temporaire et envoyé à notre adresse électronique.

2.3.13. Annuler/Refaire

Vous pouvez annuler/refaire les modifications effectués sur un fichier en utilisant la combinaison Commande+Z ou Commande+Y ou les raccourcis dans le menu *Edition*.
Il faut avoir autorisé l'option Annuler/Refaire dans les préférences générales.

2.3.14. Effacement d'un fichier de script

Pour effacer un fichier script, utilisez l'icône "corbeille" en haut de la fenêtre du script. Un message demande confirmation. Vous ne pouvez pas effacer un script prédéfini.

2.3.15. Équilibrage syntaxique

L'équilibrage syntaxique est un outil utile pour vérifier la syntaxe de votre script. Cliquez n'importe où dans votre fichier script et cliquez sur l'icône (B) en haut de la fenêtre du script. Le niveau sera mis en surbrillance.
Vous pouvez alors utiliser le raccourci Commande+"B".
Conseil : vous pouvez utiliser l'équilibrage syntaxique pour l'indentation de votre source : choisissez un niveau, " équilibrez-le" et utilisez la touche Tab pour augmenter la tabulation ou Maj+Tab pour la diminuer sur l'ensemble du niveau sélectionné.

2.3.16. Couleurs et syntaxe

L'utilisation des couleurs vous permet de voir rapidement si les mots-clefs que vous avez introduits sont connus ou non.
Voici la signification des différentes couleurs par défaut :
Vert foncé : commentaire (un commentaire commence toujours par --)
Gris foncé : chaîne (une chaîne commence et se finit par les guillemets ")
Violet : mot-clef réservé dans le langage Lua
Bleu foncé : fonctions globales de MyrScript
Bleu moyen : méthodes et champs
Rouge : erreur de syntaxe. Par exemple, une méthode non suivie de (), un champ non précédé d'un point (.), etc.

Pendant le débogage d'un script, les noms des variables sont mis en couleurs :
Orange : variable locale
Marron : variable globale

Pour plus de renseignements, reportez-vous au chapitre frdebug (page 146).
Le statut de la coloration syntaxique est sauvegardé dans le fichier script.

La coloration syntaxique permet de localiser la plupart des erreurs de syntaxe pendant la saisie du script. Par exemple, si vous voulez créer une portée dans une partition, vous taperez :

```
score.NewStaff
```

Comme vous pouvez le voir, "NewStaff" est affiché en noir, donc ce n'est pas un mot-clef. Vous auriez dû taper " AddStaff" !

Donc, vous écrirez :

```
score.AddStaff
```

"AddStaff" est affiché en rouge. Cela veut dire que c'est un mot connu, mais que la syntaxe est mauvaise. Dans ce cas, c'est parce que AddStaff est une méthode, qui doit donc être suivie de ()

Donc, vous écrirez :

```
score.AddStaff()
```

Le mot-clef "AddStaff" est affiché en bleu, la syntaxe est correcte.
Notez cependant que la coloration syntaxique ne vérifie pas la validité des paramètres.
De la même façon, elle vérifie seulement si le nom d'un membre est connu ou pas. Elle ne s'assure pas que ce membre est utilisé sur le bon type d'objet.

Vous pouvez changer les couleurs par défaut dans le paramétrage général, panneau des scripts.

2.3.17. Exécution d'un script

Pour lancer un script, cliquez sur la flèche bleue en haut de la fenêtre du script ou utilisez le raccourci clavier Commande+'R'.
Vous pouvez aussi sélectionner le script dans le menu, mais dans ce cas, les points d'arrêts (page 146) ne sont pas pris compte.
Vous pouvez arrêter l'exécution en cliquant sur l'icône "Stop" dans *n'importe laquelle* des fenêtres script ouvertes.

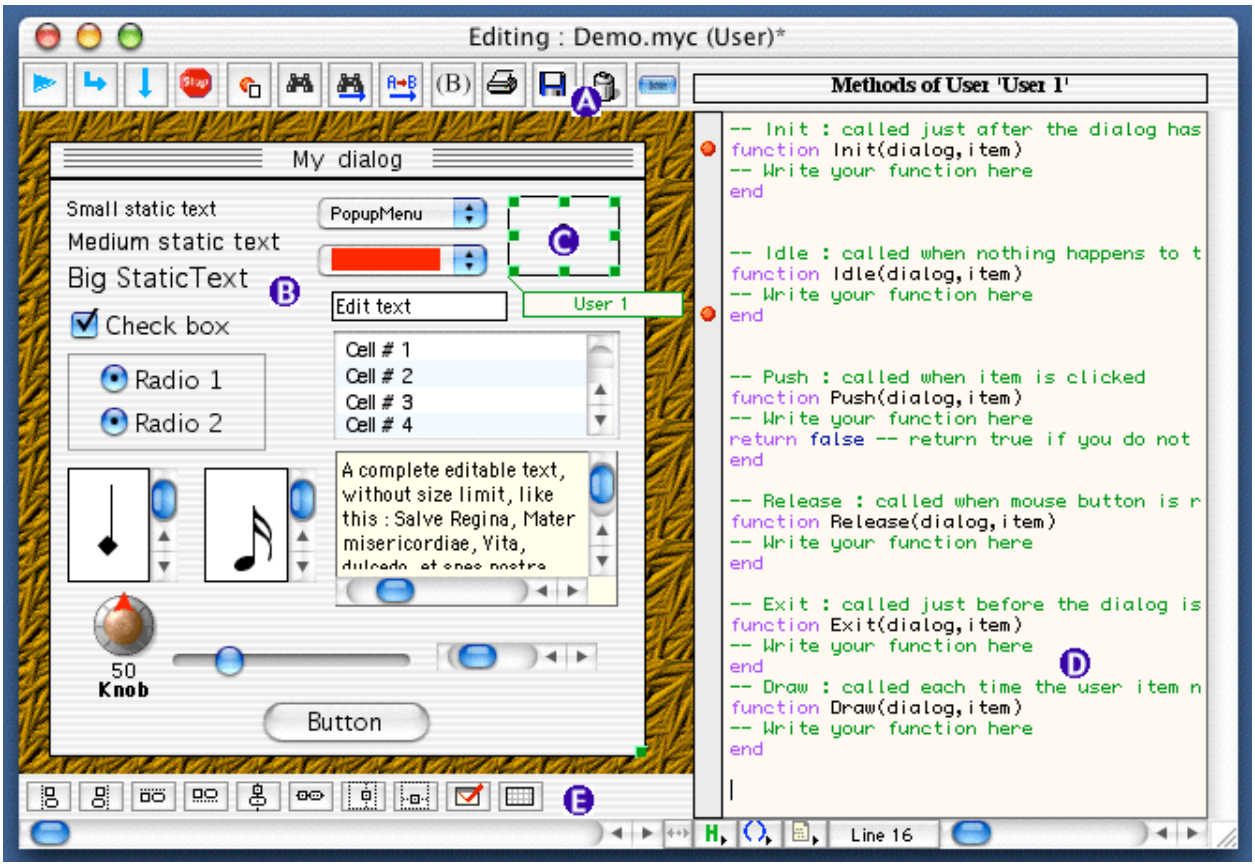
2.3.18. Ajout et suppression de points d'arrêt

Cliquez dans la marge de droite pour ajouter un point d'arrêt (point rouge). Cliquez dessus à nouveau pour l'effacer.
Vous pouvez effacer tous les points d'arrêt en utilisant l'icône "Effacer les points d'arrêt" . Pour plus de renseignemen voyez le chapitre Débogage (page 146).

2.4. L'Interface Composer

2.4.1. Généralités

Quand un script a besoin que l'utilisateur entre des valeurs ou des paramètres, il peut utiliser un objet Dialogue. L'Interface Composer permet de définir graphiquement des boîtes de dialogue ainsi que les diverses actions à appliquer quand l'utilisateur interagit avec elle. L'Interface Composer est un module intégré de Myrscript.
Un fichier Interface Composer comprend toutes les définitions des objets et les scripts qui leur correspondent.
Voici une vue de l'Interface Composer.



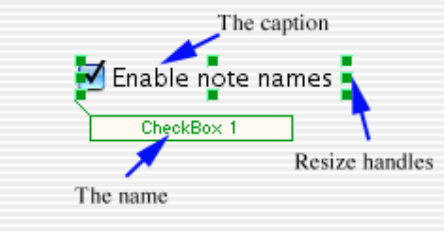
A Les icônes qui permettent de lancer, déboguer, enregistrer, effacer ou parcourir le fichier de l'Interface Composer.
B Vue de la boîte de dialogue en cours d'édition.
C Le composant en cours d'édition. Ses méthodes sont affichées dans la partie droite de la fenêtre.
D Les méthodes associées à l'objet sélectionné.
E Ces boutons permettent d'aligner les objets sélectionnés ou de générer un rapport sur le contenu de la boîte de dialogue.

2.4.2. Qu'est-ce qu'une boîte de dialogue ?

Un objet "dialog" est une fenêtre avec un titre et des **composants** à l'intérieur. Une boîte de dialogue peut être déplacée sur l'écran par l'utilisateur, en cliquant sur son titre. Une boîte de dialogue peut contenir autant de composants que nécessaire. Une boîte de dialogue peut ouvrir une autre boîte de dialogue et ainsi de suite.
Quand on sort du script, toutes les boîtes de dialogue créées par le script sont fermées automatiquement. Les boîtes de dialogue créées par d'autres scripts ne sont pas affectées.

2.4.3. Qu'est qu'un composant (item)?

Un composant est un objet graphique qui représente des valeurs. Un composant est toujours intégré à un objet dialogue. Le type de valeur contenue dans un composant varie d'un type de composant à l'autre.
Par exemple, une case à cocher (qui peut être cochée ou pas) contient une valeur booléenne. Elle peut être vraie (case cochée) ou fausse (case non cochée).



Votre script principal peut changer ces valeurs en utilisant les valeurs du composant et ses méthodes. Des scripts associés à des composants peuvent aussi changer ces valeurs ou même les valeurs d'autres composants.
Un composant est défini par : la zone qu'il occupe dans la boîte de dialogue, le texte affiché (si nécessaire), son nom, son état ... Dans une même boîte de dialogue, deux composants ne peuvent pas avoir le même nom, parce que le nom est le seul moyen que le script a de l'identifier.
Quand vous ajoutez un nouveau composant à une boîte de dialogue, ou que vous y collez des composants, un nom par défaut est donné, mais vous pouvez le changer par la suite.
Un composant peut actif ou inactif, caché ou affiché.

A noter : ne confondez pas le texte affiché dans le composant et le nom du composant. Le premier est purement graphique alors que le second n'est pas visible par l'utilisateur et représente la valeur chaîne que le script peut utiliser pour agir sur le composant.

2.4.4. Qu'est-ce qu'un fichier de l'Interface Composer ?

Un fichier de l'Interface Composer est un fichier texte qui porte l'extension **.myc.** Vous pouvez charger un fichier Interface Composer existant en utilisant le menu *Scripts>Ouvrir* ou en utilisant l'icône du placard du script principal : tous les fichiers de l'Interface Composer utilisés par le script principal sont alors affichés dans un menu déroulant.
Pour créer un nouveau fichier Interface Composer, utilisez *Scripts>Nouvelle Interface Composer...* Une boîte de dialogue qui vient d'être créée comprend deux objets définis : un bouton "OK" nommé "OK" et un bouton "Annuler" nommé "Cancel".
Quand on le charge ou qu'on le crée, le fichier est affiché dans une nouvelle fenêtre. Vous pouvez ouvrir autant de fenêtres Interface Composer que vous voulez.

2.4.5. Edition

La fenêtre de l'Interface Composer est divisée en deux parties. A gauche, vous pouvez voir un aperçu de la boîte de dialogue et des boutons de commande. A droite, vous pouvez voir le script correspondant à l'objet en cours de sélection.
Vous pouvez copier, coller, effacer des composants en utilisant les raccourcis clavier standards. Les commandes défaire, refaire sont disponibles pendant l'édition d'une telle boîte de dialogue.

Important : ce n'est qu'un aperçu. L'apparence de la boîte de dialogue changera en fonction du système sur lequel le script est lancé.

Edition de la boîte de dialogue

Pour **sélectionner** un composant, cliquer dessus. Quand il est sélectionné, six marques vertes entourent la zone du composant (poignées). En dessous, vous pouvez voir une petite zone flottante encadrée de vert : cette zone affiche le nom du composant. Vous pouvez double-cliquer dessus pour changer le nom du composant.
Pour **sélectionner plus d'un composant à la fois**, cliquez sur les composants avec la touche Maj enfoncée ou cliquez en dehors d'un composant pour définir une aire de sélection.
Pour **désélectionner un composant**, cliquez sur lui une deuxième fois.
Pour **désélectioner tous les composants**, cliquez en dehors d'un composant.
Pour **redimensionner un composant ou la boîte de dialogue**, cliquez sur une des poignées vertes et étirez le cadre.
Pour **éditer le texte** affiché par un composant, double-cliquez dessus.
Pour **déplacer un composant**, tirez-le avec la souris. Si plusieurs composants sont sélectionnés, ils se déplaceront alors ensemble.
Pour **effacer un composant** tirez-le en dehors de la boîte de dialogue ou utilisez le raccourci Commande+X. Tous les raccourcis clavier d'édition sont disponibles lors de l'édition d'une boîte de dialogue : copier, coller, effacer. Vous pouvez copier ou coller du texte ou des composants d'une fenêtre Interface Composer à une autre. On peut annuler/refaire les changements.
Pour **aligner les composants**, il faut les sélectionner avec Maj+Clic, puis cliquer sur les icônes en bas de la fenêtre d'édition. On peut aussi déplacer un composant, des lignes bleues verticales ou horizontales vous montrant alors les alignements possibles.
Pour **ajouter un nouveau composant**, sélectionnez le type dans la palette, puis faites un glisser/déplacer sur l'emplacement de la boîte de dialogue où vous voulez le placer (bouton de souris enfoncé pendant la manoeuvre).
La palette affichant tous les types de composants s'ouvre automatiquement quand un fichier d'interface Composer est chargé ou créé. Vous pouvez aussi l'ouvrir en utilisant l'icône
Pour **changer le titre d'une boîte de dialogue**, double-cliquez dessus.

Edition du script lié à un composant

Quand un composant est sélectionné, le script qui s'y rattache est affiché dans la partie droite de la fenêtre d'édition. Vous pouvez alors l'éditer ou ajouter des points d'arrêts.

Edition d'un script lié à la boîte de dialogue

Cliquez sur le titre de la boîte de dialogue dans l'aperçu. Le script correspondant s'affiche.

2.4.6. Enregistrer un fichier Interface Composer

Pour enregistrer un fichier Interface Composer, vous pouvez utiliser l'icône en haut de la fenêtre du script ou faire *Scripts>Enregistrer*, ou bien utiliser le raccourci clavier Commande+S.
Quand vous enregistrez un fichier Interface Composer, si le fichier existe déjà, une copie de sauvegarde (backup) est créée avec l'extension .bkc. (Cette option doit être activée dans le paramétrage général).

2.4.7. Débogage

Vous pouvez déboguer une boîte de dialogue de l'Interface Composer. Utilisez l'icône ou le raccourci Command+R. Un aperçu du dialogue s'affiche.Lors du débogage, le fichier "MSDefine (page 100).mys" est automatiquement chargé de façon à ce que vous puissiez utiliser les constantes prédéfinies.
Vous pouvez cliquer sur n'importe quel composant. Le nom du composant s'affiche dans la fenêtre log (compte-rendu).
Pour arrêter l'exécution, utilisez l'icône "Stop".
Si des points d'arrêt sont ajoutés dans un script lié à un composant, ils seront pris en compte. Vous pouvez avancer pas à pas dans une méthode.
Si vous appuyez sur la touche Esc pendant l'exécution d'un dialogue, l'exécution est mise en pause et la ligne en cours s'affiche.
Pour ouvrir et utiliser une boîte de dialogue depuis un script, utilisez la fonction CreateDialog. Par exemple *mondialogue=CreatDialog("Mydialog.myc")* va créer une nouvelle boîte de dialogue à partir du fichierMyDialog.myc et stocker le résultat dans la variable *Mydialog*. Le fichier de l'Interface Composer doit se trouver au même niveau que le fichier script qui l'utilise.Il est recommandé d'utiliser des scripts empaquetés pour éviter des conflits entre deux scripts qui utilisent le même nom de fichier Interface Composer.
Beaucoup plus intéressant, le stockage des fichiers scripts et des fichiers Interface Composer qui lui sont attachés dans le même dossier permet de créer facilement des archives MyrScript.

2.4.8. Fermeture d'une boîte de dialogue

Utilisez simplement la méthode Dialog.Dispose(...) (page 503)
Veuillez noter que si le script se termine à la suite d'une erreur, les boîtes de dialogues sont automatiquement fermées.

Les méthodes sont des fonctions associées à un objet qui décrivent la manière dont l'objet va réagir. Certaines méthodes sont prédéfinies (Init, Exit, Idle, Push, Release...) et sont appelées quand certains événements se produisent. Par exemple, la méthode Idle est appelée plusieurs fois par seconde, la méthode Init est appelée à l'ouverture de la boîte de dialogue, etc.
Vous trouverez dans ce chapitre une description précise des méthodes.

Pour éditer les méthodes associées à un composant, cliquez sur le composant. Le script associé s'affiche à droite de la fenêtre.
Pour éditer les méthodes associées à une boîte de dialogue, cliquez sur sa barre titre.
Vous pouvez ajouter ou supprimer des points d'arrêt dans les méthodes.

Quelques conseils :

·Nous avons vu qu'il ya des méthodes par défaut, mais vous pouvez aussi définir vos propres méthodes dans un objet puis les appeler.

Donc, dans le script associé à un objet, vous pouvez insérer :

```
function myTest()
    print ("Test")
end
```

Mais, par exemple, ceci ne va pas marcher dans un script associé à un objet :

```
function myTest()
    print ("Test")
end
function Idle(dialog,item)
    myTest()
end
```

Vous allez obtenir l'erreur suivante :
error : attempt to call global 'myTest' (a nil value)
=>Tentative d'appeler une fonction globale Mytest (valeur nulle)

Parce que **myTest** est une méthode qui appartient à l'objet item et ne peut pas être appelée sans référence à l'objet.
Il faut donc écrire :

```
function myTest()
    print ("Test")
end
function Idle(dialog,item)
    item.myTest()
end
```

·Pour la même raison, on ne peut pas commencer une définition de script par :

```
    Include "MSDefine"
```

parce qu'elle ne sera jamais appelée.

Il vous faut écrire la commande "Include" dans la méthode "Init" de la boîte de dialogue (ou dans la méthode Init d'un objet) comme ceci :

```
function Init(dialog)
    Include "MSDefine"
end
```

· Mais vous pouvez aussi appeler la méthode d'un objet depuis un autre objet. Par exemple, ce script va appeler une méthode définie dans un objet appelé "Item" depuis un objet appelé "Item2" :

Dans le script de l'item 2:

```
function Idle(dialog,item)
   dialog.GetItem("Item1").myTest()
end
```

· Si vous voulez déclarer des variables globales, le meilleur moyen est de les stocker dans la méthode Init de l'objet Dialog.
Par exemple :

```
function Init(dialog)
   myVar=0
end
```

va déclarer une variable globale appelée "MyVar"

· Le problème devient plus complexe quand vous voulez envoyer une méthode à une fonction. On utilise cela par exemple avec certaines fonctions prédéfinies dans MSLibrary.

Par exemple, la fonction "BrowseFolder" a besoin d'une fonction (call back) qui sera appelée chaque fois qu'un fichier est trouvé.

Le script suivant ne va pas fonctionner :

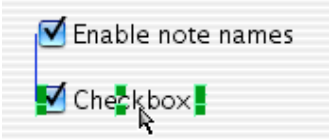
```
function CallBackProcessFile(filename)
   print (filename)
end
function Release(dialog,item)
BrowseFolder(folderName,"*.*",CallBackProcessFile,true)
end
```

Il faut écrire :

```
function CallBackProcessFile(filename)
   print (filename)
end

function Release(dialog,item)
   BrowseFolder(folderName,"*.*",function (fileName) return(GlobalDialog.CallBackProcessFile(fileName)) end,true)
end
```

2.4.9. Alignement des composants

Pour aligner des composants, sélectionnez-en plusieurs et utilisez les icônes au bas de la fenêtre : , , , ,
Vous pouvez aussi déplacer ou changer la taille du composant : une barre bleue verticale montre l'alignement le plus proche.



Si l'option "auto alignement" est active, le composant est automatiquement aligné. Il est possible de fixer la tolérance de l'alignement ou de l'annuler en utilisant l'icône

2.4.10. Création de rapport

Au bas de la fenêtre vous pouvez voir une icône avec un petit cadre sélectionné En cliquant dessus, vous allez générer un rapport sur votre boîte de dialogue qui comprend aussi un diagnostic des erreurs possibles.

2.4.11. Optimisation

Quand vous ajoutez un nouveau composant à une fenêtre de dialogue, les méthodes vides par défaut sont automatiquement créées pour ce composant. Beaucoup d'entre elles ne seront pas utilisées et resteront donc vides. Ceci est particulièrement gênant car les méthodes Idle sont appelées **plusieurs fois par seconde et peuvent donc ralentir l'application**, spécialement quand elles se trouvent dans une fenêtre flottante.
L'icône permet d'effacer toutes les méthodes inutilisées dans votre boîte de dialogue.

2.4.12. Types de composants

Vous trouverez ici une description des différents types de composants que vous pouvez ajouter à une boîte de dialogue.

Quand une boîte de dialogue est ouverte, les méthodes des objets sont appelées quand l'événement attaché se produit. Certaines méthodes sont présentes dans tous les objets (Init, Exit, Idle) et certaines sont spécifiques à certains types de composants (Keydown)

Tous les objets, y compris les composants et les boîtes de dialogue, peuvent avoir leurs propres méthodes privées.

Voici une description des méthodes :

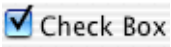
Nom de méthode	Appel de méthode	Objets	Appelé quand
Init	Init(dialog,item)	tous	une seule fois, à l'ouverture de la boîte de dialogue
Exit	Exit(dialog,item)	tous	une seule fois, à la fermeture de la boîte de dialogue
Idle	Idle(dialog,item)	tous	très régulièrement en l'absence d'un événement.
Push	Push(dialog,item)	composants	appui du bouton de la souris dans sa zone.
Release	Release(dialog,item)	composants	le bouton de la souris est relâché.
KeyDown	KeyDown(dialog,item,key)	Texte Statique, Texte Editable, Dialogue	Quand une touche du clavier est enfoncée et que le composant détient la focalisation, la méthode du composant est alors appelée puis la méthode du dialogue est appelée à son tour.
Draw	Draw(dialog,item)	Composant utilisateur, Dialogue	La méthode Draw est appelée pour dessiner le contenu du composant. Dans un dialogue, cette méthode est appelée avant les méthodes Draw des autres objets.
Track	Track(dialog,item)	Barre de défilement, Barre graduée, Potentiomètre	Appelée de façon répétitive pendant que l'utilisateur bouge le curseur ou clique sur la flèche.

Bouton

Aperçu :

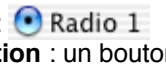
Description : un bouton est un rectangle aux coins arrondis qui contient du texte. Il lance habituellement une action instantanée comme par exemple les opérations définies par une boîte de dialogue ou la prise en compte d'un message d'erreur. Cliquer sur un bouton lance les opérations décrites dans le texte du bouton.
Par exemple, une boîte de dialogue devrait avoir un bouton 'OK' et un bouton 'Annuler'.
Légende : ce composant a une légende que vous pouvez modifier en utilisant DialogItem.Caption (page 508).
Valeur : Ce composant n'a pas de valeur.
Valeurs supplémentaires : Ce composant n'a pas de valeur supplémentaire.
Méthodes : Ce composant possède les méthodes : Init, Exit, Idle, Push, Release.

Case à cocher

Aperçu : 

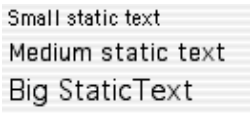
Description : une case à cocher est affichée comme une petit case qui est cochée ou pas, suivie d'une courte chaîne On peut changer son état en cliquant dessus.
Légende : Ce composant a une étiquette que vous pouvez changer en utilisant la valeur DialogItem.Caption (page 50
Valeur : Booléenne, vrai si sélectionné, faux autrement.
Valeurs supplémentaires : Ce composant n'en a pas.
Méthodes : Ce composant possède les méthodes : Init, Exit, Idle, Push, Release.

Bouton Radio

Aperçu : 

Description : un bouton radio est affiché comme un petit cercle suivi d'une chaîne. Il n'est jamais seul dans une boîte de dialogue, mais toujours inclus dans un groupe d'autres boutons radio. Pour définir un groupe, entourez le groupe avec un objet rectangle (FrameRect). Pour activer le bouton, l'utilisateur peut cliquer dessus ou sur la chaîne de texte qui le suit. Un seul bouton peut être sur "on" (allumé) dans chaque groupe à la fois. L'état "allumé" est indiqué par un petit point au centre du cercle. L'allumage d'un bouton éteint tout autre autre bouton du groupe.
Légende : Ce composant a une étiquette que vous pouvez changer en utilisant la valeur DialogItem.Caption (page 50
Valeur : Booléenne, vrai si allumé, faux autrement.
Valeurs supplémentaires : Ce composant n'en a pas.
Methods : Ce composant possède les méthodes : Init, Exit, Idle, Push, Release.

Texte Statique

Aperçu : 

Description : Un texte statique est un texte que l'utilisateur ne peut pas éditer. Trois tailles sont disponibles : petite, moyenne et grande.
Généralement, l'utilisateur n'a pas besoin d'interagir avec, mais si vous le voulez, vous pouvez associer des opération quand l'utilisateur clique dessus. Ce n'est toutefois pas recommandé.
Légende : Ce composant a une légende que vous pouvez changer en utilisant la valeur DialogItem.Caption (page 508
Valeur : Ce composant n'en a pas.
Valeurs supplémentaires : Ce composant n'en a pas.
Méthodes : Ce composant possède les méthodes : Init, Exit, Idle, Push, Release.

Texte Editable

Généralités :

Description : Un texte éditable : un texte que l'utilisateur peut éditer. Pour le différencier d'un texte statique, il est entouré d'un cadre. La longueur maximale du texte est de 240 caractères. Si vous voulez que l'utilisateur édite un texte complet, sans limite de taille, utilisez plutôt un Texte statique.
Légende : Ce composant n'en a pas.
Valeur : Chaîne, le contenu du texte édité.
Valeurs supplémentaires : Ce composant n'en a pas.
Méthodes : Ce composant possède les méthodes : Init, Exit, Idle, Push, Release, KeyDown.

Menu déroulant

Aperçu :

Description : Un bouton de menu déroulant affiche une liste de valeurs que l'utilisateur peut choisir. La partie gauche du composant contient le texte qui montre la sélection en cours. Quand l'utilisateur clique dessus, un menu s'ouvre et l'utilisateur peut changer la valeur en cours. Les différentes valeurs doivent être définies dans la méthode Init de l' objet.
Légende : Ce composant n'en a pas.
Valeur : Nombre, index de la valeur sélectionné allant de 1 à DialogItem.NumberOfExtraValues (page 510).
Valeur supplémentaires : chaînes indexées, titres des entrées du menu. Si la chaîne commence par '(' la ligne est grisée (non sélectionnable).
Méthodes : Ce composant possède les méthodes : Init, Exit, Idle, Push, Release.

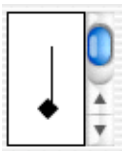
Sélecteur de couleurs

Aperçu :
Description : Un sélecteur de couleurs affiche la couleur sélectionnée. Quand l'utilisateur clique dessus, une palette s'ouvre et il peut sélectionner une couleur. La couleur varie de 0 à 255. Vous pouvez utiliser les fonctions graphiques IndexToRGB et IndexToRGB pour convertir un index en système RVB (rouge, vert bleu) et vice-versa.
Si l'utilisateur fait un Maj+Clic sur ce type de composant, la palette est réduite aux 16 couleurs "standard".
Légende : Ce composant n'en a pas.
Valeur : Nombre, c'est index de la couleur choisie de 0 à 255, 0 étant la couleur par défaut.
Valeurs supplémentaires : Ce composant n'en a pas.
Méthodes : Ce composant possède les méthodes : Init, Exit, Idle, Push, Release.

Sélecteur de Style de Texte

Aperçu :
Description : Un sélecteur de style de texte est affiché sous la forme d'un "A" bleu. Quand l'utilisateur clique dessus, il peut sélectionner la fonte, le style, le graissage et la couleur des caractères.
Légende : Ce composant n'en a pas.
Valeur : TextStyle (page 414), type de texte choisi.
Valeurs supplémentaires : Ce composant n'en a pas.
Méthodes : Ce composant possède les méthodes : Init, Exit, Idle, Push, Release.

Sélecteur de Tête de Note

Aperçu : 

Description : Un objet sélecteur de tête de note affiche une liste de types de têtes de notes. L'utilisateur peut choisir une forme en utilisant la barre de défilement attachée à l'objet.
Légende : Ce composant n'en a pas.
Valeur : Nombre, type de tête choisi (voir les constantes dans MSdefine (page 100)).

SHAPE_NORMAL=0 (normal)
SHAPE_TRIANGLE=1 (triangle)
SHAPE_DIAMOND=2 (losange)
SHAPE_CROSS=3 (croix)
SHAPE_PLUS=4 (+)
SHAPE_SQUARE=5 (carré)
SHAPE_BEAM=6 (tiret)
SHAPE_NONE=7 (aucun)
SHAPE_FLAG=8 (drapeau)
SHAPE_HALF_CIRCLE=9 (demi-cercle)
SHAPE_DROP=10 (goutte)
SHAPE_SAND_GLASS=11 (sablier)
SHAPE_ARC=12 (arc)
SHAPE_EMPTY_DIAMOND=13 (losange plein)
SHAPE_CONE=14 (cône)
SHAPE_CRESCENT=15 (croissant)
SHAPE_LEFT_TRIANGLE=16 (triangle gauche)
SHAPE_BELL=17 (cloche)
SHAPE_CONCAVE_TRIANGLE=18 (triangle concave)
SHAPE_CONVEX_TRIANGLE=19 (triangle convexe)
SHAPE_JOHNSON_TRIANGLE=20 (triangle de Johnson)
SHAPE_JOHNSON_SQUARE=21 (carré de Johnson)
SHAPE_CIRCLE_BOLD_CROSS=22 (croix épaisse dans un cercle)
SHAPE_BOLD_CROSS=23 (croix épaisse)
SHAPE_THIN_CROSS=24 (croix mince)
SHAPE_HALF_CIRCLE_THIN_CROSS=25 (croix mince et demi-cercle)
SHAPE_EMPTY=26 (tête creuse)

Valeurs supplémentaires : Ce composant n'en a pas.
Méthodes : Ce composant possède les méthodes : Init, Exit, Idle, Push, Release.

Sélecteur de durée

Aperçu :

Description : Un objet Sélecteur de Durée affiche la liste des durées d'une note. L'utilisateur peut choisir une durée à l'aide de la barre de défilement attachée à l'objet. Vous pouvez définir la liste des durées de notes en utilisant la méthode Init de cet objet.
Légende : Ce composant n'en a pas.
Valeur : Nombre, numéro de la durée choisie.
Valeurs supplémentaires : Cet objet a des valeurs supplémentaires qui vont de 1 à DialogItem.NumberOfExtraValues (page 510). Chaque valeur supplémentaire est une durée.
Un jeu de durées standard est défini par défaut, mais vous pouvez le modifier.

Par exemple, voici un "Init" personnalisé qui définit un sélecteur de durée à trois valeurs : DURATION_QUARTER, DURATION_HALF,DURATION_WHOLE, (noire, blanche, ronde)la durée DURATION_HALF (blanche) étant la valeur par défaut.

```
-- Init : appelé après l'affichage de la boîte de dialogue
function Init(dialog,item)
item.DeleteExtraValues()
item.SetExtraValue(1,DURATION_QUARTER)
item.SetExtraValue(2,DURATION_HALF)
item.SetExtraValue(3,DURATION_WHOLE)
item.Value=DURATION_HALF
end
```

Méthodes : Ce composant possède les méthodes : Init, Exit, Idle, Push, Release.

Barre de Défilement

Aperçu :

Description : Une barre de défilement s'affiche avec deux flèches et une zone de défilement. Elle peut être verticale ou horizontale.
La position de l'index montre la valeur actuelle. Par défaut, les valeurs peuvent varier de 0 à 100, mais vous pouvez modifier les valeurs minimum et maximum dans la méthode "Init" de l'objet, par exemple.
Légende : Ce composant n'en a pas.
Valeur : nombre, la valeur actuelle de la barre de défilement. de DialogItem.GetExtraValue(1) à DialogItem. GetExtraValue(2).
Valeurs supplémentaires:

Cet objet a deux valeurs supplémentaires :
1 : la valeur Minimum que l'objet peut gérer. (par défaut 0)
2 : la valeur Maximum que l'objet peut gérer. (par défaut 100)

Par exemple, pour fixer les valeurs maximum et minimum à [10,1000] vous pouvez écrire :

```
item.SetExtraValue(1,10)
item.SetExtraValue(2,1000)
```

Pour récupérer la valeur actuelle, utilisez par exemple :

```
currentValue=Item.Value
```

Méthodes : Ce composant possède les méthodes : Init, Exit, Idle, Push, Release, Track.

Barre Graduée

Aperçu :
Description : Une barre graduée peut être verticale ou horizontale. La position de l'index montre la valeur courante. Par défaut, la valeur de la barre de défilement varie de 0 à 100, mais vous pouvez changer les valeurs minimum et maximum dans la méthode Init de cet objet, par exemple.
Légende : Ce composant n'en a pas.
Valeur : Nombre, la valeur corante de l'index de DialogItem.GetExtraValue(1) à DialogItem.GetExtraValue(2).
Valeurs supplémentaires :

Ce composant a deux valeurs supplémentaires :
1 : la valeur Minimum gérée par le composant. (par défaut 0)
2 : la valeur Maximum gérée par le composant. (par défaut 100)

Par exemple, pour fixer les valeurs minimum et maximum à [10,1000] vous pouvez écrire :
item.SetExtraValue(1,10)
item.SetExtraValue(2,1000)

Pour récupérer la valeur courante, utilisez par exemple :
currentValue=Item.Value
Méthodes : Ce composant possède les méthodes : Init, Exit, Idle, Push, Release, Track.

Potentiomètre

Aperçu :

Description : Il permet à l'utilisateur de choisir une valeur à l'aide d'un bouton rond.
Légende : Ce composant a une valeur que vous pouvez changer en utilisant la valeur DialogItem.Caption (page 508).
Valeur : Nombre, la valeur courante du potentiomètre entre DialogItem.GetExtraValue(1) et DialogItem.GetExtraValue 2).
Valeurs supplémentaires :

Ce composant a deux valeurs supplémentaires :
1 : la valeur Minimum gérée par le composant. (par défaut 0)
2 : la valeur Maximum gérée par le composant. (par défaut 100)

Par exemple, pour fixer les valeurs minimum et maximum à [10,1000] vous pouvez écrire :
item.SetExtraValue(1,10)
item.SetExtraValue(2,1000)

Pour récupérer la valeur courante, utilisez par exemple :
currentValue=Item.Value

Méthodes : Ce composant possède les méthodes : Init, Exit, Idle, Push, Release, Track.

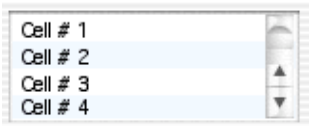
Cadre

Aperçu :

Description : Il s'agit d'un cadre qui peut être purement graphique mais quand il contient des boutons radio, il sert alors à définir un groupe de boutons radio On peut lui associer un nom.
Légende : Ce composant a une valeur que vous pouvez changer en utilisant la valeur DialogItem.Caption (page 508).
Valeur : DialogItem, bouton radio sélectionné dans le groupe (s'il y en a un)
Valeurs supplémentaires : Ce composant n'en a pas.
Méthodes : Ce composant possède les méthodes : Init, Exit, Idle, Push, Release.

Liste

Aperçu :



Description : C'est une liste de cellules. Chaque cellule affiche une chaîne. L'utilisateur peut en sélectionner une. Une barre de défilement est automatiquement associée à la liste pour permettre à l'utilisateur de la faire défiler.
Légende : Ce composant n'en a pas.
Valeur : Nombre, cellule sélectionnée dans la liste de 1 à DialogItem.NumberOfExtraValues (page 510).
Valeurs supplémentaires : chaînes indexées, titres des cellules de la liste.
On peut effacer toutes les valeurs à l'aide de la méthode DialogItem.DeleteExtraValues(...) (page 515).

Exemple:

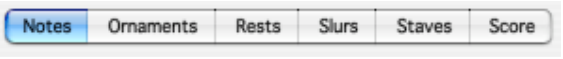
```
-- Init : appelée après la première ouverture du composant dialogue
function Init(dialog,item)
-- Remplissage de la liste avec les chaînes
   for i=1,20 do
      item.SetExtraValue(i,"Choice "..i);
   end
end

-- Release : appelée quand on relâche le bouton souris
function Release(dialog,item)
--- quand un composant est sélectionné, on affiche sa valeur
print(item.GetExtraValue(item.Value))
end
```

Le composant liste admet un mode spécial de sélection. Par défaut, quand l'utilisateur clique sur une cellule, la cellule courante est désélectionnée et la nouvelle cellule est sélectionnée.
Vous pouvez définir, par exemple, un autre mode de sélection en utilisant :
item.SetExtraValue(0,"SelectionMode",mode)
La valeur d'index "0" veut dire : "pour toute la liste"
"SelectionMode" est le mot-clef à utiliser
"mode" détermine comment est faite la sélection. Cette valeur peut être une combinaison de constantes définies dans MSDefine :

- LIST_EXTENDDRAG= 64 Déplacement souris bouton enfoncé=étend la sélection (sans Maj)
- LIST_NODISJOINT = 32 un clic annule la sélection précédente
- LIST_NOEXTEND=16 Maj n'étend pas la sélection
- LIST_NORECT=8 N'étend pas la sélection dans une marquise
- LIST_USESENSE=4 Maj teste l'état de la cellule de départ
- LIST_NONILHILITE=1 Ne pas mettre les cellules vides en surbrillance
- LIST_MULTIPLE_SELECT=LIST_EXTENDDRAG+LIST_USESENSE l'utilisateur peut sélectionner plus d'une seule cellule
- LIST_SINGLE_SELECT=0 mode par défaut

Quand la sélection multiple est active, la méthode GetExtraValue method, appliquée à une cellule donne accès à deux valeurs : la chaîne contenue dans la cellule, l'état de la cellule (valeur booléenne, sélectionnée ou pas)
Par exemple :
str,select=item.GetExtraValue(index)
Méthodes : Ce composant possède les méthodes : Init, Exit, Idle, Push, Release.

Onglets

Aperçu : 

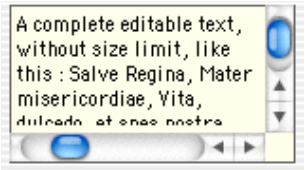
Description : C'est une liste d'onglets sélectionnables. Chaque onglet présente une chaîne,l'utilisateur peut en choisir une.
Légende : Aucune.
Valeur : Nombre, index de l'onglet sélectionné de 1 à DialogItem.NumberOfExtraValues (page 510).
Valeurs supplémentaires : Liste des noms des onglets.
On peut effacer toutes les valeurs à l'aide de la méthode DialogItem.DeleteExtraValues(...) (page 515).

Exemple:

```
-- Init : called just after the dialog has been displayed for the first time
function Init(dialog,item)
-- Filling the list with strings value
    for i=1,4 do
        item.SetExtraValue(i,"Choice "..i);
    end
end

-- Release : called when mouse button is released
function Release(dialog,item)
--- When an element is selected we display its value
print(item.GetExtraValue(item.Value))
end
```

Méthodes : Ce composant possède les méthodes : Init, Exit, Idle, Push, Release.

Memo

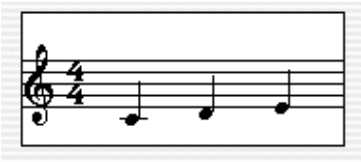
Aperçu : 

Description : C'est un cadre texte complet sans limite de taille. Deux barres de défilement sont automatiquement associées, permettant à l'utilisateur de se déplacer à travers le texte.
Quand une touche du clavier de l'ordinateur est enfoncée et que le composant détient la focalisation, la méthode " KeyDown" de l'objet est appelée.
L'objet peut alors traiter cet événement (touche enfoncée) et retourner la valeur "faux" pour spécifier que le traitement standard de l'événement "touche enfoncée" ne sera pas effectué.
Par exemple, vous pouvez utiliser la méthode "KeyDown" pour appliquer un filtrage des caractères.
Légende : Ce composant n'en a pas.
Valeur : Chaîne, le texte contenu dans le memo.
Valeurs supplémentaires : Une seule valeur supplémentaire : l'objet Text (page 578) qui permet de traiter le contenu Cette valeur est en lecture seule.
Méthodes : Ce composant possède les méthodes : Init, Exit, Idle, Push, Release, KeyDown.

Utilisateur

Description : Ce composant est dessiné par le biais de la méthode "Draw" de l'objet. Quand cette méthode est appelée, la zone de dessin est formatée en fonction de la surface de l'objet (on ne peut pas dessiner en dehors de l' objet) et l'origine est fixée à la surface du composant (les coordonnées 0,0 représentent le coin supérieur gauche de la surface du composant). La surface du composant est effacée avant l'appel de la méthode Draw. Il faut utiliser les méthodes Graph (page 516) pour dessiner dans la zone utilisateur du composant.
Légende : Ce composant n'en a pas.
Valeur : Ce composant n'en a pas.
Valeurs supplémentaires : Ce composant n'en a pas.

Par exemple, nous allons définir les méthodes d'un composant utilisateur qui va afficher une portée avec trois notes :

Aperçu :

EXEMPLE:

```
-- Init : appelée après l'ouverture de la boîte de dialogue
function Init(dialog,item)
-- Création d'une nouvelle portée
item.UserTable["Score"]=NewScore(false)
score=item.UserTable["Score"]
if(score~=nil) then
   item.UserTable["Staff"]=item.UserTable["Score"].AddStaff()
   ---Creating new staff in score
   staff=item.UserTable["Staff"]
   if(staff~=nil) then
      staff.InsertNote(0,DURATION_QUARTER,5*12)
      staff.InsertNote(DURATION_QUARTER,DURATION_QUARTER,5*12+2)
      staff.InsertNote(2*DURATION_QUARTER,DURATION_QUARTER,5*12+4)
      end
   end
end
-- Draw : appelée chaque fois que le composant utilisateur doit être redessiné
function Draw(dialog,item)
   Graph.FrameRect(0,0,item.AreaWidth,item.AreaHeight)
   Graph.DrawStaff(0,0,item.AreaWidth,item.AreaHeight,item.UserTable["Staff"])
end
-- Exit : appelée juste avant la fermeture de la boîte de dialogue
function Exit(dialog,item)
-- Effacement de la partition
   item.UserTable["Score"].Dispose()
end
```

Méthodes : Ce composant possède les méthodes : Init, Exit, Idle, Push, Release, Draw.

2.5. Fenêtre flottante

2.5.1. Généralités

Une fenêtre flottante est une fenêtre d'un type spécial que vous pouvez créer.
Nous avons vu dans les chapitres précédents qu'un script peut ouvrir et fermer une boîte de dialogue définie dans un fichier de l'Interface Composer.
Quand un script ouvre une boîte de dialogue, la boîte de dialogue est modale. C'est à dire que l'utilisateur ne peut accomplir une autre action, comme par exemple ouvrir un fichier de musique ou lancer un autre script, tant que le script n'est pas terminé.
Ces limites ne sont plus vraies avec les fenêtres flottantes :

- Plusieurs fenêtres flottantes peuvent être ouvertes en même temps
- Quand une fenêtre flottante est ouverte, toutes les actions sont permises

A beaucoup d'égards, les fenêtre flottantes sont très semblables aux palettes de l'application.

2.5.2. Qu'est-ce qu'une fenêtre flottante ?

Une fenêtre flottante est une fenêtre avec un titre et des **composants** à l'intérieur. Une fenêtre flottante peut contenir autant de composants que nécessaire.

Vous pouvez considérer qu'une fenêtre flottante est un objet de dialogue autonome de l'Interface Composer qui peut s' utiliser sans script.
Tous les types de composants utilisables dans une boîte de dialogue de l'Interface sont aussi utilisables dans une fenêtre flottante.

Une fenêtre flottante est stockée dans un fichier qui porte l'extension **myf**
Les fenêtres flottantes sont affichées dans le menu *"Scripts"*. Les fenêtres flottantes fournies avec le programme sont marquée avec un , les fenêtres flottantes utilisateur avec un .
Comme les scripts, les fenêtres flottantes peuvent être empaquetées : il vous suffit de ranger le fichier dans un dossier qui porte le même nom.

2.5.3. Comment créer une fenêtre flottante ?

Dans le menu *"Scripts"* sélectionnez "Nouvelle fenêtre flottante". Cela va ouvrir l'éditeur de l'Interface Composer.
Toutes les opération que nous avons vues à propos de l'édition dans l'Interface Composer sont disponibles.
Vous pouvez (et devez) commenter une fenêtre flottante de la même façon que vous commentez un fichier script.

2.5.4. A noter

Comme une fenêtre flottante peut être lancée sans l'aide d'un script externe (souvenez-vous que ce n'est qu'un dialogue autonome de l'Interface Composer), tout ce qui concerne l'initialisation doit se trouver dans les méthode Init de l'objet dialogue.
Par exemple, si votre fenêtre flottante a besoin d'utiliser des constantes ou des fonctions définies dans un fichier prédéfini associé, la méthode "Init" de la fenêtre flottante ressemblera à ceci :

```
function Init(dialog)
Include "MSDefine"
Include "MSSounds"
end
```

Si vous définissez une fonction locale dans une fenêtre flottante, vous pourrez l'appeler comme une nouvelle méthode.
Par exemple, cette fonction :

```
function DisplayContent(value)
 ...
end
```

déclarée dans le corps de l'objet dialogue.
Pour l'utiliser, il vous faudra écrire : "dialog.DisplayContent(1)".

2.5.5. Événements

Certains événements de haute priorité sont automatiquement envoyés à toutes les **fenêtres flottantes** ouvertes. Ceci permet d'accomplir des actions spéciales quand elles ont lieu. Par exemple, un événement de haute priorité est généré quand la musique commence ou s'arrête, ou quand la partition est imprimée.
Des méthodes définies dans le script principal de l'objet dialogue sont alors appelées. Certaines de ces méthodes peuvent retourner une valeur booléenne.
Si le résultat est "faux", l'événement de haute priorité est géré par le script mais pas exécuté par le programme.
Si aucun résultat n'est fourni, c'est la valeur "vrai" qui est retournée.

Nom de méthode	Appel	Appelée quand	Peut interrompre l'événement
OnStartPlay	OnStartPlay(dialog,score)	appelée quand la musique démarre.	Oui
OnStopPlay	OnStopPlay(dialog,score)	appelée quand la musique s'arrête.	Non
OnStartPrint	OnStartPrint(dialog,score)	appelée avant l'impression de la partition.	Oui
OnStopPrint	OnStopPrint(dialog,score)	appelée à la fin de l'impression.	Non
OnInsertSymbol	OnInsertSymbol(dialog,score,symbol)	appelée après qu'un symbole soit inséré.	Non
OnWillInjectSymbol	OnWillInjectSymbol(dialog,score,symbol)	appelée avant qu'un symbole ne soit modifié à l'aide de l'outil injection.	Oui
OnInjectSymbol	OnInjectSymbol(dialog,score,symbol)	appelée après qu'un symbole ai été modifié par l'outil d'injection.	Non
OnInjectScore	OnInjectScore(dialog,score,staff,xPosition,yPosition)	appelée quand un click a lieu en dehors de tout symbole avec l'outil d'injection.	Oui
OnViewsChange	OnViewsChange(dialog,score)	appelée quand des vues ont été éditées.	Non
OnViewChange	OnViewChange(dialog,score)	appelée quand la vue courante change.	Non
OnMouseWheel	OnMouseWheel(dialog,score,amount)	appelée quand la molette de la souris est bougée.	Non
OnScoreChange	OnScoreChange(dialog,score)	appelée lorsque des changements ont été apportés à la partition par l'utilisateur. Si le script maintient des objets de la partition en variables globales, il convient de rafraîchir	Non

Important: Quand vous débogguez une palette flottante, les événements de haute priorité ne sont pas invoqués.

Par exemple, ceci permet de créer une portée temporaire chaque fois que la musique démarre, et de l'effacer quand la musique s'arrête.

```
--- Appelée quand la musique démarre
function OnStartPlay(dialog,score)
   -- Nous créons une portée sur la partition
   staff=score.AddTemporaryStaff()
```

```
        if staff ~= nil then
        -- Nous y insérons des notes
            for i=0,20 do
                staff.InsertNote(DURATION_QUARTER*i,DURATION_QUARTER,5*12+i)
            end
        end
    return true
end
--- Appelée quand la musique s'arrête
function OnStopPlay(dialog,score)
    score.DeleteTemporaryStaves()
end
```

L' événement OnInjectScore est appellé quand l'utilisateur clique en dehors de tout symbole sur une portée. Il recoit le dialogue, la partition, la portée, la position absolue horizontale en pixel (que vous pouvez transformer en position temos via score.AbsolutePixelToWrittenTime), la position verticale relative à la ligne de base de la portée.
Voici un exemple d'un traitement qui ajoute un graphisme (disque noir) à la position de click.

```
function OnInjectScore(dialog,score,staff,xPosition,yPosition)
    staff.BeginRecord(32,32)
    Graph.PenNormal()
    Graph.PaintOval(0,0,32,32)
    staff.EndRecord(xPosition,yPosition,true)
    Application.UpdateScores()
    return false
end
```

2.5.6. Les fenêtres flottantes du point de vue de l'utilisateur

Quand l'utilisateur ouvre une (ou plus) fenêtre flottante, il peut la déplacer sur l'écran, cliquer sur ses composants, etc... La fenêtre flottante répondra aux actions de l'utilisateur.
Quand la fenêtre flottante utilise un objet "éditeur de texte" et qu'il enfonce une touche du clavier, la fenêtre flottante qui se trouve au premier plan recevra l'événement.
Votre fenêtre flottante peut tourner en arrière-plan, en utilisant les méthodes "Idle" des composants ou de l'objet dialogue qui sont appelées régulièrement pas le programme.

2.5.7. A propos des fenêtres flottantes multiples

Nous avons vu que l'utilisateur peut ouvrir autant de fenêtres flottantes qu'il veut, et les conserver ouvertes pendant qu'il travaille. Cela signifie qu'il peut ouvrir plusieurs fois la même fenêtre flottante. Si vous le désirez, vous pouvez empêcher l'utilisateur d'ouvrir deux fois la même fenêtre flottante : ajoutez le texte : "--MonoInstance" quelque part, par exemple au début du texte associé au dialogue.

2.5.8. Palette ou Dialogue

Quand une fenêtre flottante ne contient aucun composant "Éditeur de texte", elle fonctionne comme une palette. La palette est toujours affichée au premier plan, devant toutes les fenêtres du document.
Si vous ne voulez pas que votre fenêtre flottante se comporte comme une palette, insérez le texte "--NoPalette" quelque part, par exemple au début du texte associé au dialogue.

2.5.9. Transparence

Sous Windows les palettes sont transparente, vous pouvez définir le niveau de transparence en insérant :"--TransluscentRatio xxx" où xxx est le niveau de transparence. La valeur 0 signifie opaque.

2.5.10. Optimisation

Les méthodes "Idle" étant appelées pour chaque composant de nombreuses fois par seconde, il est recommandé d' effacer les méthodes inutilisées.

2.6. L'Outil de Recherche

2.6.1. Généralités

Cet outil permet de rechercher et de remplacer des chaînes dans les fenêtres de texte ou dans les fichiers.
Il y a différentes manières d'ouvrir la boîte de dialogue de l'outil de recherche :

- En cliquant sur l'icône dans un fenêtre d'édition de texte.
- En utilisant le raccourci Commande+F.
- En utilisant la commande "Rechercher" dans le menu *Scripts*

2.6.2. Recherche

Vous pouvez faire une recherche qui **respecte la casse** et/ou a une **recherche par mots entiers**.
Vous ne pouvez faire une recherche dans la fenêtre du premier plan uniquement, dans toutes les fenêtres ouvertes, dans tous les fichiers y compris les scripts prédéfinis.
Si vous utilisez la commande "Tout Rechercher", les résultats de la recherche seront affichés dans une fenêtre de texte. En cliquant sur une ligne, la ligne concernée dans le fichier script s'affiche.

2.6.3. Continuer la recherche

Quand vous n'utilisez pas la commande "Tout Rechercher", c'est la première occurrence qui s'affiche. Pour atteindre la suivante, utilisez le raccourci Commande+G ou cliquez sur l'icône .

2.6.4. Remplacer

Pour remplacer une chaîne par une autre, tapez la chaîne recherchée et la chaîne de remplacement dans la boîte de dialogue, puis choisissez "Remplacer" ou "Tout Remplacer"
Quand vous avez défini une chaîne de remplacement et que vous lancez la commande "Trouver", vous pouvez substituer le résultat de la recherche à l'aide de la commande "Remplacer" ou avec le raccourci Commande+L. L' occurrence suivante est automatiquement trouvée. Pour passer l'occurrence suivante sans la remplacer, utilisez le raccourci Commande+G.

2.6.5. Recherche rapide

Pour rechercher rapidement un mot, cliquez dessus en maintenant la touche de commande enfoncée (Ctrl/Pomme) et sélectionnez "Rechercher ceci..." dans le menu.

2.7. Impression de Textes

2.7.1. Généralités

Vous pouvez imprimer soit un script soit ce manuel tout entier. Un aperçu avant impression s'affiche si l'option n'a pas été désactivée dans le paramétrage général.
Utilisez le raccourci Commande+'P' ou la commande correspondante du placard (page 52).

2.7.2. Choix de l'échelle d'impression

Il y a plusieurs manières de changer la taille du texte sur le papier. Vous pouvez éditer les propriétés de l'imprimante (Menu fichier) et adjuster l'échelle d'impression du pilote de votre imprimante.
Vous pouvez augmenter/diminuer la taille du texte en utilisant le raccourci Commande+"+" et Commande+"-" ou en choisissant une nouvelle taille du texte avec le menu placard (page 52) associé à la fenêtre.

2.7.3. Coloration syntaxique

Si l'option Coloration syntaxique est active et que la fenêtre est une fenêtre de script, le texte sera imprimé en couleurs.
Vous pouvez désactiver cette option en utilisant le menu déroulant placard (page 52).

2.8. Fenêtre des Logs (compte-rendu)

La Fenêtre des Logs est une fenêtre texte. Il n'y a qu'une seule fenêtre de logs ouverte à la fois. Chaque fois qu'un script est lancé, la fenêtre des logs est effacée. La fenêtre des logs conserve les 1000 dernières lignes. Quand cette limite est atteinte, la première ligne est effacée. La fenêtre des logs reçoit :

- Tous les résultats de la fonction MyrScript *print*
- Les erreurs générées par un script. En cliquant sur une ligne, on ouvre le fichier script ou le fichier Interface Composer correspondant.

Veuillez noter que vous pouvez définir votre propre fenêtre de texte, en utilisant l'objet TextWindow (page 560).

2.9. Débogage d'un script

2.9.1. Généralités

Le meilleur moyen de localiser les problèmes dans un script et de les résoudre c'est d'utiliser le mode de débogage. En mode débogage, vous pouvez mettre des points d'arrêt pour arrêter l'exécution d'un script, avancer pas à pas, montrer les variables locales ou globales, etc.

2.9.2. L'interface de débogage

En haut de la fenêtre de script, vous pouvez voir quelques icônes qui sont utilisées en mode exécution et débogage. Quand vous exécutez pas à pas un script qui modifie une partition, la fenêtre de la partition est automatiquement mise à jour à l'écran mais vous ne pouvez pas modifier manuellement la fenêtre de la partition. Si vous le faites, ceci arrêtera le script.

Voici une description des diverses commandes :

L'icône Exécuter

Le raccourci clavier est Commande+R.
L'icône lance l'exécution du script. Si un point d'arrêt est rencontré, le script passe en pause et la ligne courante s'affiche.
La fenêtre des variables est automatiquement mise à jour. Elle affiche les valeurs courantes des variables locales et globales que vous avez ajoutées dans la fenêtre des variables.
Vous pouvez interrompre l'exécution d'un script en appuyant sur la touche Esc. La ligne en cours d'exécution s'affiche alors. Vous pouvez ensuite relancer l'exécution.

L'icône Pas à Pas détaillé

Le raccourci clavier est Commande+D.
Cette icône exécute une seule ligne. Si le script appelle une fonction, le mode pas à pas continue avec cette fonction. Si le script inclut un autre fichier script (fonction *Include*) le nouveau script est chargé dans une autre fenêtre.

L'icône Pas à Pas global

Le raccourci clavier est Commande+E.
Cette icône exécute une seule ligne. Si le script appelle une fonction ou s'il inclut un autre fichier script, la fonction ou le script ne sont pas affichés.

L'icône Arrêt

Cette commande arrête l'exécution du script. Le statut courant de MyrScript s'affiche dans la fenêtre des logs. Apr cela, vous ne pouvez pas reprendre l'exécution.

L'icône Effacer tous les points d'arrêt

Cette commande efface tous les points d'arrêts qui ont été posés dans le script.

2.9.3. Points d'arrêt

Les points d'arrêt sont signalés dans la marge gauche du script par un petit point rouge. Vous pouvez poser ou enlever un point d'arrêt en cliquant dans la marge. Si des lignes sont insérées ou effacées dans le script, les points d'arrêts sont remis à la bonne place. Les points d'arrêts sont sauvegardés dans le fichier script.

A Noter : si vous lancez un script depuis le menu, les points d'arrêts ne sont pas pris en compte.

2.9.4. Fenêtre des variables

La fenêtre des variables affiche le type courant et la valeur de toutes les variables locales (les variables qui ont été définies en utilisant le mot clef *local*) et des variables globales que vous avez ajoutées à la fenêtre des variables. Vous pouvez éditer le texte qui s'affiche dans la fenêtre des variables pour changer la valeur d'une variable locale. ou ajouter une nouvelle variable globale (vous ne pouvez pas changer la valeur d'une variable globale).
Pour ajouter rapidement une variable globale à la fenêtre des variables, double-cliquez sur son nom dans la fenêtre du script en maintenant la touche Commande enfoncée.
Pour enlever, ajouter ou effacer toutes les variables globales, faites un clic dans la fenêtre des variables, touche Commande enfoncée.

2.9.5. Performances

A cause des points d'arrêt et autres processus internes, un script tourne de 50% à 75% plus lentement en mode débogage.

2.9.6. Plantages

Toutes les méthodes et les fonctions MyrScript sont protégées contre les entrées de données illégales, mais dans certains cas un script peut faire planter le programme. Donc, avant de partager un script avec d'autres utilisateurs, vérifiez-le en profondeur...
Par exemple, voici un script qui contient des erreurs :
EXEMPLE:

```
local score
-- Prend la partition
score=FrontScore()
-- Première erreur : utiliser une partition sans vérifier qu'elle existe
if(score.Dispose()==true) then
-- Deuxième erreur : utiliser une partition après l'avoir fermée.
print("La partition : "..score.PathName..score.FileName.." est désormais fermée.")
end
```

Le script retournera une erreur Runtime, mais peut planter dans certains cas.

2.10. Le langage Lua

2.10.1. Généralités

Le coeur de MyrScript est le **Lua** : un langage gratuit en open source. Nous avons été stupéfaits de voir la puissance, la rapidité et la stabilité de cette librairie. Chapeau bas à M R. Lerusalimschy, L. H. de Figueiredo et W. Celes de **l'équipe Lua** Quel merveilleux travail !

Vous trouverez dans ce chapitre tous les renseignements essentiels sur la syntaxe du **Langage Lua** .

Si vous souhaitez avoir plus de renseignements sur le **Lua**, y compris une documentation complète, allez sur le site : http://www.lua.org/

Vous pouvez télécharger le manuel complet au format .PDF ici :
"Reference manual of the programming language Lua 4.0", PDF format : http://www.lua.org/ftp/refman.pdf

2.10.2. Types et Balises

Le Lua est un langage typé dynamiquement. Ce qui veut dire que les variables n'ont pas de type, seules les valeurs en ont un.
Par conséquent, il n'y a pas de définition de type dans le langage. Toutes les valeurs comportent leur propre type. En plus d'un type, toutes les valeurs on une balise.
Il y a six types de base dans le **Lua**: nil (nul), nombre, chaîne, fonction, et table.

· Nil est le type de la valeur nulle, dont la principale propriété est d'être différente de toute autre valeur.

·Nombre représente des nombres réels (virgule flottante en double précision), alors que chaîne a le sens habituel. **Lua** a un codage 8-bits, donc les chaînes peuvent contenir n'importe quel caractère d'un octet, y compris les zéros imbriqués ('\0').

·Le type fonction retourne une chaîne qui décrit le type d'une valeur donnée.
Les fonctions sont considérées comme des valeurs de première classe en **Lua**. Cela veut dire que les fonctions peuvent être stockées dans des variables, passées comme arguments à d'autres fonctions et renvoyées comme résultats.

·Le type table gère les tableaux associatifs, c'est-à-dire qui peuvent s'indexer non seulement avec des nombres mais aussi avec n'importe quelle valeur (sauf nul). Par conséquent ce type peut s'utiliser non seulement pour représenter des tableaux ordinaires mais aussi des tables de symboles, des enregistrements, des graphes, des arbres, etc.
Les tables sont le principal mécanisme de structuration des données en **Lua**.
Pour représenter des enregistrements, le **Lua** utilise le nom du champ comme index.
Le langage assure cette représentation en donnant a.name comme marqueur syntaxique de a["name"].
Les tables peuvent aussi comporter des méthodes : parce que les fonctions sont des valeurs de première classe, les champs des tables peuvent contenir des fonctions.
La représentation t:f(x) est la marqueur syntaxique de t.f(t,x), qui appelle la méthode f dans la table t, en passant la table elle-même comme premier paramètre.

A Noter : les tables sont des objets, pas des valeurs. Les variables ne contiennent pas de tables, seulement des références à celles-ci.
L'affectation, le passage de paramètre et les retours manipulent toujours des références aux tables, et n'impliquent aucun type de copie.
De plus, les tables doivent être explicitement créées avant d'être utilisées.
Chacun des types nul, nombre et chaîne a une balise différente.
Toutes les valeurs de ces types ont la même balise prédéfinie.
Pour finir, les valeurs du type table peuvent avoir des balises variables, affectées par le programmeur.
La fonction balise retourne la balise d'une valeur donnée. Les balises utilisateur sont créées avec la fonction newtag.
La fonction settag est utilisée pour changer la balise d'une table.
Les balises sont principalement utilisées pour choisir les méthodes des balises quand certains événements se produisent.
Les méthodes des balises sont le principal mécanisme que **Lua** utilise pour étendre sa sémantique.

2.10.3. Le Langage

Cette section décrit le lexique, la syntaxe et la sémantique du **Lua**.

Conventions Lexicales

Les identifiants en **Lua** peuvent être n'importe quelle chaîne de nombres, de chiffres et de souligné, qui ne commence pas pas un chiffre. Cela coïncide avec la définition des identifiants dans la plupart des langages, si ce n' est que la définition de la lettre dépend de l'environnement local. N'importe quel caractère considéré comme alphabétique dans l'environnement local peut être utilisé dans un identifiant. Les mots suivants sont réservés et ne peuvent donc pas être utilisés comme identifiants :

and	break	do	else	elseif
end	for	function	if	in
local	nil	not	or	repeat
return	then	until	while	

Le Lua is a un langage qui tient compte de la casse : *and* est un mot réservé, mais *And* et *and* (si l'environnement local le permet) sont deux identifiants distincts valides.
Par convention, les identifiants qui commencent par le souligné suivi de majuscules (comme _INPUT) sont réservés pour les variables internes.

```
maVariable=2
mavariable=3
print(maVariable," différent de ",mavariable)
```

La chaîne suivante montre d'autres jetons-clefs:

~=	<=	>=	<	>	==	=	+	-	*	/
(	)	{	}	[	]	;	,	.	..	...

Les chaînes littérales peuvent être délimitées par des guillemets simples ou doubles (ouverts et fermés !) et peuvent contenir des séquences d'échappement comme en C `\a' (cloche), `\b' (retour arrière),`\f' (saut de page), `\n' (saut de ligne), `\r' (retour de chariot), `\t' (tabulation horizontale), `\v' (tabulation verticale), `\\' (antislash), `\"' (guillemets), `\'' (apostrophe), and `\nouvelle ligne' (c'est-à-dire un antislash suivi d'une nouvelle ligne, ce qui insère une nouvelle ligne dans la chaîne). Un caractère dans une chaîne peut aussi être spécifié par sa valeur numérique, dans la séquence d' échappement `\ddd', où ddd est une séquence de 3 chiffres décimaux maximum. Les chaînes en **Lua** peuvent contenir n'importe quelle valeur sur 8 bits, y compris des zéros imbriqués qui sont notés `\000'.
Les chaînes littérales peuvent aussi être délimitées par des crochets (justifiés !) [[...]]. Les écritures entre crochets peuvent occuper plusieurs lignes, peuvent contenir des paires enchâssées [[...]], et n'interprètent pas les séquences d' échappement. Cet emploi est particulièrement utile pour écrire des chaînes qui contiennent des morceaux de programme ou d'autres chaînes citées. Par exemple, dans un système qui utilise l'ASCII, les trois écritures sont équivalentes :

1) "alo\n123\""
2) '\97lo\10\04923"'
3) [[alo
   123"]]

Les commentaires commencent n'importe où à l'extérieur d'une chaîne qui a un double tiret (--) et vont jusqu'au bout de la ligne.

```
-- Ceci est un commentaire
-- Ce programme ne fait rien...
```

Les constantes numériques peuvent s'écrire avec la partie décimale facultative et un exposant décimal facultatif. Voici des exemples de constantes numériques valides :

3 3.0 3.1416 314.16e-2 0.31416E1

Conversion

Lua gère les conversions automatiques entre les valeurs en cours d'utilisation. Toute opération arithmétique appliquée à une chaîne essaie de convertir cette chaîne en nombre, en suivant les règles habituelles. De la même façon, quand un nombre est utilisé alors qu'une chaîne est attendue, ce nombre est converti en chaîne, dans le format adéquat. Le format est choisi de façon à reproduire exactement le nombre quand il est converti en chaîne puis converti à nouveau en nombre. En conséquence, la conversion ne produit pas nécessairement un texte qui a belle apparence dans le cas de certains nombres. Pour conserver le contrôle complet sur la manière dont les nombres sont convertis en chaînes, utilisez la fonction *format*.

```
myString="100"
myString=myString+1
-- va afficher 101
print(myString)
```

Ajustement

Les fonctions en **Lua** peuvent renvoyer plusieurs valeurs. Comme il n'y a aucune déclaration de type, quand une fonction est appelée, le système ne peut pas savoir combien de valeurs la fonction va renvoyer, ou de combien de paramètres elle a besoin. Donc, il faut parfois ajuster une liste de valeurs, en cours d'exécution, à une longueur donnée. S'il y a plus de valeurs que nécessaire, les valeurs en trop sont abandonnées. S'il y a moins de valeurs que nécessaire, la liste est alors complétée avec des valeurs nulles. Cet ajustement a lieu lors de nombreuses assignations et dans les appels de fonctions.

```
-- la fonction myTest retourne deux valeurs
function myTest()
return 1,2
end
local a
local b
a,b=myTest()
print("a= "..a.."", b= "..b)
```

Déclarations

Lua fournit un jeu presque classique de déclarations, semblables à celles du Pascal ou du C. Ces commandes classiques comprennent l'assignation, les structures de contrôle et les appels de procédures. Des commandes non-classiques comprennent les constructeurs de tables et les déclarations de variables locales.

Blocs

Un bloc est une liste de déclarations ; syntaxiquement un bloc est l'équivalent d'un chunk.

bloc ::= chunk

Un bloc peut être délimité de manière explicite.:

stat ::= do block end

Les blocs explicites sont utiles pour contrôler l'ensemble des variables locales. Les blocs explicites sont parfois utilisés pour ajouter un retour ou une déclaration de sortie (break) au milieu d'un autre bloc.

Assignation

Lua permet l'assignation multiple. Par conséquent la syntaxe pour l'assignation. Par conséquent, la syntaxe pour l' assignation définit une liste de variables à gauche et une liste d'expressions à droite. Dans chaque liste, les éléments sont séparés par des virgules :

```
stat ::= varlist1 `=' explist1
varlist1 ::= var {`,' var}
```

Cette déclaration évalue d'abord toutes les valeurs de droite et les éventuels indices à gauche, puis fait les assignations. Donc le code :

```
x = 3
x, a[x] = 4, 20
```

affected a[3] à 20, mais n'affecte pas a[4] parce que le x dans a[x] est évalué avant qu'il soit assigné à 4.
L'assignation multiple peut s'utiliser pour échanger deux valeurs comme dans :

```
x, y = y, x
```

Les deux listes d'une assignation multiple peuvent avoir des longueurs différentes. Avant l'assignation, la liste de valeurs est ajustée à la longueur de la liste des variables.
Un nom peut définir une variable globale, une variable locale ou un paramètre écriture :

```
var ::= nom
```

```
a,b,c=1,2,3
print("a= "..a.., b= "..b..",c= "..c)
```

Les crochets sont utilisés pour indexer une table :

```
var ::= varorfunc `[' exp1 `]'
varorfunc ::= var | functioncall
```

varorfunc devrait donner une valeur de table, de laquelle le champ indexé par la valeur de l'expression exp1 tire la valeur assignée.
La syntaxe var.NAME n'est qu'un équivalent syntaxique de var["NAME"]:

```
var ::= varorfunc `.' nom
```

A l'aide de la méthode des balises, on peut changer la signification des assignations et évaluations des variables globales et des variables indexées. En fait, une assignation x = val, où x est une variable globale, est équivalente à un appel à setglobal("x", val) et une assignation t[x] = val est équivalente à settable_event(t,x,val).

Structures de Contrôle

Les structures de contrôle if, while, and repeat (si, tant que, répéter) ont le sens et la syntaxe habituels.

```
stat ::= while exp1 do block end
```

```
local a
a=0
while a<20 do
   print(a)
   a=a+1
end
```

```
stat ::= repeat block until exp1
```

```
local a
a=0
repeat
   print(a)
   a=a+1
until a>=20
```

```
stat ::= if exp1 then block {elseif exp1 then block} [else block] end
```

La condition de l'expression exp1 d'une structure de contrôle peut renvoyer n'importe quelle valeur. Toutes les valeurs différentes de nil (nul) sont considérées comme vraies. Seule la valeur nulle (nil) est considérée comme fausse.

```
local a
a=0
while a<20 do
   if a<10 then
      print(a)
   end
   a=a+1
end
```

La déclaration de retour (return) est utilisée pour renvoyer des valeurs d'une fonction ou d'un chunk (bloc). Comme les fonctions ou les chunks peuvent renvoyer plus d'une seule valeur, la syntaxe de la déclaration de retour est :

stat ::= return [explist1]

La déclaration d'interruption (break) peut s'utiliser pour mettre fin à l'exécution d'une boucle, en sautant la déclaration qui suit la boucle :

stat ::= break
Une interruption met fin à la boucle la plus imbriquée (while, repeat ou for)
Pour des raisons syntaxiques, les déclarations de retour et d'interruption ne peuvent s'écrire qu'à la toute fin d'un bloc. S' il est vraiment nécessaire de faire un retour ou une interruption au milieu d'un bloc, un bloc explicite imbriqué peut être utilisé, comme dans l'idiome `do return end', parce qu'alors return est bien la dernière déclaration du bloc imbriqué.

Déclaration For

La déclaration For a deux formes, une pour les nombres et une pour les tables. Le nombre pour les boucles a la syntaxe suivante :

stat ::= for name `=' exp1 `,' exp1 [`,' exp1] do block end

```
local a
for a=0,20 do
   print(a)
end
```

Une déclaration For comme :

```
   for var = e1 ,e2, e3 do block end
```
est équivalente au code :

```
   do
    local var, _limit, _step = tonumber(e1), tonumber(e2), tonumber(e3)
    if not (var and _limit and _step) then error() end
    while (_step>0 and var<=_limit) or (_step<=0 and var>=_limit) do
     block
     var = var+_step
    end
   end
```
A noter :
* sep=0pt
* _limit and _step sont des variables invisibles.. Les noms sont mis à des fins de clarté.
* Le comportement est indéfini si vous assignez var à l'intérieur du bloc.
* Si la troisième expression (le pas) est absente, un pas de -1 est utilisé.
* La limite et le pas sont tous deux évalués une seule fois, avant l'exécution de la boucle.
* La variable var est locale pour cette déclaration, vous ne pouvez pas utiliser sa valeur en dehors de la boucle.
* Vous pouvez utiliser break pour sortir de la boucle. Si vous avez besoin de la valeur de l'index, assignez-la à une autre variable avant l'interruption.

La table d'une déclaration parcourt toutes les paires (index, valeur) d'une table donnée. Elle a la syntaxe suivante :

```
   stat ::= for name `,' name in exp1 do block end
```

Une déclaration de boucle for comme

```
   for index, value in exp do block end
```
est équivalente au code:

```
   do
    local _t = exp
    local index, value = next(t, nil)
    while index do
     block
     index, value = next(t, index)
    end
   end
```
A noter:
* sep=0pt
* _t est une variable invisible. Le nom n'est utilisé qu'à des fins de clarté.
* Le comportement n'est pas défini si vous assignez une valeur à l'index au milieu du bloc.
* Le comportement n'est pas défini si vous changez la table _t pendant le balayage.
* Les variables index et value sont locales dans la déclaration ; vous ne pouvez pas utiliser leurs valeurs en dehors de la boucle.
* Vous pouvez utiliser break pour sortir de la boucle. Si vous avez besoin de la valeur de l'index, assignez-la à une autre variable avant l'interruption.
* L'ordre dans lequel les variables sont balayées n'est pas défini, même pour des indices numériques. Si vous voulez balayer les indices dans un ordre numérique, utilisez un for numérique.

Déclarations d'appels de fonctions

A cause d'effets secondaires possibles, les appels de fonctions peuvent s'exécuter en tant que déclarations :

```
   stat ::= functioncall
```
Dans ce cas, toutes les valeurs renvoyées sont abandonnées.

Déclarations locales

Les variables locales peuvent être déclarées n'importe où à l'intérieur d'un bloc. La déclaration peut comporter une assignation initiale :

```
stat ::= local declist [init]
declist ::= name {`,' name}
init ::= `=' explist1
```

Si elle est présente, l'assignation initiale a la même sémantique qu'une assignation multiple. Sinon, toutes les variables sont initialisées à la valeur nulle.
Un chunk est aussi un bloc, et donc les variables locales peuvent être déclarées en dehors de n'importe quel bloc explicite.
La visibilité des variables locales commence après la déclaration et dure jusqu'à la fin du bloc. Par conséquent, le code local print=print crée une variable locale nommée print dont la valeur initiale est celle d'une variable de même nom.

```
for index=0,20 do
-- Déclare 'var' comme locale
   local var
   var=index*2
end
-- imprimera 'nil' car var est locale dans la boucle
-- et n'a pas de valeur en dehors
print(var)
```

Expressions

·Expressions basiques
Les expressions basiques en **Lua** sont

```
exp ::= `(' exp `)'
exp ::= nil
exp ::= number
exp ::= literal
exp ::= var
exp ::= upvalue
exp ::= function
exp ::= functioncall
exp ::= tableconstructor
```

L'accès à une variable globale x est équivalent à un appel getglobal("x") et un accès à une variable indexée t[x] est équivalent à un appel gettable_event(t,x).
La variable non-terminale exp1 est utilisée pour indiquer que les valeurs renvoyées par une expression doivent être ajustées à une seule et unique valeur.:

```
exp1 ::= exp
```

·Opérateurs Arithmétiques
Lua gère les opérateurs arithmétiques habituels: les opérateurs binaires, + (addition), - (soustraction), * (multiplication), / (division), and ^ (exponentiation); et unaire - (négation). Si les opérandes sont des nombres, ou des chaînes convertibles en nombres, alors toutes les opérations excepté l'exponentiation ont leur signification habituelle. Sinon, une méthode balisée habituelle est appelée. Une exponentiation appelle toujours une méthode de balise. La librairie mathématique standard redéfinit cette méthode pour les nombres, en donnant à l'exponentiation la signification attendue

- Opérateurs relationnels
Les opérateurs relationnels en **Lua** sont

```
== ~= < > <= >=
```

Ces opérateurs renvoient la valeur nulle (nil) si faux et une valeur non nulle si vrai.
Égalité (==) compare d'abord les balises de l'opérande. Si elles sont différentes, le résultat est nul. Sinon, les valeurs sont comparées. Les nombres et les chaînes sont comparés de la manière habituelle. Les tables et les fonctions sont comparées par référence, c'est-à-dire que deux tables sont considérées comme égales seulement si elle sont la même table. (* (page 99)). L'opérateur ~= est exactement la négation de l'égalité (==).
Les règles de conversion ne s'appliquent pas aux comparaisons d'égalité. Par conséquent, "0"==0 est évalué à faux, et t[0] and t["0"] désignent des entrées différentes dans une table.
L'ordre des opérateurs fonctionne de la manière suivante : si les deux arguments sont des nombres, ils sont alors comparés comme tels. Sinon, si les deux arguments sont des chaînes, leurs valeurs sont alors comparées en utilisant l' ordre alphabétique. Sinon, la méthode balisée ``lt'' est appelée.

·Opérateurs logiques
Les opérateurs logiques en **Lua** sont

```
and or not
```

Comme les structures de contrôle, tous les opérateurs logiques considèrent la valeur nulle (nil) comme faux et tout le reste comme vrai.
L'opérateur de conjonction and retourne une valeur nulle si le premier argument est nul ; sinon, il retourne le second argument. L'opérateur de disjonction or retourne le premier argument s'il est différent de la valeur nulle ; sinon, il retourne le second argument. and et or utilisent une évaluation courte,c'est-à-dire que le deuxième opérande est évalué uniquement si nécessaire.
Il y a deux idiomes **Lua** qui utilisent des opérateurs logiques. Le premier idiome est

```
x = x or v
```

qui est équivalent à

```
if x == nil then x = v end
```

Cet idiome met x à une valeur par défaut v quand x n'est pas défini.
Le second idiome est

```
x = a and b or c
```

qu'il faudrait lire comme x = (a and b) or c. Cet idiome est équivalent à

```
if a then x = b else x = c end
```

à condition que b ne soit pas nul.

·Concaténation

L'opérateur de concaténation en **Lua** est signalé par deux points (`..'). Si les deux opérandes sont des chaînes ou des nombres, ils sont alors convertis en chaînes selon les règles. Sinon la méthode balisée ``concat'' est appelée.

·Ordre de priorité
L'ordre de priorité en **Lua** suit la table ci-dessous, de la priorité la plus basse à la priorité la plus élevée.:

```
and  or
<  >  <= >= ~= ==
..
+  -
*  /
not  - (unary)
^
```

Tous les opérateurs binaires sont associatifs à gauche, sauf ^ (exponentiation), qui est associative à droite. Le pré-compilateur peut changer l'ordre d'évaluation des opérateurs associatifs (comme_.. or_+), dans la mesure ou cette optimisation ne change pas le résultat normal. Cependant, cette optimisation peut changer certains résultats si vous définissez des méthodes balisées non-associatives pour ces opérateurs.

·Constructeurs de tables
Les constructeurs de tables sont des expressions qui créent de tables ; chaque fois qu'un constructeur est évalué, une nouvelle table est céée.Les constructeurs peuvent être utilisés pour créer des tables vides ou pour créer une table et initialiser certains de ses champs. La syntaxe générale pour les constructeurs est :

```
tableconstructor ::= `{' fieldlist `}'
fieldlist ::= lfieldlist | ffieldlist | lfieldlist `;' ffieldlist | ffieldlist `;' lfieldlist
lfieldlist ::= [lfieldlist1]
ffieldlist ::= [ffieldlist1]
```

L'écriture lfieldlist1 est utilisée pour initialiser la table:

```
lfieldlist1 ::= exp {`,' exp} [`,']
```

Les expressions de la liste sont assignées à des indices numériques consécutifs, en commençant à 1. Par exemple,

```
a = {"v1", "v2", 34}
```

est équivalent à

```
do
 local temp = {}
 temp[1] = "v1"
 temp[2] = "v2"
 temp[3] = 34
 a = temp
end
```

L'écriture ffieldlist1 initialise d'autres champs dans une table :

```
ffieldlist1 ::= ffield {`,' ffield} [`,']
ffield ::= `[' exp `]' `=' exp | name `=' exp
```

Par exemple,

```
a = {[f(k)] = g(y), x = 1, y = 3, [0] = b+c}
```

est équivalent à

```
do
 local temp = {}
 temp[f(k)] = g(y)
 temp.x = 1  -- or temp["x"] = 1
 temp.y = 3  -- or temp["y"] = 3
 temp[0] = b+c
 a = temp
end
```

Une expression comme {x = 1, y = 4} est en fait un équivalent syntaxique de {["x"] = 1, ["y"] = 4}.
Les deux écritures peuvent avoir une éventuelle virgule à la suite et peuvent être utilisés dans le même constructeur séparés par un point-virgule. Par exemple, toutes les écritures ci-dessous sont correctes :

```
x = {;}
x = {"a", "b",}
x = {type="list"; "a", "b"}
x = {f(0), f(1), f(2),; n=3,}
```

```
-- pour utiliser ce script :
-- lancez-le et examinez le résultat dans la fenêtre des logs,
-- et comparez-le au code source correspondant
-- pour comprendre comment on peut utiliser les tables.

-- Cette fonction affiche le contenu entier d'une table
function tableDump(table,indentStr)
local index,val
    -- Parcourt les éléments de la table
    for index,val in table do
        -- appel récursif si l'élement de la table est lui-même une table
        if type(val)=="table" then
            print(indentStr.."["..index.."] :")
            tableDump(table[index],indentStr.."  ")
        -- sinon, affiche dans la fenêtre des logs
        else print(indentStr.."["..index.."] :",val)
        end
    end
end

-- 1- Table Simple (index numériques)
c={} -- c est une table vide
c[1]=1 -- remplit avec les valeurs
c[2]=4
c[3]=9
c[4]=16
c[5]=c[3]+c[4]
print("Afficher la table (1)")
tableDump(c,"")

-- 2- Table Simple table (index sur des mots)
c={} -- c est une table vide
c["One"]=1 -- remplit avec les valeurs
c["Two"]=4
c.Three=9 -- Equivalent à c["Three"]=9
print("\rAffichage de la table (2)")
tableDump(c,"")

-- 3- Table de table
c={} -- c est une table vide
c[1]={} -- c[1] est aussi une table
c[1].double=2
c[1].triple=3
c[2]={} -- c[2] est aussi une table
c[2].double=4
c[2].triple=6
print("\rAffichage de la table (3)")
tableDump(c,"")

-- 4- table pré-initialisée
c={"Bonjour","Monde",34}
print("\rAffichage de la table (4)")
tableDump(c,"")

-- 5- table pré-initialisée
c={First="Bonjour",Second="Monde",[3]=34}
print("\rAffichage de la table (5)")
tableDump(c,"")

-- 6- table pré-initialisée complexe
c={"Un","Deux",{cell=1,line=2},"Quatre"}
print("\rAffichage de la table (6)")
tableDump(c,"")
```

·**Appels de fonctions**
Un appel de fonction en **Lua** a la syntaxe suivante:

    functioncall ::= varorfunc args
D'abord, varorfunc est évaluée. Si sa valeur est du type fonction, la fonction est alors appelée avec les arguments donnés. Sinon, la méthode balisée ``function'' est appelée, avec comme premier paramètre la valeur de varorfunc, suivi des arguments d'origine de l'appel.

L'écriture

```
functioncall ::= varorfunc `:' name args
```

peut s'utiliser pour appeler ``methods''. Un appel call v:name(...) est un équivalent syntaxique de v.name(v, ...), si ce n' est que v est évaluée une seule fois.
Les arguments ont la syntaxe suivante :

```
args ::= `(' [explist1] `)'
args ::= tableconstructor
args ::= literal
explist1 ::= {exp1 `,'} exp
```

Toutes les expressions et arguments sont évalués avant l'appel. Un appel du type f{...} est un équivalent syntaxique de f({...}), c'est-à-dire que la liste d'arguments est une seule table nouvelle. Un appel du type f'...' (ou f"..." ou f[[...]]) est un équivalent syntaxique de f('...'), c'est-à-dire que la liste des arguments est une seule chaîne littérale.
Parce qu'une fonction peut renvoyer n'importe quel nombre de résultats, le nombre de résultats doit être ajusté avant qu' ils ne soient utilisés. Si la fonction est appelée en tant que déclaration, sa liste de retour est ajustée à 0, en abandonnant toutes les valeurs renvoyées. Si la fonction est appelée à un endroit qui nécessite une seule valeur, (syntaxiquement marquée par le exp1 non-terminal), sa liste de retour est alors ajustée à 1, en excluant donc toutes les valeurs sauf la première. Si la fonction est appelée à un endroit qui peut contenir beaucoup de valeurs (syntaxiquement marquée par le exp non-terminal), alors aucun ajustement n'est fait. Les seuls emplacements qui peuvent contenir beaucoup de valeurs sont la dernière (ou la seule) expression dans une assignation, une liste d'arguments ou dans une déclaration de retour. Voici quelques exemples :

```
f()        -- ajustée à 0 résultat
g(f(), x)     -- f() est ajustée à 1 résultat
g(x, f())     -- g prend x plus toutes les valeurs retournées par f()
a,b,c = f(), x   -- f() est ajustée à 1 résultat(et c prend nil)
a,b,c = x, f()   -- f() est ajustée à 2
a,b,c = f()    -- f() est ajustée à 3
return f()     -- retourne toutes les valeurs renvoyées par f()
return x,y,f()   -- retourne a, b, et toutes les valeurs renvoyées par f()
```

·Définitions de Fonctions

La syntaxe d'une définition de fonction est

```
function ::= function `(' [parlist1] `)' block end
stat ::= function funcname `(' [parlist1] `)' block end
funcname ::= name | name `.' name | name `:' name
```

La déclaration

```
function f () ... end
```

est un équivalent syntaxique de

```
f = function () ... end
```

et la déclaration

```
function v.f () ... end
```

est un équivalent syntaxique de

```
v.f = function () ... end
```

La définition d'une fonction est une expression exécutable dont la valeur a le type fonction. Quand **Lua** pré-compile un bloc, tous les corps des fonctions sont pré-compilés eux aussi. Ensuite, chaque fois que **Lua** exécute la définition de fonction, les valeurs supérieures sont fixées, et la fonction instanciée (ou fermée). Cette instanciation de fonction (ou sa fermeture) est la valeur finale de l'expression. Différentes instances peuvent avoir des valeurs supérieures différentes.
Les paramètres agissent comme des variables locales, initialisées avec les valeurs des argument :

```
parlist1 ::= `...'
parlist1 ::= name {`,' name} [`,' `...']
```

Quand une fonction est appelée, la liste d'arguments est ajustée à la longueur de la liste de paramètres, à moins que la fonction ne soit une fonction vararg, ce qui est indiqué par trois points (`...') à la fin de sa liste de paramètres. Une fonction vararg f n'ajuste pas sa liste d'arguments ; au lieu de cela, elle stocke tous les arguments supplémentaires dans un paramètre implicite, appelé arg. La valeur de arg est une table, avec un champ field_n dont la valeur est le nombre d' arguments supplémentaires et les arguments sont aux positions 1,_2,_...,_n.
Par exemple, considérerons les définitions suivantes :

```
function f(a, b) end
function g(a, b, ...) end
function r() return 1,2,3 end
```

Nous aurons alors la distribution suivante des arguments aux paramètres :

```
APPEL     PARAMETRES

f(3)       a=3, b=nil
f(3, 4)     a=3, b=4
f(3, 4, 5)    a=3, b=4
f(r(), 10)    a=1, b=10
f(r())      a=1, b=2

g(3)       a=3, b=nil, arg={n=0}
g(3, 4)     a=3, b=4, arg={n=0}
g(3, 4, 5, 8)  a=3, b=4, arg={5, 8; n=2}
g(5, r())    a=5, b=1, arg={2, 3; n=2}
```

Les résultats sont renvoyés en utilisant la déclaration de retour. Si le contrôle atteint la fin d'une fonction sans rencontrer de déclaration de retour, la fonction ne retourne alors aucun résultat.
La syntaxe

```
funcname ::= name `:' name
```

est utilisée pour définir des méthodes, c'est-à-dire, des fonctions qui ont un paramètre supplémentaire explicite propre.
La déclaration

```
function v:f (...) ... end
```

est un équivalent syntaxique de

```
v.f = function (self, ...) ... end
```

Notez que cette fonction a un un paramètre formel supplémentaire appelé self.

Visibilité and valeurs

Le corps d'une fonction peut faire référence à ses propres valeurs locales (qui incluent ses paramètres) et à des variables globales, tant qu'elle ne sont pas masquées par des variables locales qui portent le même nom à l'intérieur des fonctions qui les englobent. Une fonction ne peut pas avoir accès à une variable locale d'une fonction qui l'englobe, puisque de telles variables n'existent plus quand la fonction est appelée. Toutefois, une fonction peut avoir accès à la valeur d'une variable locale d'une fonction qui l'englobe en utilisant les valeurs supérieures, dont la syntaxe est

```
upvalue ::= `%' name
```

Une valeur supérieure est en quelque sorte similaire à une expression de variable mais dont la valeur est gelée quand la fonction à l'intérieur de laquelle elle se trouve est instanciée. Le nom utilisé dans une valeur supérieure peut être le nom de n'importe quelle variable à l'endroit où la fonction est définie, c'est-à-dire des variables locales ou globales qui appartiennent à la fonction qui l'englobe immédiatement. Notez que lorsqu'une valeur supérieure est une table, seule la référence à cette table (qui est la valeur de l valeur supérieure) est gelée ; on peut changer le contenu de la table comme on veut. L'utilisation des valeurs supérieures est une technique qui permet d'avoir un état modifiable mais privé attaché aux fonctions.
Voici quelques exemples ::

```
a,b,c = 1,2,3  -- variables globales
local d
function f (x)
 local b = {} -- x et b sont locales pour f; b masque b (globale)
 local g = function (a)
  local y   -- a et y sont locales pour g
  p = a    -- OK, accès à la la locale `a'
  p = c    -- OK, accès à la globale `c'
  p = b    -- ERREURR: ne peut pas accéder à une variable non visible
  p = %b   -- OK, accès à la valeur gelée de `b' (locale pour `f')
  %b = 3   -- ERREUR: ne peut changer une valeur supérieure
  %b.x = 3  -- OK, change le contenu de la table
  p = %c   -- OK, accès à une valeur gelée de la globale `c'
  p = %y   -- ERREUR: `y' n'est pas visible là où `g' est définie
  p = %d   -- ERREUR: `d' n'est pas visible là où `g' est définie
 end      -- g
end       -- f
```

2.10.4. Bibliothèques Standard

Les bibliothèque standard offrent des fonctions utiles qui sont implémentées directement dans l'API (interface programmeur)

- bibliothèque de base;
- manipulation de chaînes;
- fonctions mathématiques (sin, log, etc);

Fonctions de base

La bibliothèque standard fournit quelques fonctions centrales au **Lua**. En conséquence, si vous n'incluez pas cette bibliothèque dans votre application, vous devriez vérifier soigneusement si vous aurez besoin de fournir une implémentation de rechange pour certains usages. (Par exemple, sans la fonction _ERRORMESSAGE, **Lua** est incapable d'afficher des messages d'erreur.)

_**ALERT** (message)
Imprime son seul argument chaîne sur stderr. Tous les messages d'erreur dans **Lua** sont imprimés par l'intermédiaire de la fonction stockée dans la variable globale _ALERT. Par conséquent, un programme peut assigner une autre fonction à cette variable pour changer la manière dont les messages sont affichés.(par exemple, pour les systèmes qui n'ont pas de stderr).

```
-- New __ALERT function, pour afficher un message dans une boîte d'alerte
function _ALERT(message)
message=gsub(message,"\n","\r") -- a besoin de \r pour finir les lignes
Message("Erreur Interne du Script: ",message)
end
-- exemple d'erreur
a=10/0
```

assertion (v [, message])
Génère une erreur ``l'assertion a échoué!'' quand son argument v est nul. Cette fonction est équivalente à la fonction **Lua** suivante:

```
function assert (v, m)
 if not v then
  m = m or ""
  error("l'assertion a échoué! " .. m)
 end
end
```

call (func, arg [, mode [, errhandler]])
Appelle la fonction func avec les arguments donnés par la table arg. L'appel est équivalent à

```
func(arg[1], arg[2], ..., arg[n])
```

où n est le résultat de getn(arg). Tous les résultats de func sont simplement renvoyés par l'appel.
Par défaut, si une erreur survient pendant l'appel à func, l'erreur se propage. Si le mode chaîne contient "x", l'appel est alors protégé. Dans ce mode, un appel à une fonction ne propage pas une erreur, peu importe ce qui a pu se passer pendant l'appel. Au lieu de cela, elle retourne nul pour signaler l'erreur (en plus d'appeler le gestionnaire d'erreur approprié).
Si errhandler est fourni, la fonction d'erreur _ERRORMESSAGE est temporairement dirigée vers errhandler, pendant que la fonction s'exécute. En particulier si errhandler est nulle, aucun message d'erreur ne sera généré pendant l'exécution de la fonction appelée.

collectgarbage ([limit])
Fixe le seuil de la garbage collection (nettoyage mémoire) à une limite donnée (en Koctes), et vérifie avec le compteur d'octets. Si le seuil est inférieur au compteur d'octets, alors **Lua** lance immédiatement le nettoyage. Si la limite est absente, elle est mise à zéro par défaut.(forçant ainsi un cycle de nettoyage).

copytagmethods (tagto, tagfrom)
CCopie toutes les méthodes balisées d'une balise à l'autre ; renvoie tagto.

dostring (string [, chunkname])
Execute une chaîne donnée comme un bloc en **Lua**.S'il y a une erreur pendant l'exécution de la chaîne, alors dostring retourne nil. Sinon, elle retourne les valeurs renvoyées par le bloc ou une valeur non nulle si le bloc ne retourne pas de valeurs. Le paramètre facultatif chunkname est le nom du bloc, utilisé dans les messages d'erreur et les renseignements de débogage.

error (message)
Appelle le gestionnaire d'erreurs puis termine la dernière fonction protégée appelée (en _C: lua_dofile, lua_dostring, lua_dobuffer, ou lua_callfunction; en **Lua**: dostring, ou appel en mode protégé). Si le message est nu, le gestionnaire d'erreurs n'est pas appelé. La fonction erreur n'a jamais de retour.

foreach (table, func)
Execute la fonction donnée func sur tous les éléments de la table. Pour chaque élément, la fonction est appelée avec l'index et la valeur respectifs comme arguments. Si la fonction retourne n'importe quelle valeur non nulle, la boucle est interrompue et cette valeur est renvoyée comme valeur finale de foreach. Cette fonction pourrait être définie en **Lua**:

```
function foreach (t, f)
 for i, v in t do
  local res = f(i, v)
  if res then return res end
 end
end
```

Le comportement de foreach est indéfini si vous changez la table t pendant son balayage.

foreachi (table, func)
Execute la fonction donnée func sur les indices numériques de la table. Pour chaque index, la fonction est appelée avec l'index et la valeur respectifs comme arguments. Les indices sont parcourus en ordre séquentiel, de 1 à n, où n est le résultat de getn(table). Si la fonction retourne n'importe quelle valeur non nulle, la boucle est interrompue et cette valeur est renvoyée comme valeur finale de foreachi. Cette fonction pourrait être définie en **Lua**:

```
function foreachi (t, f)
 for x=1,getn(t) do
  local res = f(x, t[x])
  if res then return res end
 end
end
```

getglobal (name)
Prend la valeur d'une variable globale ou appelle une méthode balisée pour `getglobal''. Le nom de la chaîne n'est pas nécessairement un nom de variable syntaxiquement valide.

getn (table)
Retourne la ``size'' (taille) d'une table, quand elle est vue comme une liste. Si la table a un champ n avec une valeur numérique, cette valeur est la taille (``size'') de la table. Sinon, la taille est l'index numérique le plus grand qui ait une valeur non nulle dans la table. Cette fonction pourrait être définie en **Lua**:

```
function getn (t)
 if type(t.n) == "number" then return t.n end
 local max = 0
 for i, _ in t do
  if type(i) == "number" and i>max then max=i end
 end
 return max
end
```

gettagmethod (tag, event)
Retourne la méthode balisée courante pour une paire donnée (balise, événement). Cette fonction ne peut pas être utilisée pour avoir une méthode balisée pour l'événement ``gc''. (De telles méthodes balisées ne peuvent être manipulées que par du code C)

globals ([table])
Retourne la table courante de globales. Si l'argument de la table est donné, elle déclare aussi cette table comme table des globales.

newtag ()
Retourne une nouvelle balise.

next (table, [index])
Permet à un programme de balayer tous les champs d'une table. Son premier argument est une table, son deuxième argument est un index de la table,. next retourne l'index suivant de la table et la valeur associée à cet index. Quand elle est appelée avec une valeur nulle comme deuxième argument, next retourne le premier index de la table et sa valeur associée. Quand elle est appelée avec le dernier index, ou avec une valeur nulle dans une table vide, next retourne une valeur nulle. Si le second argument est absent, il est alors interprété comme étant nul.
Lua n'a pas de déclaration de champs ; sémantiquement, il n'y a pas de différence entre un champ qui n'est pas présent dans une table et un champ dont la valeur est nulle. Par conséquent, next ne prend en compte que les champs qui ont une valeur non nulle. L'ordre dans lequel les indices sont énumérés n'est pas spécifié, même pour des indices numériques (pour balayer une table dans l'ordre numérique, utilisez un for numérique ou la fonction foreachi).
Le comportement du next est indéfini si vous changez la table pendant le balayage.

print (e1, e2, ...)
Reçoit n'importe quel nombre d'arguments et imprime leurs valeurs en utilisant les chaînes renvoyées par tostring. Cette option n'est pas adaptée pour les sorties formatées, c'est seulement un moyen rapide de montrer une valeur, pour le débogage par exemple.

rawget (table, index)
Prend la valeur réelle de table [index], sans invoquer de méthode balisée. table doit être une table, et index peut être n' importe quelle valeur non nulle.

rawset (table, index, value)
Fixe la valeur réelle de table[index] à value, sans invoquer de méthode balisée. table doit être une table, index index peut être n'importe quelle valeur non nulle, et value est n'importe quelle valeur **Lua**.

setglobal (name, value)
Fixe la variable globale désignée à la valeur donnée, ou appelle une méthode balisée pour ``setglobal''. Le nom de la chaîne n'est pas nécessairement un nom de variable syntaxiquement valide.

settag (t, tag)
Fixe la balise d'une table donnée. tag doit être une valeur créée avec newtag . settag retourne la valeur de son premier argument (la table).

settagmethod (tag, event, newmethod)

Fixe une nouvelle méthode balisée pour la paire donnée (tag, event) et retourne l'ancienne méthode. Si newmethod est nulle, alors settagmethod rétablit le comportement par défaut pour l'événement donné. Cette fonction ne peut pas être utilisée pour fixer une méthode balisée pour l'événement ``gc''. (De telles méthodes balisée ne peuvent être manipulées qu'avec du code C)

sort (table [, comp])
Trie les éléments d'une table dans un ordre donné, surplace, de table[1] à table[n], où n est le résultat de getn(table) . Si comp est donné, il doit être une fonction qui reçoit deux éléments table, et retourne vrai (c'est-à-dire une valeur non nulle)quand le premier est inférieur au second.(si bien que not comp(a[x+1], a[x]) sera vrai après le tri). Si comp n'est pas donné, c'est l'opérateur standard **Lua** < qui est alors utilisé.
L'algorithme de tri n'est pas stable (c'est-à-dire que des éléments considérés comme égaux d'après l'ordre donné peuvent avoir des positions relatives changées par le tri).
tag (v)
Permet aux programmes **Lua** de tester la balise d'une valeur. Elle reçoit un seul argument et retourne sa balise 'un nombre)

tonumber (e [, base])
Essaie de convertir son argument en nombre. Si l'argument est déjà un nombre ou une chaîne convertible, alors tonumber retourne ce nombre ; sinon elle retourne la valeur nulle.
Un argument facultatif spécifie la base à utiliser pour interpréter un numéral. La base peut être n'importe quel entier entre 2 et 36 (compris). Dans des bases au-dessus de 10, la lettre `A' (majuscule ou minuscule) représente 10, `B' représente 11, et ainsi de suite,`Z' représentant 35. En base 10 (par défaut), le nombre peut avoir une parie décimale ainsi qu'une partie exposant facultative. Dans d'autres bases, seuls les entiers non signés sont acceptés.

tostring (e)
Reçoit un argument de n'importe quel type et le convertit en chaîne dans un format adéquat. Pour avoir un contrôle complet sur la conversion des nombres, utilisez la fonction format.

tinsert (table [, pos] , value)
Insert la valeur de l'élément à la position pos de la table, en décalant les autres éléments dans un espace libre si nécessaire. La valeur par défaut de pos est n+1, où n est le résultat de getn(table), de façon à ce qu'un appel de tinsert(t,x) insère x à la fin de la table t. Cette fonction fixe aussi ou incrémente le champ n de la table à n+1. Cette fonction est l'équivalent de la fonction **Lua** suivante, mis à part que les accès à la table sont tous bruts (c'est-à-dire sans méthodes balisées):

```
function tinsert (t, ...)
 local pos, value
 local n = getn(t)
 if arg.n == 1 then
  pos, value = n+1, arg[1]
 else
  pos, value = arg[1], arg[2]
 end
 t.n = n+1;
 for i=n,pos,-1 do
  t[x+1] = t[x]
 end
 t[pos] = value
end
```

tremove (table [, pos])
Enlève de la table l'élément de la position pos, décalant les autres éléments pour garnir l'espace, si nécessaire. Retourne la valeur de l'argument enlevé. La valeur par défaut de pos est n, où n est le résultat de getn(table), de façon à ce qu'un appel à tremove(t) enlève le dernier élément de la table t. Cette fonction fixe aussi ou décrémente le champ n de la table à n-1.
Cette fonction est équivalente à la fonction **Lua** suivante sauf que les accès à la table sont tous bruts (c'est-à-dire sans méthodes balisées):

```
function tremove (t, pos)
 local n = getn(t)
 if n<=0 then return end
 pos = pos or n
 local value = t[pos]
 for x=pos,n-1 do
  t[x] = t[x+1]
 end
 t[n] = nil
 t.n = n-1
 return value
end
```

type (v)

Permet aux programmes **Lua** de tester le type d'une valeur. Elle reçoit un argument et retourne son type, codé dans une chaîne. Les résultats possibles de cette fonction sont "nil" (une chaîne, pas la valeur nulle), "number", "string", "table" and "function" (* (page 99))

Manipulation de chaîne

Cette bibliothèque fournit les fonctions génériques pour la manipulation de chaînes, comme la recherche et l'extraction de sous-chaînes et la concordance de modèles. Lors de l'indexation d'une chaîne en **Lua**, le premier caractère est à la position 1 (pas 0, comme en C). On admet aussi des indices négatifs et ils sont alors interprétés comme une indexation à l'envers, en partant de la fin de la chaîne. Par conséquent,.le dernier caractère est à la position -1, etc...

strbyte (s [, i])
Retourne le code numérique interne du i-ème caractère de s. Si i est absent, on assume qu'il est à 1. i peut être négatif. Les codes numériques ne sont pas nécessairement portables d'un système à l'autre.

strchar (i1, i2, ...)
Reçoit 0 ou plus d'entiers. Elle retourne une chaîne dont la longueur est égale au nombre d'arguments, dans laquelle chaque caractère a un code numérique interne égal à son argument correspondant.
Les codes numériques ne sont pas nécessairement portables d'un système à l'autre.

strfind (s, pattern [, init [, plain]])
Recherche la première équivalence du modèle dans s. Si elle en trouve une, alors strfind retourne les indices de s où l' occurrence commence et finit ; sinon elle retourne la valeur nulle. Si le modèle spécifie captures (voir gsub ci-dessous), les chaînes capturées sont renvoyées comme résultats supplémentaires. Un troisième argument numérique init spécifie où commencer la recherche ; sa valeur par défaut est 1, et peut être négative. Une valeur de 1 comme quatrième argument plain facultatif interdit les routines d'équivalence, de façon à ce que la fonction fasse une opération ``find substring'' simple, sans caractères dans le modèle qui soient considérés comme ``magic''. Notez que si plain est donné, alors init doit être donné aussi.

strlen (s)
Reçoit une chaîne et retourne sa longueur. La chaîne vide "" a une longueur de 0. Les zéros imbriqués sont comptés et donc "a\000b\000c" a une longueur de 5.

strlower (s)
Reçoit une chaîne et retourne une copie de cette chaîne dont toute les lettres sont en minuscules. Les autres caractères sont laissés inchangés. La définition d'une lettre majuscule dépend du système local.
strrep (s, n)
Retoune une chaîne qui est la concaténation de n copies de la chaîne s.

strsub (s, i [, j])
Retourne une autre chaîne, qui est une sous-chaîne de s, commençant à i et allant jusqu'à j ; i et j peuvent être négatifs, si j est absent il est assumé qu'il est égal à -1 (autrement dit la longueur de la chaîne). En particulier, l'appel de strsub(s, 1,j) retourne un préfixe de s de longueur j, et l'appel à strsub(s, -i) retourne un suffixe de s de longueur i.

strupper (s)
Reçoit une chaîne et retourne une copie de cette chaîne dont toutes les lettres minuscules ont été mises en majuscules. Tous les autres caractères sont inchangés. La définition d'une lettre minuscule dépend du système local.

format (formatstring, e1, e2, ...)
Retourne une version formatée du nombre variable d'arguments Returns en suivant la description faite dans le premier argument (qui doit être une chaîne). La chaîne format suit les mêmes règles que la famille printf des fonctions C standard. Les seules différences sont que les options/modificateurs *, l, L, n, p, et h ne sont pas acceptés et qu'il y a une option supplémentaire, q. L'option supplémentaire q formate une chaîne de telle façon qu'elle soit lisible à nouveau sans problème par l'interpréteur **Lua**. La chaîne est écrite entre deux guillemets, et tous les guillemets, les retours et les antislash de la chaîne sont échappés correctement quand ils sont écrits. Par exemple, l'appel:

```
    format('%q', 'une chaîne avec des "guillemets" et \n nouvelle ligne')
```
produira la chaîne:

```
"une chaîne avec des "guillemets" et "
nouvelle ligne"
```

Les conversions peuvent être appliquées au n-ième argument de la liste d'arguments plutôt qu'à l'argument suivant inutilisé. Dans ce cas, le caractère de conversion % est remplacé par la séquence %d$, où d est un chiffre décimal dans l'intervalle [1,9], donnant la position de l'argument dans la liste des arguments. Par exemple, l'appel format("%2$d -> %1$ 03d", 1, 34) aura pour résultat "34 -> 001". Le même argument peut être utilisé dans plusieurs conversions.
Les options c, d, E, e, f, g, G, i, o, u, X, and x attendent toutes un nombre comme argument, alors que q et s attendent une chaîne. Le modificateur * peut être simulé en construisant la chaîne format appropriée. Par exemple, "%*g" peut être simulé par "%"..width.."g".
Ni la chaîne format ni les valeurs de chaîne à formater avec %s ne peuvent contenir de zéros encapsulés. %q gère les valeurs de chaîne contenant des zéros encapsulés.

gsub (s, pat, repl [, n])
Retourne une copie de s dans laquelle toutes les occurrences du modèle pat ont été remplacées par une chaîne de remplacement spécifiée par repl. gsub retourne aussi, comme seconde valeur, le nombre total de substitutions faites.
Si repl est une chaîne, alors sa valeur est utilisée pour le remplacement. Toute séquence dans repl de forme %n, n étant

compris entre 1 et 9 représente la valeur de la n-ième sous-chaîne capturée.
Si repl est une fonction, alors cette fonction est appelée chaque fois qu'une équivalence est trouvée, toutes les sous-chaînes étant alors passées comme arguments, dans l'ordre (voir ci-dessous). Si la valeur retournée par cette fonction est une chaîne, alors elle est utilisée comme chaîne de remplacement ; sinon la chaîne de remplacement est une chaîne vide.
Le dernier paramètre facultatif n limite le nombre maximum de substitutions à faire. Par exemple, quand n est à 1 seule la première occurrence de pat est remplacée.
Voici quelques exemples:

```
x = gsub("hello world", "(%w+)", "%1 %1")
--> x="hello hello world world"

x = gsub("hello world", "(%w+)", "%1 %1", 1)
--> x="hello hello world"

x = gsub("hello world from Lua", "(%w+)%s*(%w+)", "%2 %1")
--> x="world hello Lua from"

x = gsub("home = $HOME, user = $USER", "%$(%w+)", getenv)
--> x="home = /home/roberto, user = roberto" (for instance)

x = gsub("4+5 = $return 4+5$", "%$(.-)%$", dostring)
--> x="4+5 = 9"

local t = {name="lua", version="4.0"}
x = gsub("$name - $version", "%$(%w+)", function (v) return %t[v] end)
--> x="lua - 4.0"

t = {n=0}
gsub("first second word", "(%w+)", function (w) tinsert(%t, w) end)
--> t={"first", "second", "word"; n=3}
```

Modèles

Classe de caractères:
Une classe de caractères est utilisée pour représenter un jeu de caractères. Les combinaisons suivantes sont autorisées pour décrire une classe de caractères
x
(où x est n'importe quels caractères magiques ^$()%.[]*+-?) - représente le caractère x lui-même.
.
- (a dot) représente tous les caractères.
%a
- représente toutes les lettres.
%c
- représente tous les caractères de contrôle.
%d
- représente tous les chiffres.
%l
- représente toutes les minuscules.
%p
- représente tous les signes de ponctuation.
%s
- représente tous les espaces.
%u
- représente toutes les majuscules.
%w
- représente tous les caractères alphanumériques.
%x
- représente tous les chiffres hexadécimaux.
%z
- représente les caractères de représentation 0.
%x
(où x est tout caractère non alpha-numérique) - représente le caractère x.
C'est la manière standard d'échapper les caractères magiques. Nous recommandons que tout signe de ponctuation (même les non-magiques) soit précédé de % quand il est utilisé pour se représenter lui-même dans un modèle.
[char-set]
- représente la classe qui est l'union de tous les caractères du jeu de caractères (char-set). Une étendue de caractères peut être spécifiée en séparant la fin des caractères du groupe par un tiret -. Toutes les classes %x décrites ci-dessus peuvent aussi être utilisées comme composants dans un jeu de caractères (char-set). Tous les autres caractères de char-set se représentent eux-mêmes. Par exemple, [%w_] (or [_%w]) représente tous les caractères alpha-numériques

plus le trait de soulignement, [0-7] représente les chiffres octaux et [0-7%l%-] représente les chiffres octaux plus les minuscules plus le tiret.
L'interaction entre les intervalles et et les classes n'est pas définie. Par conséquent, les modèles comme [%a-z] ou [a-%%] n'ont pas de sens.
[^char-set]
- représente le complément de char-set, où char-set est interprété comme ci-dessus.
Pour toutes les classes représentées par des lettres uniques (%a, %c, ...), la lettre majuscule correspondante représente le complément de la classe. Par exemple, %S représente tous les caractères sauf l'espace.
La définition de lettre, espace, etc... dépend du système local. En particulier, la classe [a-z] peut ne pas être équivalente à %l. Préférez la deuxième écriture pour une meilleure portabilité.

Pattern Item:
Un élément de modèle peut être
* une classe à caractère unique, qui correspond à n'importe quel caractère unique de la classe;
* une classe à caractère unique suivie de *, qui correspond à 0 ou plus de répétitions de caractères de la classe. Ces éléments répétés correspondront toujours à la séquence la plus longue possible;
* une classe à caractère unique suivie de +, qui correspond à 1 ou pplus de répétitions de caractères dans la classe. Ces éléments répétés correspondront toujours à la séquence la plus longue possible;
* une classe à caractère unique suivie de -, qui correspond aussi à 0 ou plus de répétitions de caractères dans la classe. A la différence de *, ces éléments répétés correspondront toujours à la séquence la plus courte possible.;
* une classe à caractère unique suivie de ?, qui correspond à 0 ou 1 occurrence d'un caractère dans la classe;
* %n, pour n entre 1 et 9; un tel élément correspond à une sous-chaîne égale à la n-ième chaîne trouvée.(voir ci-dessous);
* %bxy, où x et y sont deux caractères distincts;un tel élément correspond aux chaînes qui commencent par _x, se terminent par _y, et où x, y sont équilibrés. Cela signifie que, si on lit la chaîne de droite à gauche, en comptant +1 pour un x et -1 pour un y, le y final est le premier y pour lequel le comptage atteint 0. Par exemple, l'élément %b() correspond à l'expression dont les parenthèses sont équilibrées.

Modèle:
un modèle est une séquence d'éléments modèles. Un ^ au début d'un modèle fixe la correspondance au début de la chaîne ciblée. Un $ à la fin d'un modèle fixe la correspondance à la fin de la chaîne ciblée. A d'autres emplacement, ^ et $ n'ont pas de signification particulière et se représentent eux-mêmes.

Captures:
Un modèle peut contenir des sous-modèles placés entre parenthèses, ils décrivent les correspondances. Quand une correspondance est trouvée, les sous-chaînes de la chaîne ciblée qui correspondent à celle qui a été trouvée sont stockées (capturées) pour utilisation ultérieure. Ces captures sont numérotées selon leur parenthèses gauches. Par exemple, dans le modèle "(a*(.)%w(%s*))", la partie de la chaîne qui correspond à "a*(.)%w(%s*)" est stockée en tant que première capture (et porte donc le numéro 1); le caractère correspondant à . est capturé avec le numéro 2, et la partie correspondant à %s* porte le numéro 3.
Un modèle ne peut pas contenir de zéros encapsulés. Utilisez plutôt %z .

Fonction mathématiques

Cette bibliothèque est l'interface de certaines des fonctions de la bibliothèque C_math du C standard. En plus, elle inclut une méthode balisée pour l'opérateur binaire ^ qui retourne x^y quand elle est appliquée aux nombres x^y.
Cette bibliothèque contient les fonctions suivantes:

abs acos asin atan atan2 ceil cos deg exp floor log log10
max min mod rad sin sqrt tan frexp ldexp random randomseed

plus une variable globale PI. La plupart d'entre elles ne sont que des interfaces des fonctions homonymes de la bibliothèque C, sauf que, pour les fonction trigonométriques, tous les angles sont exprimés en degrés, pas en radians. Les fonctions deg et rad peuvent être utilisées pour les conversions en radians ou degrés.
La fonction max retourne la valeur maximum de ses arguments numériques. De la même façon, min calcule le minimum. On peut utiliser les deux avec 1, 2 arguments ou plus.
Les fonctions random and randomseed sont interfacées avec les fonctions du générateur aléatoire rand and srand, fournies en ANSI C. (on ne peut donner aucune garantie de leurs propriétés statistiques) La fonction random, quand elle est appelée sans argument, retourne un nombre réel pseudo-aléatoire compris dans l'intervalle [0,1]. Quand elle est appelée avec un nombre n, random retourne un entier pseudo-aléatoire compris dans l'intervalle [1,n]. Quand elle est appelée avec deux arguments, l et u, random retourne un entier pseudo-aléatoire compris dans l'intervalle [l,n].

2.10.5. La syntaxe complète de Lua

```
    chunk ::= {stat [`;']}

    block ::= chunk

    stat ::= varlist1 `=' explist1
  | functioncall
  | do block end
  | while exp1 do block end
  | repeat block until exp1
  | if exp1 then block {elseif exp1 then block} [else block] end
  | return [explist1]
  | break
  | for `name' `=' exp1 `,' exp1 [`,' exp1] do block end
  | for `name' `,' `name' in exp1 do block end
  | function funcname `(' [parlist1] `)' block end
  | local declist [init]

    funcname ::= `name' | `name' `.' `name' | `name' `:' `name'

    varlist1 ::= var {`,' var}

    var ::= `name' | varorfunc `[' exp1 `]' | varorfunc `.' `name'

    varorfunc ::= var | functioncall

    declist ::= `name' {`,' `name'}

    init ::= `=' explist1

    explist1 ::= {exp1 `,'} exp

    exp1 ::= exp

    exp ::= nil | `number' | `literal' | var | function | upvalue
  | functioncall | tableconstructor | `(' exp `)' | exp binop exp | unop exp


    functioncall ::= varorfunc args | varorfunc `:' `name' args

    args ::= `(' [explist1] `)' | tableconstructor | `literal'

    function ::= function `(' [parlist1] `)' block end

    parlist1 ::= `...' | `name' {`,' `name'} [`,' `...']

    upvalue ::= `%' `name'

    tableconstructor ::= `{' fieldlist `}'
    fieldlist ::= lfieldlist | ffieldlist | lfieldlist `;' ffieldlist | ffieldlist `;' lfieldlist
    lfieldlist ::= [lfieldlist1]
    ffieldlist ::= [ffieldlist1]
    lfieldlist1 ::= exp {`,' exp} [`,']
    ffieldlist1 ::= ffield {`,' ffield} [`,']
    ffield ::= `[' exp `]' `=' exp | `name' `=' exp

    binop ::= `+' | `-' | `*' | `/' | `\^{ ' | `..'
  | `<' | `<=' | `>' | `>=' | `==' | `\_{ '=}
  | and | or}

    unop ::= `-' | not
```

2.10.6. Notes de l'équipe Myriad

Voici quelques remarques, mises en garde et conseils à propos de Lua que nous avons trouvés lors de l'écriture de nos premiers scripts en langage **Lua**. Notez cependant que ces remarques sont directement liées à notre expérience de programmeurs. Peut-être rencontrerez-vous des problèmes que nous n'avions pas considérés comme tels...

Division par zéro

C'est un gros problème en **Lua** : sur certains systèmes, la division par zéro quitte l'application.
La bibliothèque Lua a été modifiée pour générer une erreurs Lua standard au lieu d'un crash quand il y a une division par zéro..

```
-- Ce programme écrira un message d'erreur
-- dans la fenêtre des logs au lieu de générer un crash.
print(10/0)
```

Vérification du faux, vrai et nul

Dans beaucoup de langages de programmation, la valeur nulle (nil ou NULL) représente une valeur de 0. De même que la valeur booléenne faux (false).
Dans de tels langages, vous pouvez écrire soit

if (myVariable==true) then ...
soit
if (myVariable) then ...

Ces deux syntaxes ayant le même effet.

Ce n'est PAS le cas en Lua. Nil est un type, pas une valeur.
Ce qui veut dire que :

if (myVariable==true) then ...
va réussir si myVariable est défini (non-nul) ET si sa valeur est vrai, comme attendu, alors que :
if (myVariable) then ...
ne va que vérifier l'existence de myVariable, et donc va réussir si la variable existe, quelle que soit sa valeur (vrai ou faux).
Donc, en Lua, cette dernière syntaxe est en fait équivalente à :
if myVariable~=nil then ...

```
-- pour tester, choisir une des ces écritures pour la ligne suivante:
-- variable = true
-- variable = false
-- variable = nil
-- ou pas de ligne du tout
variable=false

if variable then print("cette variable existe")
else print("cette variable n'existe pas (nul)")
end
if variable==true then print(" cette variable est vraie") end
if variable==false then print(" cette variable est fausse") end
```

index de table (tableau) non-entier

Dans beaucoup de langages de programmation, l'index d'un tableau est automatiquement une valeur entière. En Lua, un index de tableau (table)peut être soit un entier, un nombre en virgule flottante ou une chaîne.
Ce qui veut dire que :

myTable[10] est le même élément que myTable[20/2], mais pas le même que myTable[10.5], myTable[21/2] or myTable["10"].

2.10.7. Modification de la syntaxe standard de Lua

Afin d'implémenter les types objets qui permettent à un script de gérer les objet programme, et pour simplifier la tâche du programmeur, certains changements ont été effectués dans la syntaxe habituelle de Lua.
Ces changements ne peuvent s'appliquer qu'aux types objets prédéfinis.
Si vous créez votre propre type d'objet dans votre script, ils suivront la syntaxe standard comme expliqué dans la documentation sur Lua.

Types Objets

Tous les nouveau types objets ont été implémentés en tant que tables Lua. Cependant ce ne sont pas **vraiment** des tables Lua.
En définissant un champ avec "object.field=value", vous définissez en fait le champ correspondant dans l'objet interne de Harmony Assistant.
En lisant un champ avec "value=object.field", vous obtenez des renseignement sur le champ correspondant dans l'objet interne de Harmony Assistant.
Cela veut dire que, dans certains cas spéciaux, certains champs peuvent être lus mais pas écrits (lecture seule), ou que plusieurs champs peuvent avoir des manières différentes de lire ou écrire la même valeur interne,donc fixer la valeur de l'un va changer la valeur de tous.

Méthodes d'appel

En appelant une méthode pour de tels objets, la syntaxe Lua "object:method(paramètres)" ne doit plus être utilisée.
Pour assurer la compatibilité avec la plupart des langages orientés objets actuels, une méthode doit être appelée avec " object.method(paramètres)" .

Valeurs

Les valeurs possibles pour les types objets ont été étendues. La fonction "type" en Lua standard peut désormais retourner le nom du type objet comme défini dans ce manuel : "Score" (partition),"Staff"(portée),"Symbol"(symbole)...

Contenu des objets

Les objets de ces nouveaux types ont des références à l'objet réel. Ce qui veut dire que, si vous transférez telle variable dans une autre (par exemple, score2=score1), vous ne ferez pas une copie de score1, mais vous allez faire en sorte que score2 se réfère en interne à score1.
Si, après cela, vous faites n'importe quel changement sur score1, la valeur de score2 sera changée (et vice-versa).

Comparaison d'objets

Ces nouveaux objets peuvent être facilement comparés, sans les restrictions qui s'appliquent aux tables Lua standard.
Si deux objets se réfèrent au même objet en interne, ils sont égaux. Sinon, ils sont différents.

2.11. Lexique

2.11.1. Généralités

Afin de vous aider dans la compréhension de la description des objets, vous trouverez ici un petit lexique Anglais-Français des termes les plus usités.

Termes musicaux

- Bar : Mesure
- Clef : Clef
- Diagram : Diagramme
- Dynamic : Nuance
- Engraver : Mode gravure
- Grace note : Appogiature
- Key Signature : Tonalité
- Note : Note
- Ornament : Ornement
- Rest : Silence
- Score : Partition
- Setting : Configuration
- Slur : Coulé
- Staff : Portée
- Tie : Liaison
- Dot : Pointé
- Long : Longue
- Breve : Breve
- Whole : Ronde
- Half : Blanche
- Quarter : Noire
- Height : Croche
- 16th : Double-croche
- 32th : Triple-croche
- 64th : Quadruple-croche

Termes informatiques

- Device : Périphérique
- Display : Affichage
- File : Fichier
- Folder : Dossier
- Path : Chemin

Termes de programmation

- Breakpoint : Point d'arrêt
- Debug : Déboguer
- Item : Elément
- Method : Méthode
- Object : Objet
- Progress Bar : Barre de progression
- Step : Pas à pas

Termes divers

- Send : Envoyer
- Above : Au dessus
- Below : Au dessous
- to handle : Gérer
- a handle : Poignée
- Split : Diviser

3. Global keywords

3.1. Script

This section describes special language keywords related to scripts management

Include(...)

Description : Include a script in another script. Typically, the included script is a set of definitions or general functions. For example
- "MSDefine (page 100)" All current constants
- "MSLibrary (page 111)" A set of very useful functions
- "MSSounds (page 110)" Definition of all the sound IDs you can use.

File name is provided without path. It is searched first in the "Includes" folder of the user area. If not found, it is searched in the program "Includes" folder.

You can write, for example, **Include ("MSDefine.mys")** or more simply **Include "MSDefine"**.
Parameters : String : the file to include
Return value : None

Use(...)

Description : Does nothing, but declares the file as being used by the script. When building a compressed archive for this script, the file will be added.
For example:
Use "MyInterface.myc"
Parameters : String : the file to declare as used
Return value : None

Run(...)

Description : Run a script from another script. The launched script will share its global variables and functions.
The file name is provided without path. It is searched first at the same level as the parent script. If the script is not found, its name is searched in the list of installed scripts.
Parameters : String : the file to run
Return value : Boolean : true if the script has been launched.

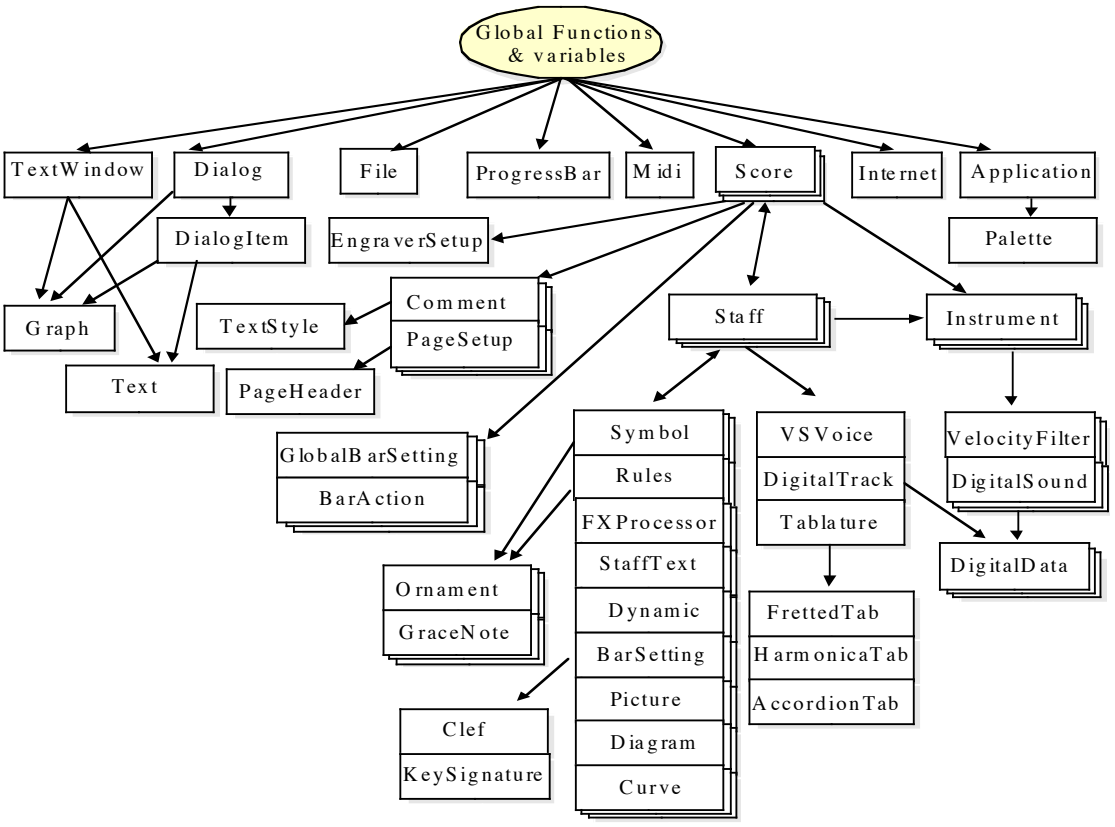
4. Object types

4.1. Overview

Many object types have been defined in order to manage access to internal program objects. From most of such types, it is possible to access other object types by getting a value or calling a method.
For example, from a score, you will be able to browse the staves, and from each of these staves, browse the miscellaneous music symbols that appear on this staff...

Below is the **object type tree**. From each object type, it shows what other object type can be retrieved.

Arrows point to the object type that can be retrieved. Stacked frames indicates the multiple or indexed object types, i.e. objects that can are usually present as a list, and can be browsed through Next and Previous fields.



In the following part of this manual, each method and field of each object type will be described.

A field can either be read (**variable=object.field** syntax) or written (**object.field=value** syntax) except when the field name is followed by the R icon. In this case, the field can only be read.

Be careful : such read-only fields can provide access to other objects. That does not mean the embedded object is read-only too.
For example, Score.FirstStaff (page 208) is read-only R, which means you are not allowed to set a value in this field by wri myStaff.FirstStaff=value. But fields in the Staff (page 252) object you get by reading this field may be read or written, depending on their own read/write status.

4.2. Application Object Type

General description of this object type : Application object type. There is only one object of this type (Application global variable), which refers to the currently running application, in this case Harmony Assistant.
You can read values from this object and apply specific high-level settings.

4.2.1. Application global functions

4.2.1.1. Script

SetErrorMessage(...)

Description : When an error occurs on a script, this message will be added to the log window.
Parameters : Styled string (page 177): the message to display
Return value : None

4.2.1.2. Binary operation

This section describes special language keywords related to binary and boolean arithmetic.

BinAnd(...)

Description : Perform a binary AND on the input values.
Parameters : Number : First operand
Number : Second operand
Return value : Number : First operand AND Second operand
Note : Operands must be integer numbers, otherwise their integer value is processed.

BinOr(...)

Description : Perform a binary OR on the input values.
Parameters : Number : First operand
Number : Second operand
Return value : Number : First operand OR Second operand
Note : Operands must be integer numbers, otherwise their integer value is processed.

BinXor(...)

Description : Perform a binary XOR on the input values.
Parameters : Number : First operand
Number : Second operand
Return value : Number : First operand XOR Second operand
Note : Operands must be integer numbers, otherwise their integer value is processed.

BinShl(...)

Description : Perform a binary 32-bits left shift of the input value.
Parameters : Number : value to be shifted
Number : number of binary shifts (1..32)
Return value : Number : Shifted value
Note : Operands must be integer numbers, otherwise their integer value is processed.
Bits that fall over the 32th are lost.

BinShr(...)

Description : Perform a binary 32-bits right shift of the input value.
Parameters : Number : value to be shifted
Number : number of binary shifts (1..32)
Return value : Number : Shifted value
Note : Operands must be integer numbers, otherwise their integer value is processed.
Bits that fall below the 1st are lost.

4.2.1.3. Bit operation

BitTst(...)

Description : Test the Nth bit of the operand and return true if this bit is set.
Bits are numbered starting from 0, the 0th bit being the less significant (2^0)
Parameters : Number : Operand
Number : Bit number (0 to 31)
Return value : Boolean : true if the bit is set, false if clear
Note : Operands must be integer numbers, otherwise their integer value is processed.

BitClr(...)

Description : Return the operand with its Nth bit cleared.
Bits are numbered starting from 0, the 0th bit being the less significant (2^0)
Parameters : Number : Operand
Number : Bit number (0 to 31)
Return value : Operand with its nth bit cleared
Note : Operands must be integer numbers, otherwise their integer value is processed.

BitSet(...)

Description : Return the operand with its Nth bit set.
Bits are numbered starting from 0, the 0th bit being the less significant (2^0)
Parameters : Number : Operand
Number : Bit number (0 to 31)
Return value : Operand with its nth bit set
Note : Operands must be integer numbers, otherwise their integer value is processed.

4.2.1.4. String operations

ConvertStringEncoding(...)

Description : Converts a string from an encoding to another.
Four kinds of encoding are available (constants in MSDefine) :
- UNKNOWN_STR_ENCODING : unknown encoding, to be determined by the program by analyzing the string content
- MS_DOS_STR_ENCODING : MS-DOS encoding
- WINDOWS_STR_ENCODING : Windows encoding (Windows ANSI, occidental)
- MACINTOSH_STR_ENCODING : Macintosh encoding (Macintosh ANSI, occidental)
- HTML_STR_ENCODING : HTML encoding (7-bits ASCII entities, Unicode compatible)
- UTF8_STR_ENCODING: UTF-8 encoding
- MAC_SHIFT-JIS_ENCODING : Shift-JIS (Japanese ANSI extension) Macintosh encoding

Difference between encodings is the way accented characters are stored.
Parameters : String : String on which encoding has to be changed
Number : Original encoding
Number : Target encoding
Optional:
For HTML encoding (target encoding equal to HTML_STR_ENCODING)
  Boolean : true is space character is substituted by %20, false is space character is left as this. (false by default)
  Boolean : true if carriage return is substituted by <br>, false is left as this. (false by default)
For HTML decoding (source encoding equal to HTML_STR_ENCODING)
  Number : encoding for source 8-bits non-ansi characters
    Can be:
    UNKNOWN_STR_ENCODING non-ansi characters in source page are considered to be already encoded using the target encoding (default)
    WINDOWS_STR_ENCODING non-ansi characters in source page are Windows (ISO 8859) encoded
    MACINTOSH_STR_ENCODING non-ansi characters in source page are Mac Roman
    UTF8_STR_ENCODING non-ansi characters in source page are UTF-8
Return value : newly encoded string
Example :

```
Include "MSDefine"
a=""
a=AddRawChar(a,169) -- (c) (Macintosh)
a=AddRawChar(a,32) -- Space (ASCII)
a=AddRawChar(a,strbyte('M',1)) -- M (ASCII)
a=AddRawChar(a,strbyte('e',1)) -- e (ASCII)
a=AddRawChar(a,170)  -- tm (Macintosh)
print("Raw length of Macintosh string: "..RawStrlen(a))
utfstr=ConvertStringEncoding(a,MACINTOSH_STR_ENCODING,UTF8_STR_ENCODING)
 -- Displays it
print("Printing utf8 (unicode) conversion: "..utfstr)
print("In utf8, the strxxx functions make you think the string is stored in plain Unicode")
print(RawStrlen(utfstr).." raw characters for "..strlen(utfstr).." unicode chars")
```

Note : In a score, all internal strings are encoded as UTF-8 string, whatever the current platform is.
Also true for litteral strings in script source code, name and values of interface items, etc.
All MyrScript strxxx operations (strlen, strsub, or concatenation or print ...) only work on UTF-8 strings, and behave like
if the string was in 16-bits unicode, encoding and decoding back and forth character values from UTF-8 to plain unicode.
This encoding and decoding operations can fail (and even crash) when performed on encodings other than UTF-8.
Any attempt to use the regular string functions or operators on a Windows or Macintosh ANSI string, or on a Shift-Jis string can
result in a severe application crash.
In your programs, encodings others than UTF-8 have to be managed only transiently, and be converted to UTF-8 as soon as possible.
Because standard (UTF-8) strings can be styled, all informations of style are lost when encoding is changed.
The only available operations on non-UTF8 strings are:
RawStrlen(...) (below)
GetRawChar(...) (below)
SetRawChar(...) (below)
AddRawChar(...) (below)
DeleteRawChar(...) (page 177)

RawStrlen(...)

Description : Gets the raw length (number of bytes) of an encoded string.
Parameters : String: Encoded string
Return value : : The result string
Note : Fails when asking for a character after the end of the string

GetRawChar(...)

Description : Gets the raw ASCII (8-bits) value of the nth character in an encoded string.
Parameters : String: Encoded string
Number: raw position in string (1= first character)
Return value : Number: 8-bits value of the nth character of the encoded string
nil when asking for a character after the end of the string

SetRawChar(...)

Description : Sets (changes) the raw ASCII (8-bits) value of the nth character in an encoded string.
Parameters : String: Encoded string
Number: raw position in string (1= first character)
Number: 8-bits value of the nth character of the encoded string
Return value : String: New string
Note : When setting a character after the end of the string, appends this character to the string.

AddRawChar(...)

Description : Adds a raw character to an encoded string, without trying to "understand" what this character is.
Parameters : String: Encoded string
Number: ASCII value (8 bits) to add
Return value : String: The result string
Note : This function is equivalent to SetRawChar(string,RawStrlen(string)+1)

DeleteRawChar(...)

Description : Deletes the nth raw character from an encoded string, without trying to "understand" what this character is.
Parameters : String: Encoded string
Number: raw position in encoding string of the byte to delete
Return value : String: The result string

4.2.1.5. Local object copy management

Local copies are used to store all the values of a given object into a regular Lua table.

GetLocalCopy(...)

Description : Performs a local copy of the object into a regular Lua table
Parameters : Any type : Object to get local copy of
(optional) Boolean : true if you want only the writeable fields to be defined in the local lua table (default : false)
(optional) Boolean : true to restrict the copy to only Boolean, strings and numbers (no sub-objects)
Return value : Table : local table that contains a copy of all the original object values
Example :

```
localscore=GetLocalCopy(FrontScore()) -- Create a local copy of the front score
foreach(localscore,function(i,v) print(i,"=",v) end) -- Display its fields
```

Note : The output table can be used to examine, store locally or save the values of an object.
Contrarily to the object itself, the lua table are no more directly linked to the actual parameters of the object.
It means changing them won't affect any object on score, and they remain valid even when the object is closed or disposed.
You can then use PutLocalCopy(...) to store the fields of the Lua table into another object of the same type.
Fields you do not want to store can be invalidated by setting them to nil.

PutLocalCopy(...)

Description : Stores a regular Lua table into an object
Parameters : Table : lua table
Any type : Object to put local copy to
Return value : None
Example :

```
staff=FrontScore().FirstStaff
staff2=staff.Next
-- Get parameters of 1st staff
localstaff=GetLocalCopy(staff)
-- Store them in 2nd staff
PutLocalCopy(localstaff,staff2)
```

Note : This function searches the lua table for each writeable field of the object.
If found, the value is copied from the Lua table to the matching field in the object.
The lua table can be got from GetLocalCopy(...) or built programatically.
Important : only fields of type : Number, Boolean or String are written into the object.
All other fields are ignored.

4.2.1.6. Styled string

This section describes operation on styled string (above). A styled string (above) is a string including font, face, size, colo changes.

PurgeStringStyle(...)

Description : From a styled string (above) provide a string without style. If no style is currently related to the string, an identical string is returned.
Parameters : Styled String
Return value : String

DeleteStringStyle(...)

Description : Delete a part of a string
Parameters : String
Number : first character to delete from 1 to the end of the string
Optional : Number : last character to delete, by default to the end
Return value : Styled String

InsertStringStyle(...)

Description : Insert a styled string (page 177) inside a styled string (page 177)
Parameters : String : String to insert
String : Target string
Optionnal : Number : Position in target string, by default at the beginning (if lower than 1, the string will be added at the beginning of the target string)
Return value : Styled String

SetStringStyle(...)

Description : Set the string or a part of a string to a specified style
Parameters : String : the source string
Optional : Number : The first character in the string to apply the style from 1 to the end of the string (default: 1)
Optional : Number : The last character in the string to apply the style, or 0 if the style will be applied to the whole string (default:0)
Optional : String : The font name to apply or -1 if you don't want to change the font (default: see discussion below)
Optional : String : The face to apply (B,I,U,O,S,C,E,N) or number -1 if you don't want to change the face (default: see discussion below)
Optional : Number : The size to apply or -1 if you don't want to change the size (default: see discussion below)
Optional : String : The color to apply (RGB components) or number -1 if you don't want to change the color (default: see discussion below)

Discussion:
The default value for optional parameters is -1 (no change) as soon as at least 2 parameters are provided.
If only one parameter is provided (as in SetStringStyle(string)) the style for the whole string is set to a standard font:

Geneva, standard, 9, 000000
Return value : String : the modified string with style applied
Example :

```
score=FrontScore()
staff=score.FirstStaff
-- Set all the title of the first staff of the front score to Times bold italic, size 14, red color
staff.Title=SetStringStyle(staff.Title,0,0,"Times","bi",14,"FF0000")
-- Change the first character to size 18
staff.Title=SetStringStyle(staff.Title,1,1,-1,-1,18,-1)
```

GetStringStyle(...)

Description : Get the styles of a string
Parameters : String : the source string
Return value : An array.
First, there is one entry per style, each entry provides:
Position: Number, the start position in the string, 1 based
Font: String, the font name
Face: String, the face name
Size: Number, the size
Color: String, the RGB color
The last entry only provides the Position, necessary to know the last style size.
Example :

```
-- Creating a styled string
str=SetStringStyle("Hello, ",0,0,"Times","BI",16)
str=str..SetStringStyle("how ",0,0,"Arial","",12)
str=str..SetStringStyle("are you ?",0,0,"Monaco","I",12,"FF0000")
-- We print it
print(str)
-- Extracting style table
table=GetStringStyle(str)
-- Iterating the table
for x=1,getn(table)-1 do
   -- Printing the styled substring
   print(strsub(str,table[x].Position,table[x+1].Position-1))
end
```

4.2.2. Application values

4.2.2.1. Human Voice

Application.humanVoice

Type : Number
Description : Provide the current human voice index
Example :

```
Message("The current voice is "..Application.GetHumanVoiceInformation(Application.humanVoice))
```

4.2.2.2. Interface

Application.Palette

Type : Palette (page 565) object type
Description : Provides access to all Application tool palettes.
Example :

```
Application.Palette.Open("Play")
```

Application.CurrentSymbolDuration

Type : Number
Description : Current note or rest duration selected in palettes (tuplet and dot are taken into accoun)

Application.IsNotePaletteActive

Type : Boolean
Description : Indicate if note palette is active. False if rest palette is active

4.2.2.3. Playing Music

Application.CurrentPlayingScore

Type : Score (page 198) object type
Description : Provides the current playing score. If no score is currently playing, return nil.
You can use the Score.PlayMusic(...) (page 237) method to play a score.

4.2.2.4. Settings

Application.Language

Type : Number
Description : Provides the current language that has been selected by the user. Languages are defined in MSDefine (page 100) as is :

LANGUAGE_FRENCH=0
LANGUAGE_ENGLISH=1
LANGUAGE_ITALIAN=2
LANGUAGE_GERMAN=3
LANGUAGE_SPANISH=4
LANGUAGE_JAPANESE=5
LANGUAGE_PORTUGUESE=6
LANGUAGE_DUTCH=7

This enables to adapt your interface to the current language.
Example :

```
Include "MSDefine"
if(Application.Language==LANGUAGE_ENGLISH) then
   Message("Current language is English")
   end
if(Application.Language==LANGUAGE_FRENCH) then
   Message("Current language is French")
   end
if(Application.Language==LANGUAGE_ITALIAN) then
   Message("Current language is Italian")
   end
if(Application.Language==LANGUAGE_GERMAN) then
   Message("Current language is German")
   end
if(Application.Language==LANGUAGE_SPANISH) then
   Message("Current language is Spanish")
   end
if(Application.Language==LANGUAGE_PORTUGUESE) then
   Message("Current language is Portuguese")
   end
if(Application.Language==LANGUAGE_JAPANESE) then
   Message("Current language is Japanese")
   end
```

Application.MyrScriptVersion

Type : String
Description : Provides the MyrScript version hosted by the Application. Enables to test for a minimal version of Myrscript for a script to run properly.
Example :

```
Message(Application.MyrScriptVersion)
```

Application.Version

Type : String
Description : Provides the Application name and version.
Example :

```
Message(Application.Version)
```

Application.Computer

Type : String
Description : Provides the platform type : "PC" (Windows) "Mac" (MacOS) or "Linux".
Example :

```
Message(Application.Computer)
```

Application.IsActive

Type : Boolean
Description : True if the application is the frontmost application, false if the application runs in background.

Application.CurrentSoundBaseVersion

Type : String
Description : Provides the current SoundBase version.
Example :

```
Message(Application.CurrentSoundBaseVersion)
```

Application.IsAngloSaxonNotation

Type : Boolean
Description : Provides the current user setting about note name notation. If true, notes are named A,B,C,D... if false, notes are named Do,Re,Mi...

Application.IsLeftHandedNotation

Type : Boolean
Description : Provides the current user setting about left handed notation

Application.IsInvertionNotation

Type : Boolean
Description : Provides the current user setting about chord invertion notation

Application.RGBSelectColor

Type : String
Description : Provides the current color of the selection.

Application.StorageArea

Type : StorageArea
Description : Gets the global storage area related to this session. This storage area will be lost at the end of the session and enables to store temporary values several scripts can share during a same program session.

Application.LimitEditingToOneBar

Type : Boolean
Description : "Limit editing to one bar" status

Application.NumberOfChordDiagrams

Type : Number
Description : Number of chord diagram in chord diagram database

4.2.2.5. Sound Base

Application.CurrentSoundBaseName

Type : String
Description : Provides the current SoundBase name
Example :

```
Message(Application.CurrentSoundBaseName)
```

4.2.2.6. Virtual Singer

Application.IsVirtualSingerInstalled

Type : Boolean
Description : Tells whether Virtual Singer is currently installed.
Example :

```
if(Application.IsVirtualSingerInstalled==true) then
   Message("Virtual Singer is installed")
else
   Message("Virtual Singer is not installed")
end
```

Application.VirtualSingerVersion

Type : String
Description : Provides the current Virtual Singer version.
Example :

```
Message(Application.VirtualSingerVersion)
```

Application.IsVirtualSingerActive

Type : Boolean
Description : Tells whether Virtual Singer is currently active.

4.2.3. Application methods

4.2.3.1. Converting

Application.ConvertStringToBase64(...)

Description : Convert a string to a string encoded in base64
Parameters : ·String : The string to encode
·Number (optional) : the line length
Return value : String : The bases64 encoded string with carriage return each line length.

Application.ConvertFileToBase64(...)

Description : Convert a file to a string encoded in base64
Parameters : ·String : The file name
·Number (optional) : the line length
Return value : String : The bases64 encoded string with carriage return each line length.

Application.ConvertBase64ToFile(...)

Description : Convert a string encoded in base64 to a file
Parameters : ·String : The file name
·String : The bases64 encoded string
RETURN
Boolean true if ok

Application.ConvertBase64ToString(...)

Description : Convert a a string encoded in base64 to a string
Parameters : String : The base 64 string to decode
Return value : String : The decoded string

Application.ConvertChordNameToElements(...)

Description : Convert a chord name to chord components.
Parameters : ·String : The chord name
·Boolean (Optional) True if the notation will follow the Anglo-saxon notation (C,D,E,F...) false, will follow the Italian notation (Do,Re,Mi,Fa...),if omitted the returned string follows the current user setting.
Return value : · Number : the root note (C=0,D=1,...)
· Number : the accidental as defined in MSDefine
· Number : the third as defined in MSDefine :
SEMITONE_OFFSET_DOUBLE_DIMINISHED=-2
SEMITONE_OFFSET_DIMINISHED=-1
SEMITONE_OFFSET_NONE=2
SEMITONE_OFFSET_MAJOR=0
SEMITONE_OFFSET_AUGMENTED=1
· Number : the fifth as defined in MSDefine.
· Number : the seventh as defined in MSDefine.
· Number : the ninth as defined in MSDefine.
Example :

```
-- Computing complex chord name
Include "MSDefine"
print(Application.ConvertElementsToChordName(true,1,NATURAL,SEMITONE_OFFSET_DIMINISHED,
SEMITONE_OFFSET_MAJOR,SEMITONE_OFFSET_MAJOR,SEMITONE_OFFSET_DIMINISHED))
```

Will display : "DmM7-9"

Application.ConvertElementsToChordName(...)

Description : Convert a chord description into a chord name.
Parameters : · Number : The notation kind, -1= default notation as defined by the user in the global preference, 0= Italian notation (Do,Re,Mi,Fa...),Anglo-saxon notation (C,D,E,F...)
· Number : the root note (C=0,D=1,...)
Optional :
· Number : the accidental as defined in MSDefine, by default NATURAL
· Number : the third as defined in MSDefine. By default : SEMITONE_OFFSET_MAJOR
SEMITONE_OFFSET_DOUBLE_DIMINISHED=-2
SEMITONE_OFFSET_DIMINISHED=-1
SEMITONE_OFFSET_NONE=2
SEMITONE_OFFSET_MAJOR=0
SEMITONE_OFFSET_AUGMENTED=1
· Number : the fifth as defined in MSDefine. By default : SEMITONE_OFFSET_MAJOR
· Number : the seventh as defined in MSDefine. By default : SEMITONE_OFFSET_NONE
· Number : the ninth as defined in MSDefine. By default : SEMITONE_OFFSET_NONE
Return value : String : the chord name
Example :
You can use these methods to convert chord written in Anglo-Saxon notation to Italian notation :

```
print(Application.ConvertElementsToChordName(false,Application.ConvertChordNameToElements("F#m7",true)))
```

will display "Fa#m7"

Application.ConvertPitchListToChord(...)

Description : Convert a pitch list into a chord.
Parameters : · Number : The notation kind, -1= default notation as defined by the user in the global preference, 0= Italian notation (Do,Re,Mi,Fa...),Anglo-saxon notation (C,D,E,F...)
· Pitch list (1 based)
· (optional) String. Key signature name.
· (optional) Number. Chord complexity (3,4 or 5) default 3
Return value :
String : the chord name
Lua table : pitches from 1 to 5, index 1=root note, 2=third, etc
Example :

```
local tab
local chordname
tab={}
tab[1]=61
tab[2]=63
tab[3]=64
chordname=Application.ConvertPitchListToChord(1,tab,"C",5)
root,accidental,third=Application.ConvertChordNameToElements(chordname)
print("name="..chordname..",root="..root..",accidental="..accidental..",third="..third)
```

will display "name=Em,root=2,accidental=2,third=-1"

Application.GetChordDiagramFromIndex(...)

Description : Provide the chord diagram from general chord diagram database
Parameters : Number : index (1 based)
Return value : Diagram (page 363) : object or nil if not found

Application.GetDigitalOutputDelay(...)

Description : Returns the current digital output delay (ms). This value is supposed to match the latency from the software/hardware
digital output interface (i.e. the delay between the time a sound is sent and the time it is actually heard)
Parameters : None
Return value : Current digital output delay (latency)

Application.SetDigitalOutputDelay(...)

Description : Sets the current digital output delay (ms). This value is supposed to match the latency from the software/hardware
digital output interface (i.e. the delay between the time a sound is sent and the time it is actually heard)
Parameters : Latency value (in millisecond) to set
Optional: true if you want to make these changes permanent (saves the digital settings file)
Return value : None

Application.StartRecording(...)

Description : Start recording digital data or notes from the MIDI or computer keyboard. You cannot start a recording if a recording is already running.
Recording stops when :

- End of piece is reached.
- Music is stopped by the user.
- Application.StopRecording(...) (page 185) method is called.
- Score.StopMusic(...) (page 238) method is called.

Parameters : Optional

- Score : the score where to record. If nil (or no parameter) the currently playing score is used.

If there is no playing score, the front window is used.
If there is no front window an empty document is created.

- Boolean, recording from digital input (true by default)
- Boolean, recording from MIDI input (false by default)
- Boolean, recording from computer keyboard input (false by default).
- Boolean, indicating whether a custom processing of the result will be applied (parameters follow). If false (by default) the standard dialog box will open.

Parameters for digital input :

- Staff, first destination staff for digital input (left channel or mono channel). If nil, a new staff is created. (nil by default). The staff will be returned by the StopRecording method.
- Staff, second destination staff for digital input (right channel). If nil, and if the input result provide a right channel, a new staff is created. (nil by default). The staff will be returned by the StopRecording method.
- Number, Applying mode. 0 : the data will be applied, 1 : the data will be inserted, 2 : the data will be added (0 by default)
- Number, Stereo computation mode. 0: the result will be mono, 1 : the result will be stereo, 2 : if the input is stereo, the result will be mono and the singer voice will be removed (1 by default)
- Boolean, If true blocking point will be respected. (false by default)

Return value : Score : the score where the recording has been started. Nil if error, for example, if a recording was already running.
Note : When recording is started, system tasks must be managed so that recorded digital data can be written on disk.
If your script is not a floating palette, Application.UserBreak(...) (page 187) must be called frequently to enable this.

Application.StopRecording(...)

Description : Stops the current recording, if any.
Parameters : None
Return value : Score : the score on which recording has been made. Nil if error.
Number : time starting position where data have been applied (in milliseconds)
Number : number of recorded samples
Staff : left result staff
Staff : right result staff

Application.AbortRecording(...)

Description : Aborts the current recording, if any.
If the score is empty (no staff), it is automatically disposed.
Parameters : None
Return value : None

Application.GetCurrentRecordedLength(...)

Description : Gets the currently recorded length, in samples.
Parameters : None
Return value : Number of samples already recorded (nil if no recording is in progress)
Note : Recorded samples are actually applied on score ONLY when recording stops.
This method provides an estimation of the number of already recorded samples.
Because of the way digital data are recorded, the returned value does not increase smoothly, but by blocks of about one second (44100 samples).

Application.GetLiveRecordedData(...)

Description : Gets a block of recorded data during the recording process. Recorded data are expressed as values between -1 and +1
Parameters : RawTable where to store the recorded data.
If RawTable is too small, only stores the n first recorded values
If RawTable is too big, the last values are padded with zeroes.
Return value : Number of available samples (all stored in the RawTable if large enough)
Number : block ID
Example :

```
-- Creates a RawTable large enough to store 1 sec of data
data=NewRawTable(60000) -- Must be more than 44100 (1 sec)
-- Creates a new, invisible score
score=NewScore(false)
if score then
   print("Hit spacebar to stop")
   -- Starts recording digital data
   Application.StartRecording(score,true,false,false,true,nil,nil,0,0)
   lastid=-1
   repeat
      -- Get live data in RawTable
      number,id=Application.GetLiveRecordedData(data)
      -- Only computes volume when buffer changes
      if id and id>=0 and id~=lastid then
         lastid=id
         -- Get maximum absolute value
         maxi=max(abs(data.MaximumValue),abs(data.MinimumValue));
         -- Display volume
         print("Input volume:",floor(100*maxi),"%")
      end
   until  Application.IsKeyCodePressed(49)==true
   -- Do not save result
   Application.AbortRecording() -- the fake score is disposed
   -- Dispose rawtable
   data.Dispose()
   print("Bye")
end
```

Note : Live data are returned by blocks of about 1 second.
The returned table contains the same data until block ID changes. You can use Application.TickCount(...) (page 186 manage delayed but smooth processing of a recorded block

4.2.3.2. Date & Time

Application.GetCurrentDate(...)

Description : Provides the current date.
Parameters : None
Return value : Numbers :
- Year
- Month
- Day
- Hour
- Minute
- Second

Example :

```
year,month,day=Application.GetCurrentDate()
Message("Current date is "..month.."/"..day.."/"..year)
```

Application.ConvertSystemTimeToDate(...)

Description : Converts a system time value to readable date and time.
Parameters : A system time value, as provided by FileModificationTime(...) (page 593)
Return value : Numbers :
- Year
- Month
- Day
- Hour
- Minute
- Second

Example :

```
-- Asks the user to select a file and displays its modification date.
local time
theFileName=GetFileNameLoad()
if theFileName ~=nil then
   time=FileModificationTime(theFileName)
   year,month,day=Application.ConvertSystemTimeToDate(time)
   Message("Modification date of "..theFileName.." is "..month.."/"..day.."/"..year)
end
```

Application.ConvertDateToSystemTime(...)

Description : Converts a date to system time.
Parameters :
- Year
- Month
- Day

Optional :
- Hour (by default 0)
- Minute (by default 0)
- Second (by default 0)

Return value : Number: System time
Example :

```
--- Displays the current date in system time format
year,month,day=Application.GetCurrentDate()
Message("Current system time is "..Application.ConvertDateToSystemTime(year,month,day))
```

Application.TickCount(...)

Description : Provides the number of 60th of seconds elapsed since the computer started.
Parameters : None
Return value : Number of 60th of seconds
Example :

```
Message("This computer is running from "..(Application.TickCount()/60).." seconds.")
```

Application.WaitDelay(...)

Description : Just waits. User can break using the ESC key. The "waiting" mouse cursor is displayed during this delay.
Parameters : Number of 60th of second to wait
Return value : None
Example :

```
Application.WaitDelay(100)
```

Application.StringDuration(...)

Description : Convert a duration in seconds to a string.
Parameters : Number of seconds
Return value : string
Example :

```
print(Application.StringDuration(60*60*60*25*2+4250))
```

4.2.3.3. Devices

Application.UserBreak(...)

Description : Checks whether the user wants to stop the running script. The user can press the "ESC" key on the keyboard or Command+dot or Command+mouse click.
If the user performs one of these actions, script is halted and the message "User break" displayed on the log window.
When your script can run a int32_t loop, call this method regularly.
Parameters : None
Return value : None

Application.KeyStroke(...)

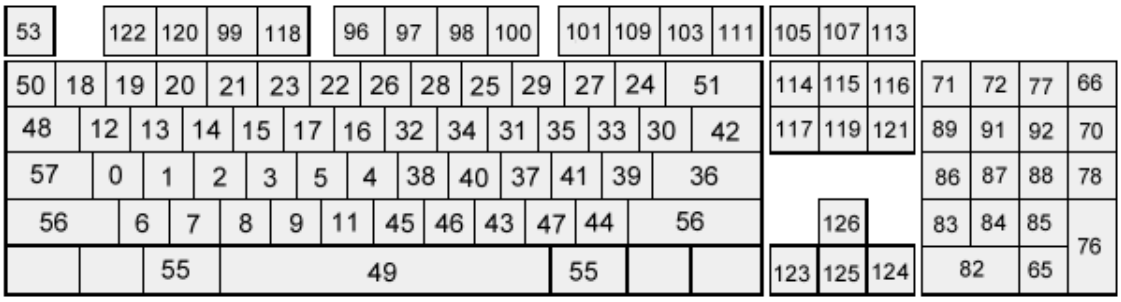
Description : Checks whether the user hits a key on the computer keyboard, and provides the key ASCII value. You cannot be sure about the location of the key on the keyboard because the user can own a different keyboard as yours.
If you want to map the keyboard you can useApplication.IsKeyCodePressed(...) (page 188)
Parameters : None
Return value : Key pressed on the keyboard or nil if no key.
Example :

```
--- Starts a loop until user hits a key
print("Hit keys on the keyboard or ESC to quit")
local myKey=0
while myKey~=27 do
   myKey=Application.KeyStroke()
   if myKey ~= nil then
      print("ASCII of this key is "..myKey)
   end
end
```

Application.IsKeyCodePressed(...)

Description : Checks whether a given key is pressed on the computer keyboard, using the key map.
The key map does not change from a computer to another of from a keyboard to another. It is language independent.
IsKeyCodePressed is faster as KeyStroke. You can use it to test more than one key pressed at the same time.

Here is the key map :

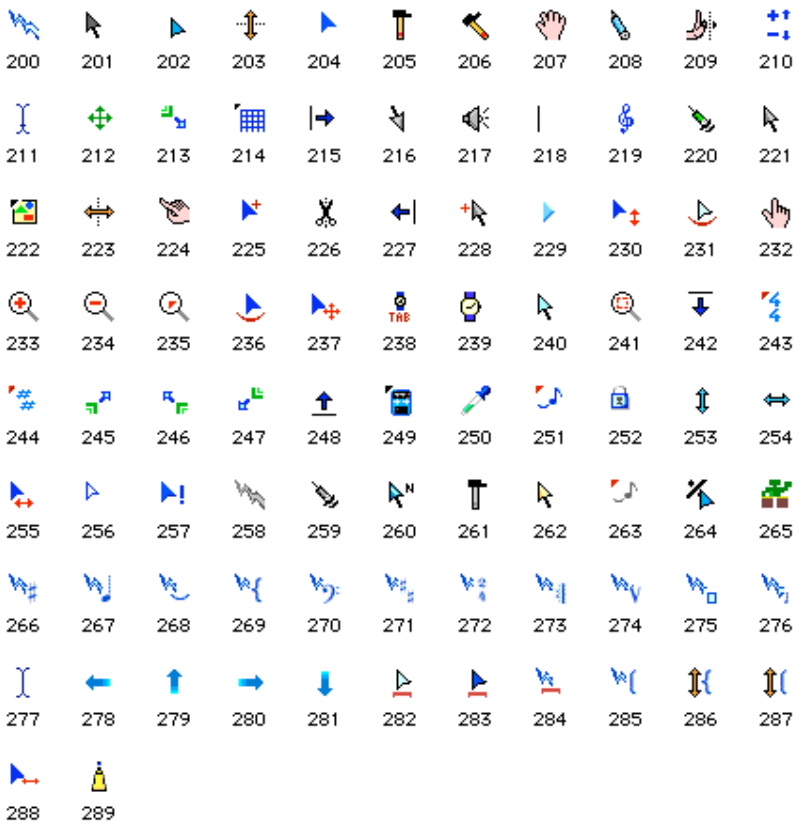


Parameters : Key code to test.
Return value : True if this key is pressed.
Example :

```
--- Starts a loop until user hits the ESC key
while Application.IsKeyCodePressed(53)==false do
   print("Hit The ESC key to exit")
end
```

Application.SetMouseCursorShape(...)

Description : Changes the mouse cursor shape. Please note, if the user moves the mouse on a document or another area related to a different mouse cursor shape, the cursor shape will change.
When you use a Floating Window, use the Dialog.SetMouseCursorShape(...) (page 505) method instead. This will relate the mouse cursor to the window, so the mouse cursor won't change.
Parameters : Number from 200 to 289, the shape to use.



Return value : None
Example :

```
Application.SetMouseCursorShape(224)
```

Application.GetMouseButton(...)

Description : Provides the current status of mouse button(s)
Parameters : None
Return value : Number: 0 if no button pressed, 1 if first button pressed, 2 it second button pressed, etc.

Be aware, mouses on Macintosh computers often have one button, the second button is emulated by l click + Ctrl key.

Application.GetMouseDoubleClickButton(...)

Description : Provides the current status double click event
Parameters : None
Return value : True if double click

Application.GetMousePosition(...)

Description : Provides the current mouse position in local coordinate of the dialog box window. (0,0 is the upper left corner of the dialog window user area)
Parameters : None
Return value : Two numbers : X position, Y position

Application.GetMouseGlobalPosition(...)

Description : Provides the current mouse position in global coordinates. (0,0 is the upper left corner of the screen)
Parameters : None
Return value : Two numbers : X position, Y position

4.2.3.4. Display

Application.UpdateScores(...)

Description : When your script applies changes to a score, this method updates all modified scores. This enables to show what you have done on a document.
Unmodified scores are not updated.
In any case, score windows are updated when the script ends.
Note : if you use a modal dialog window which apply changes to the score, modified score will be automaticaly updated when you close the dialog box.
But if you use a floating window (which remains open) you must call this method when you want the score to be graphically updated.
Parameters : None
Return value : None

Application.UpdateAllWindows(...)

Description : Redraw all palettes after loading again the current icon set. Redraw all documents windows.
Parameters : None
Return value : None

4.2.3.5. Floating Window

Application.IsFloatingWindowOpen(...)

Description : Search if a floating window (page 70) with a given script is already open. The name to search is the fi name of the floating window.
Parameters : String : the name to search
Return value : Boolean, true if the floating window is already open.

Application.Close(...)

Description : Close a custom floating window by name
Parameters : String : the script file name which manage the floating window. This name was used by Run("xxx.myf')
Return value : Boolean : true if the script has been launched.

4.2.3.6. Getting Virtual Singer language info

Application.GetVSLangDetails(...)

Description : Fills a table with the details for a language code
The table fields will be contain:
- Name: printable name of the language
- Credits: Credits, copyright
- Comments: Additional comments about this language
- Flag: Surface containing the language flag
<< Not implemented yet
- Charset: list of all the available characters for this language
>>
Parameters : language code (3letters-1number)
Return value : Table containing the language details
Example :

```
t=Application.GetVSLangList()
for i=1,getn(t) do
    t2=Application.GetVSLangDetails(t)
    print("------")
    print("Code:       "..t)
    print("Name:       "..t2.Name)
    print("Credits:    "..t2.Credits)
    print("Comment: "..t2.Comment)
end
```

4.2.3.7. Getting Virtual Singer language list

Application.GetVSLangList(...)

Description : Fills a table with the language code of all available languages for Virtual Singer
Each element of the table will contain the 3letters-1number code for the language
Parameters : None
Return value : Table containing the language codes (1-based)

4.2.3.8. Human Voice

MyrScript will use the system resources to synthesize human voices.
This feature is not available on all systems and you should use Application.GetCapability("CanSpeak") to check this.

Application.Say(...)

Description : Use the system to say the sentence. This operation can be asynchronous.
You should use Application.IsSaying() to check the system is not currently speaking.
Sending a new sentence when one is curently playing stop it.
Sending an empty string stop the voice.
Parameters : String.The sentence to say with the current voice.
Boolean : True if myrscript wait the end of the sentence. The user can break this with esc key. False by default (optional)
Return value : Error. -1 if the user breaked the command.
Example :

```
Application.SetHumanVoice(Application.SearchHumanVoice("","en"))
Application.Say("To be, or not to be, that is the question.")
print("Waiting end of sentence")
while Application.IsSaying()==true do end
print("Done")
```

```
Application.SetHumanVoice(Application.SearchHumanVoice("","en"))
if(Application.Say("To be, or not to be, that is the question.",true)<0) then
    print("Breaked by user")
else
    print("Done")
end
```

Application.IsSaying(...)

Description : Tell if the system is currently saying a sentence.
Return value : Boolean : true if the system is current saying something.

Application.SayOnFile(...)

Description : Use the system to say the sentence on a file.
Parameters :
String : The complete file name to generate
String :The string to say.
Return value : Number: Error
Example :

```
-- Asks the user for selecting a file name for saving
theFileName=GetFileNameSave("SayOnFile","brt")
if theFileName ~=nil then
   Application.SayOnFile(theFileName,"Hello world.")
end
```

Application.SetHumanVoice(...)

Description : The voice will be the current voice
Parameters : Human voice index from 1 to Application.GetNumberOfHumanVoices
Return value : Number: Error
Note : You can use Application.humanVoice (page 179) to got the current human voice index.
Example :

```
Application.SetHumanVoice(Application.SearchHumanVoice("","en"))
Application.SetHumanVoicePitch(40)
Application.SetHumanVoiceRate(175)
Application.Say("How are you dear ?")
```

Application.SetHumanVoiceDefault(...)

Description : Reset all default values for human voice : voice number, rate, pitch, etc.
Parameters : None
Return value : None

Application.SetHumanVoiceRelativePitch(...)

Description : Sets the voice pitch relatively to its default.
From -100 (low-pitched) to 100 (hi-pitched)
Parameters : Float, the relative pitch to set.
Return value : Number: Error

Application.SetHumanVoicePitch(...)

Description : The pitch can vary from 90 a male, to 300 a children
a negative value resets to default
Parameters : Float, the pitch to set.
Return value : Number: Error

Application.GetHumanVoicePitch(...)

Description : The current pitch
Parameters : None
Return value : Number: Error

Application.SetHumanVoicePitchModulation(...)

Description : A pitch modulation value of 0.000 corresponds to a monotone in which all speech is generated at the frequency corresponding to the speech pitch
Parameters : Float, the pitch to set.
Return value : Number: Error

Application.GetHumanVoicePitchModulation(...)

Description : The current pitch
Parameters : None
Return value : Number: Error

Application.SetHumanVoiceRate(...)

Description : The rate is the number of word per minute.
Speaking rates range from around 150 words per minute to around 220 words per minute.
a negative value resets to default
Parameters : Float, the rate to set.
Return value : Number: Error

Application.GetHumanVoiceRate(...)

Description : The current rate
Parameters : None
Return value : Number: Error

Application.SetHumanVoiceVolume(...)

Description : From 0 (silent) to 1(max)
a negative value resets to default
Parameters : Float, the volume to set.
Return value : Number: Error

Application.GetHumanVoiceVolume(...)

Description : The current volume from 0 to 1.
Parameters : None
Return value : Number: Error

Application.GetNumberOfHumanVoices(...)

Description : Provide the number of human voices available.
Parameters : None
Return value : Number: Number of human voices
Example :

```
if Application.GetCapability("CanSpeak")=="Yes" then
   print("There is "..Application.GetNumberOfHumanVoices().." voices availables")
else
   print("No human voices on this system")
end
```

Application.GetHumanVoiceInformation(...)

Description : Provide information about the human voice.
Parameters : Number (Optional): Human voice index from 1 to Application.GetNumberOfHumanVoices
If no parameter, provide information about current human voice
Return value : String : voice name
Number : gender (0 for unknown, 1 for male, 2 for female)
Number : age (years)
Number : default putch (Hz)
Number : language (constant as 0=French, 2=Italian, 3=German, 4=Spanish, 5=Japanese, 6=Portuguese, 7=Dutch,
1=English (or other)
String : language name (2-letters code e.g. EN, FR ...)
String : region name (2-letters code e.g. US, AU, QC...)
String : Demo string for this language
Example :

```
for i=1,Application.GetNumberOfHumanVoices() do
   print(Application.GetHumanVoiceInformation(i))
end
```

Application.SearchHumanVoice(...)

Description : Search an human voice by name, language, gender, etc.
Parameters :
String : Desired voice name or empty string if none. This can be a complete name like "Agnes" or a search pattern (as a file search). For example "a*" will search all names
beginning with "a". "?l*" will search all names with a "L" at second place. You can combine this with the search index to extract all names matching (see following sample)
String (optional) : Desired language : "en", "fr" ... or empty string if any languages
String (optional) : Desired region or empty string if any regions
String or Number (optional) : Desired gender Empty string or -1 for any, "M" or 1 for male, "F" or 2 for female, "N" or 0 for neutral (by default -1)
Number (optional) : Minimal age (-1 for any)
Number (optional) : Maximal age (-1 for any)
Number (optional) : Index of voice in all voices matching (by default 1)
Return value : Number : index of the voice if found,0 if not found
Example :

```
print("Search voice by language and say the name of the voice")
print("Extracting first english voice")
num=Application.SearchHumanVoice("","en")
if num>0 then
   Application.SetHumanVoice(num)
   name=Application.GetHumanVoiceInformation(num)
   print("The first english voice is : "..name)
   Application.Say("Hi, I am "..name,true)
else
   print("No voice for english")
end
print("Extracting first british english voice")
num=Application.SearchHumanVoice("","en","gb")
if num>0 then
   Application.SetHumanVoice(num)
   name=Application.GetHumanVoiceInformation(num)
   print("The first british english voice is : "..name)
   Application.Say("Hello, I am english. My name is"..name,true)
else
   print("No voice for british english")
end
print("Extracting first french male voice")
num=Application.SearchHumanVoice("","fr","","M")
if num>0 then
   Application.SetHumanVoice(num)
   name=Application.GetHumanVoiceInformation(num)
   print("The first french male voice is : "..name)
   Application.Say("Bonjour je suis "..name,true)
else
   print("No voice for french male")
end
---
```

```
print("Extracting  male voices, in all languages")
num=1
voice=1
while voice>0 do
   voice=Application.SearchHumanVoice("","","","M",-1,-1,num)
   if voice>0 then
      Application.SetHumanVoice(voice)
      name=Application.GetHumanVoiceInformation(voice)
      print("Name :"..name)
      num=num+1
   end
end

---
```

```
print("Searching for a voice named Agnes")
local voice
voice=Application.SearchHumanVoice("Agnes")
if voice>0 then
   Application.SetHumanVoice(voice)
   name=Application.GetHumanVoiceInformation(voice)
   Application.Say("You find me, I am "..name)
else
   Application.Say("Sorry I can't found this voice name")

end
```

```
print("Search voices by pattern : all voices beginning by a A")
num=1
voice=1
while voice>0 do
   voice=Application.SearchHumanVoice("a*","","","",-1,-1,num)
   if voice>0 then
      name=Application.GetHumanVoiceInformation(voice)
      print("Name :"..name)
      num=num+1
   end
end
```

```
print("Extracting  english voices which are not american english")
num=1
voice=1
while voice>0 do
   voice=Application.SearchHumanVoice("","en","","",-1,-1,num)
   if voice>0 then
      Application.SetHumanVoice(voice)
      name,void,void,void,void,region,demo=Application.GetHumanVoiceInformation(voice)
      if region~="US" then
         print("Name :"..name..".."  "..region.."  "..demo)
         Application.Say(demo,true)
      end
      num=num+1
   end
end
```

```
print("Extracting kids voices")
num=1
voice=1
while voice>0 do
   voice=Application.SearchHumanVoice("","","","",0,17,num)
   if voice>0 then
      Application.SetHumanVoice(voice)
      name,void,age,void,void,void,demo=Application.GetHumanVoiceInformation(voice)
      print("Name :"..name..".." is "..age.."  "..demo)
      Application.Say(demo,true)
      num=num+1
   end
end
```

4.2.3.9. Language

Application.DumpKeyWordList(...)

Description : Dumps all known MyrScript language keywords.
Parameters : None
Return value : None
Example :

```
Application.DumpKeyWordList()
```

4.2.3.10. Log Window

Application.ClearLog(...)

Description : Clear the log window
Parameters : None
Return value : None

Application.SaveLog(...)

Description : Save the text displayed on the log window in a text file.
If the window is empty, nothing is saved.
Parameters : Full path on file
Return value : True if saved

4.2.3.11. Manual

Application.OpenManual(...)

Description : Open one page of the application manual in the current language
Parameters : String : the page name
Return value : None

4.2.3.12. MyrWeb

Application.WebExportSetup(...)

Description : Opens the setup dialog box for MyrWeb export. The values are automatically recorded in a preference file.
Parameters : (optional) true if enhanced setup (by default false)
Return value : Boolean : true if user validated the box.
Boolean : true if user changed the setup.
Related methods and values:
Application.GetWebExportSetup(...) (below)
Score.WebExport(...) (page 230)
Staff.MyrWebGroup (page 268)

Application.GetWebExportSetup(...)

Description : Provide the parameters describing the MyrWeb export setup as defined in WebExportSetup.
Parameters : none
Return value :
· String : Advanced setup, all settings separated by ';',for example : playmarker=bar;volume=80;
· Boolean : true if the score will be displayed on the web page (by default the last value defined in WebExportSteup, by default true)
· Boolean : true if all views will be included, false only main view (by default the last value defined in WebExportSteup, by default true)
· Boolean : true if lyrics could be displayed in a separate page (as a karaoke) (by default the last value defined in WebExportSteup, by default true)
· Boolean : true if music could be played (by default the last value defined in WebExportSteup, by default true)
· Boolean : true if piano keyboard could be displayed (by default the last value defined in WebExportSteup, by default true)
· Boolean : true if metronome could be played (by default the last value defined in WebExportSteup, by default true)
· Number : music quality from 0 (low quality, shortest file) to 4 (by default the last value defined in WebExportSteup, high quality , biggest file) (by default 2)
· Boolean : true if myr file will be embedded (by default the last value defined in WebExportSteup, by default true)
· Boolean : true if data will be compressed (by default the last value defined in WebExportSteup, by default true)

· Number : automatic group of staves. On the web page the user can select which group is played. (by default the last value defined in WebExportSteup, by default 0)
0 = No automatic group, use Staff.MyrWebGroup (page 268) to group the staves in a same audio track.
1 = First staff in a track, all others in another track.
2 = All staves with lyrics in a track, all others staves in another track.
3 = Each staves with lyrics in a separate track, all others staves in another track.
4 = Each staves in a separate track.

4.2.3.13. Reserved

Application.GenerateManual(...)

Description : For internal use only. Generates a full MyrScript manual from developer source files.
Parameters : String : Password
String :Chapter name to generate. This can be an object name like "Score" or "Midi", a source file, an HTLM file name, or nil to generate a full MyrScript manual.
Boolean : Generates HTML (false by default)
Boolean : Generates Index (false by default)
Boolean : Display log messages (false by default)
Boolean : Save result in Myrscript docs folder (false by default, false if the chapter name is not nil)
Boolean : Compact manual on disk (true by default)
Return value : None

Application.SwitchKooplet(...)

Description : For internal use only.
Return value : None

Application.IsKoopletMode(...)

Description : For internal use only.
Return value : None

Application.UpdateScriptMenu(...)

Description : For internal use only. Refresh the script menus
Parameters : None
Return value : None

Application.GenerateWebFolder(...)

Description : For internal use only. Generates a complete "installscript" folder.
Parameters : None
Return value : None

4.2.3.14. Settings

This section enables to configure some application behaviours.

Application.ScriptBackGround(...)

Description : Specifies whether the script will run in background or not. By default, a script is not running in background. When a script is running in background, user can open menu, open files, save files etc. while the script is running. Be aware, if the script uses objects got from a score and this score is closed by the user, a crash can occur.
For example, this kind of script can be used to get data from MIDI input and apply operations on it.
As the user can act on all the application features, he can launch a script while a script is already running in background. In this case, the background script will "sleep" until the new script is terminated. All variables and data are not visible from a script to another.
Parameters : True if the script has to run in background, false if not.
Return value : None
Example :

```
--- This sample initiates Midi in and displays Note on and Note off events until
--- the user presses ESC key.
--- Please note the use of Application.ScriptBackGround(true) which enables
--- this script to run in background
Include "MSDefine"
Application.ScriptBackGround(true)
print("Opening MIDI input device")
if Midi.InBegin()==true then
   print("Press the ESC Key to Quit")
   repeat
      event,channel,pitch,velocity=Midi.GetEvent()
      if event>0 then
         if(event==1) then print("Note On "..pitch)
         else  print("Note Off "..pitch) end
      end
      until Application.KeyStroke()==27

   Midi.InEnd()
else
   print("Cannot open Midi in device, please check your Midi setup")
   end
print("End")
```

Note : Before defining a background script, look first if you can't use a floating window instead, this is usually more efficient...

Application.GetCapability(...)

Description : Test the capability of the system and return its value. The value type depends of the tested capability.
Parameters : String : The capability name
Return value : String : the capability result, or "Unknown" if the capabilty name is unknown.

Name	Returned values	Description
"MediaCanManageFiles"	"Yes" or "No"	Tells whether the system can manage media files.
"MediaCanManageImages"	"Yes" or "No"	Tells whether the system can manage images in media files.
"MediaCanManageSounds"	"Yes" or "No"	Tells whether the system can manage sounds in media files.
"MediaCanManageTexts"	"Yes" or "No"	Tells whether the system can manage texts in media files.
"MediaCanManageEffects"	"Yes" or "No"	Tells whether the system can manage real time effects in media files.
"MediaFileSaveExtension"	A string	Extension for media files when saving.
"MediaFileLoadExtension"	A string	Extension(s) for media files when loading.
"CanSpeak"	"Yes" or "No"	Tells whether the system can say sentences with an human voice.

4.2.3.15. Sound Base

Application.GetSoundName(...)

Description : Provide the sound name and the set name of a given sound in the currently selected language
Parameters : The standard sound ID
Return value : Strings : the sound name, the set name. If the ID does not exist, returns nil.

Application.IsSoundAlias(...)

Description : If the sound is an alias (not a real sound in the sound base but an alias on an existing sound) return true.
Parameters : The standard sound ID
Return value : Boolean : true if the sound is an alias,false if the sound is a real sound, If the ID does not exist, return nil.

4.3. Notation

This section describe the score object type and the object types related to it.

4.3.1. Score Object Type

General description of this object type : A Score object represents a music document.
You will be able to add staves, and manage all score-related items (chord grid, accompaniments...)

Note about protected scores

If the score is protected by its author, most of the values can be read but not written. Many operations on the score are disabled.
So, the developer must check first whether the score is protected using :

- Score.IsProtectedAgainstEditing (page 203)
- Score.IsProtectedAgainstPrinting (page 203)
- Score.IsProtectedAgainstGraphicalExport (page 203)
- Score.IsProtectedAgainstExport (page 203)
- Score.IsProtectedAgainstSaving (page 203)

If the protection is active the script must skip this score or ask the user for the password.

4.3.1.2. Score global functions

4.3.1.2.1. Creating

These functions enable to create new scores.
You can also create a score from a pre-defined template.

NewScore(...)

Description : Create a new empty score.
Parameters : Optional : a boolean value indicating whether the score is visible or not.
If you create an invisible score, you can make it become visible by using Score.ShowWindow(...) (page 250).
A boolean value indicating if the default metronome instruments will be added (true by default)
Return value : A new score or nil if the function failed.
Example :

```
-- Create an empty score
-- and add one staff to it
local myScore
local myStaff
myScore=NewScore()
if(myScore ~= nil) then
   myScore.AddStaff()
end
```

```
--Creating a new invisible score
local myScore
myScore=NewScore(false)
if(myScore ~= nil) then
   -- Add a staff
   myScore.AddStaff()
   -- Show it
   myScore.ShowWindow()
   -- Update it
   Application.UpdateScores()
   -- Delay
   Application.WaitDelay(120)
   -- Hide it
   myScore.HideWindow()
   --Dispose it without confirmation
   myScore.Dispose(false)
end
```

```
--- This sample create a MyrScript source file
--- Which will create a score, playing the Jingle you selected
-- on the front score
score=FrontScore()
if(score) then
   staff=score.FirstSelectedStaff
   symbol=staff.FirstSelectedSymbol
   symbolStop=staff.LastSelectedSymbol
   print("score=NewScore()")
   print("score.SetGeneralTempo("..score.GetGeneralTempo()..")")
   print("staff=score.AddStaff()")
   print("posTime=0")
   while symbol  do
      if(symbol.IsRest==true) then
         print("staff.InsertRest(posTime,"..symbol.Duration..")")
      else
         print("staff.InsertNote(posTime,"..symbol.Duration..","..symbol.Pitch..")")
      end
      print("posTime=posTime+"..symbol.Duration)
      if(symbol==symbolStop) then symbol=nil
      else symbol=symbol.Next
      end
   end
   print("score.PlayMusic()")
end
```

NewScoreFromTemplate(...)

Description : Ask the user to select a new score in the template window and create a new score from this templare.
Parameters : None
Return value : A new score or nil if the function failed.
Example :

```
local myScore
myScore=NewScoreFromTemplate()
```

4.3.1.2.2. Managing

FrontScore(...)

Description : Current front score. This is the current active score. This enables to start browsing all open scores.

If a score is provided, this score becomes the frontmost score and the previous front score is returned.
Parameters : Optional : score to bring to front.
Return value : A score or nil if there is no score.
Example :

```
local myScore
myScore=FrontScore()
if(myScore ~= nil) then
   Message("The front score file name is : "..myScore.FileName)
else
   Message("No score loaded.");
end
```

GetAlreadyOpenedScore(...)

Description : Provide the score already opened
Parameters : The complete file name of the score
Return value : A score or nil if the score is not opened

4.3.1.3. Score values

4.3.1.3.1. Ambitus

The ambitus is the range of note pitches for a staff or a multiple merged staves.
Ambitus is displayed before the clef as two quarter notes without stem related by a line.
Ambitus must be allowed for the score using Score.IsAmbitusEnabled (page 211). This value is false by default.
Ambitus must be allowed for the each staff or merged staves using Staff.IsAmbitusEnabled (page 268). This value is tr
by default.

Score.AmbitusLineStyle

Type : Number
Description : Ambitus line appearance. Line, wavy, crochet...

Score.AmbitusLineThickness

Type : Number
Description : Ambitus line thickness.

Score.AmbitusHorizontalSize

Type : Number
Description : Ambitus horizontal size. Size of the area before the clef.

Score.AmbitusHorizontalOffset

Type : Number
Description : Ambitus horizontal offset. Offset between the two quarters. 0 mean vertical aligned.

4.3.1.3.2. Browsing scores

Score.Previous

Type : Score
Description : Provides the previous score. If there is no previous score, the value is nil.
Note : You can read this value even if the score is protected by its author.

4.3.1.3.3. Chord grid computation

Score.IsChordGridComputedOutsideModalArray

Type : Boolean
Description : Status of the chord grid computation : follow or not the modal array. Changing this value will recompute the chord grid.

4.3.1.3.4. Display Setup

Score.IsRulerDisplayed

Type : Boolean
Description : Status of the ruler display. Tell you if the ruler is displayed.Changing this value will activate the display

Score.IsChordGridDisplayed

Type : Boolean
Description : Status of the chord grid display.Tell you if the chord grid is displayed.Changing this value will activate the display

Score.IsChordGridDisplayDiagrams

Type : Boolean
Description : Status of the chord grid display : display or not the guitar diagrams.

Score.NumberOfChordDiagrams

Type : Number
Description : Number of custom chord diagrams related to the chord grid

Score.ChordGridDiagramNeck

Type : Number
Description : Kind of fretted instrument used to draw chord diagrams..

Score.IsChordGridLeftAligned

Type : Boolean
Description : Status of the chord grid display : chord texts will be left aligned if true.

Score.IsRythmMatrixDisplayed

Type : Boolean
Description : Status of the rythm matrix display.Tell you if the rythm matrix is displayed.Changing this value will activate the display

Score.IsAccompaniementsMatrixDisplayed

Type : Boolean
Description : Status of the accompaniment matrix display. Tells whether the accompaniment matrix is displayed. Changing this value will activate the display

Score.IsColoredMultiVoices

Type : Boolean
Description : True if multivoices are displayed in color.

Score.TransposedMode

Type : Boolean
Description : True if the score is displayed in transposed mode, false if displayed in Concert Key mode.

Score.CurrentDisplayDpi

Type : Number
Description : Current display precision in dpi. You can change this value to modify the display scale factor. Please note, Scroll Mode and Page Mode have different display dpi.

Score.IsDisplayControlHandles

Type : Boolean
Description : True if the control handles are displayed. Only meaningful in page mode, control handles being always displayed otherwise
(see Score.GetPageMode(...) (page 251) and Score.SetPageMode(...) (page 251)).

4.3.1.3.5. Engraver

Score.EngraverSetup

Type : EngraverSetup (page 403)
Description : This is the engraver setup for displaying the score.

4.3.1.3.6. FreeObjects

Score.NumberOfFreeObject

Type : Number
Description : Number of free objects in this score

Score.FreeObjects

Type : FreeObject
Description : FreeObject (page 391) objects in the score. Valid even if there is no FreeObject object. Must be used as an array, by providing an index.

Score.FirstFreeObject

Type : FreeObject
Description : First FreeObject (page 391) object in the score

Score.LastFreeObject

Type : FreeObject
Description : Last FreeObject (page 391) object in the score

4.3.1.3.7. Information

Score.StorageArea

Type : StorageArea
Description : StorageArea object for this score

Score.NumberOfBars

Type : Number
Description : Number of bars in this score

Score.NumberOfPlayedBars

Type : Number
Description : Number of played bars in this score

Score.Title

Type : Comment (page 413) object type
Description : The title of the score. This is a styled text. This string is displayed when information are asked on the Score, for example with the Score.DisplayComments method. The string is printed on the first page of the document is is setting is enabled in the PageSetup (page 406).

Score.Composer

Type : Comment (page 413) object type
Description : The composer of the score. This is a styled text. This string is displayed when information are asked on the Score, for example with the Score.DisplayComments method. The string is printed on the first page of the document is is setting is enabled in the PageSetup (page 406).

Score.Remarks

Type : Comment (page 413) object type
Description : The remarks on the score. This is a styled text. This string is displayed when information are asked on the Score, for example with the Score.DisplayComments method. The string is printed on the first page of the document is is setting is enabled in the PageSetup (page 406).

Score.IsProtectedAgainstEditing

Type : Boolean
Description : True if the docment is protected by its author against editing. A script can't change values on a score when the score is protected. Many data can't be read. The script can ask the user to input the password using the Score.AskPassword(...) (page 240) method.

Score.IsProtectedAgainstPrinting

Type : Boolean
Description : True if the docment is protected by its author against printing. A script can't print a score when the score is protected. The script can ask the user to input the password using the Score.AskPassword(...) (page 240 method.

Score.IsProtectedAgainstSaving

Type : Boolean
Description : True if the docment is protected by its author againts saving. A script can't save a score when the score is protected. The script can ask the user to input the password using the Score.AskPassword(...) (page 240 method.

Score.IsProtectedAgainstGraphicalExport

Type : Boolean
Description : True if the document is protected by its author against graphical export. A script cannot export graphically a score when the score is protected. The script can ask the user to input the password sing the Score.AskPassword(...) (page 240) method.

Score.IsProtectedAgainstExport

Type : Boolean
Description : True if the document is protected by its author against digital export on file. A script can't export a score when the score is protected. The script can ask the user to input the password using the Score.AskPassword(...) (page 240) method.

Score.FileName

Type : String
Description : Score file name.

Score.PathName

Type : String
Description : Path on the folder where the score is. The path is ended by the '/' character.

Score.IsChanged

Type : Boolean
Description : true if the score was changed and need to be saved.
Example :

```
-- This script revert the front score to the saved version on disk.
local score
local completeName
-- Getting front score
score=FrontScore()
---Is a score opened ?
if(score~=nil) then
   if(score.IsChanged==true and
      Confirmation("The score '"..
            score.FileName..
            "' will be loaded from the last version on disk and all changes will be lost.")==true)
            then
      -- We tag the score as "not changed"
      score.IsChanged=false
      -- We record the complete file and path name
      completeName=score.PathName..score.FileName
      -- We close the score
      if(score.Dispose()==true) then
         Load(completeName)
      end
   else
      Message("Revert to saved : The front score is not changed.")
   end
else
   Message("Revert to saved : There is no front score.")
end
```

Score.AuthorWebPage

Type : String
Description : Web page of the author.This string is displayed when information are asked on the Score, for example with the Score.DisplayComments(...) (page 234) method.

Score.AuthorEmail

Type : String
Description : Email address of the author. This string is displayed when information are asked on the Score, for example with the Score.DisplayComments(...) (page 234) method.

Score.IsDisplayComments

Type : Boolean
Description : True if comments are automatically displayed when the score is loaded.

Score.CreationDate

Type : String
Description : Date of the score creation. The format is month/day/year hour:minute:seconds.

Score.ModificationDate

Type : String
Description : Date of the last score modification. The format is month/day/year hour:minute:seconds.

Score.TimeSpentInSeconds

Type : Number
Description : Number of seconds spent on this document.

4.3.1.3.8. Managing

Score.Next

Type : Score
Description : Provide the next document. This is the document which window is behind the current score.
Provides the next score. If there is no next score, the value is nil.
Parameters : Current score
Return value : a score or nil if no next score
Note : You can read this value even if the score is protected by its author.

4.3.1.3.9. Managing barlines and breaks

Score.NumberOfBarAction

Type : Number
Description : Number of custom barlines and break symbols in this score

Score.BarActions

Type : BarAction
Description : BarAction (page 322) objects in the score. Valid even if there is no BarAction object.
Must be used as an array, by providing an index.
Example :

```
-- Gets the 4th BarAction object in this staff
if myScore.NumberOfBarAction>=4 then myFourthBarAction=myScore.BarActions[4] end
```

Score.FirstBarAction

Type : BarAction
Description : First BarAction (page 322) object in the score

Score.LastBarAction

Type : BarAction
Description : Last BarAction (page 322) object in the score

4.3.1.3.10. Managing Instruments

None of these fields can be accessed if the score is protected against editing.

Score.NumberOfInstruments

Type : Number
Description : Number of instruments in this score
Example :

```
local myScore
local validNumber
-- Extracting score
myScore=FrontScore()
if(myScore~=nil) then
   -- Tells how many valid instruments are included in this score
   validNumber=0
   for index=1,myScore.NumberOfInstruments do
      if myScore.Instruments[index].IsUsed==true then validNumber=validNumber+1 end
   end
   Message("Number of valid instruments in ",myScore.Title.String,":",validNumber)
else
   Message("Please load a score")
end
```

Note : there can be "empty holes" in the instrument list.
When an instrument is deleted, its index can remain valid,but its "IsUsed" field becomes false.
When a new instrument is added, it searches for the first unused place in the instrument list and uses it.
So when browsing instruments, these "empty holes" have to be skipped
Due to these restrictions, it is recommended to use Score.FirstInstrument (page 206), Score.LastInstrument (pag
Instrument.Next (page 429) and Instrument.Previous (page 429)

Score.Instruments

Type : Instrument
Description : Instrument array for this score. Valid even if there are no instruments.
Must be used as an array, by providing an index from 1 (the first instrument) to Score.NumberOfInstruments
Example :

```
local myScore
-- Extracting score
myScore=FrontScore()
if(myScore~=nil) then
   -- Displays the name of the 4th instrument in this score
   if myScore.NumberOfInstruments>=4 then
      --- Extracting instrument #4
      instrument=myScore.Instruments[4];
      if instrument.IsUsed==true then
         Message("4th instrument is",instrument.Name)
      else
         Message("This instrument is unused")
      end
   else
      Message("There is no 4th instrument in this score")
   end
else
   Message("Please load a score")
end
```

Note : there can be "empty holes" in the instrument list.
When an instrument is deleted, its index can remain valid,but its "IsUsed" field becomes false.
When a new instrument is added, it searches for the first unused place in the instrument list and uses it.
So when browsing instruments, these "empty holes" have to be skipped
Due to these restrictions, it is recommended to use Score.FirstInstrument (below), Score.LastInstrument (below), Instrument.Next (page 429) and Instrument.Previous (page 429)

Score.FirstInstrument

Type : Instrument
Description : First instrument in score.
Example :

```
Include "MSSounds"
-- Displays name of all active instruments of the front score
local myScore
-- Extracting score
myScore=FrontScore()
if(myScore~=nil) then
   instrument=myScore.FirstInstrument
   while instrument~=nil do
      print("Instrument Name : ",instrument.Name," is related to sound : ",
          SoundArray[instrument.SoundID])
      instrument=instrument.Next
   end
else
   Message("Please load a score")

end
```

Note : Only valid instruments are taken into account. Empty "holes" in the Instrument array are skipped.

Score.LastInstrument

Type : Instrument
Description : Last instrument in score.
Example :

```
-- Displays name of all active instruments in a score, in reverse order
local myScore
-- Extracting score
myScore=FrontScore()
if(myScore~=nil) then
   instrument=myScore.LastInstrument
   while instrument~=nil do
      print(instrument.Name)
      instrument=instrument.Previous
   end
else
   Message("Please load a score")
end
```

Note : Only valid instruments are taken into account. Empty holes in the Instrument array are skipped.

4.3.1.3.11. Managing Metronome

Changing the metronome values when the score is playing is taken into account immediatly.

Score.IsMetronomeActive

Type : Boolean
Description : Specify if the metronome is active. Changing this value whill update the "Play" palette.
Example :

```
--- Activating the metronome of the front score
score=FrontScore()
if(score~=nil) then
   score.IsMetronomeActive=true
   Message("The metronome is now active")
else
   Message("Please load a score.");
end
```

Score.MetronomeTickAtQuarterNote

Type : Boolean
Description : True if the metronome will play a note each quarter note, false if the metronome will play each bar.

Score.MetronomePitchForBarTick

Type : Number
Description : Pitch of the note which will be played at the beginning of the bar.

Score.MetronomePitchForQuarterNoteTick

Type : Number
Description : Pitch of the note which will be played each quarter note in the bar.

Score.MetronomeInstrumentForBarTick

Type : Instrument
Description : Instrument used for playing the tick at the beginning of each bar.
Example :

```
-- Displaying the name of the instrument related to metronome bar tick.
score=FrontScore()
if(score~=nil) then
   instrument=score.MetronomeInstrumentForBarTick
   Message("Instrument related to Metronome Bar Tick is "..instrument.Name)
else
   Message("Please load a score.");
end
```

Score.MetronomeInstrumentForQuarterNoteTick

Type : Instrument
Description : Instrument used for playing the tick at each quarter note in the bar.

4.3.1.3.12. Managing staves in score

Score.NumberOfStaves

Type : number
Description : Provide the number of staves in the document.
Example :

```
--- Display the number of staves in the front score
local myScore
-- Extracting score
myScore=FrontScore()
if(myScore~=nil) then
    Message("Number of staves is :"..myScore.NumberOfStaves)
else
   Message("Please load a score")
end
```

Note : You can access this value even if the score is protected by its author.

Score.FirstStaff

Type : Staff
Description : Provides the first staff of the score. If there is no staff in the score, the value is nil.
Example :

```
-- Display the staff names of the front score
myScore=FrontScore()
if(myScore ~= nil) then
   myStaff=myScore.FirstStaff
   while myStaff~=nil do
      print (myStaff.Title)
      myStaff=myStaff.Next
   end
end
```

Note : You can't access this value if the score is protected by its author.

Score.LastStaff

Type : Staff
Description : Provides the last staff of the score. If there is no staff in the score, this value is nil.
Note : You cannot get this value if the score is protected by its author.

4.3.1.3.13. Page Setup

Score.PrintSetup

Type : PageSetup (page 406)
Description : This is the page setup for printing or displaying the score in page mode.

Score.ExportSetup

Type : PageSetup (page 406)
Description : This is the page setup for exporting the score in various graphical formats like BMP, PICT, EPS, etc.

Score.StreetOrganSetup

Type : PageSetup (page 406)
Description : This is the page setup for exporting the score in street organ cards.

Score.ChordGridSetup

Type : PageSetup (page 406)
Description : This is the page setup for printing or displaying the chord grid in page mode.
This page is added at the end of the score.

Score.TextSetup

Type : PageSetup (page 406)
Description : This is the page setup for printing or displaying the lyrics in page mode.
Each lyric line is related to a separate page which is added at the end of the score.

4.3.1.3.14. Paper size setup

You can define here how pages are displayed in page mode and ask or set the paper size and margins.
These values can be provided by your printer driver. You can change them by selecting another paper size from your printer or switching the score to user-defined paper size. By setting these values, you automatically switch score from "printer-defined paper size" to "user-defined paper size".

Score.IsKeepFirstPageAside

Type : Boolean
Description : Setup the page mode display to keep the first page of the document aside.

Score.IsRelatePageTwoByTwo

Type : Boolean
Description : Setup the page mode display to relate the page two by two.

Score.PaperMode

Type : number
Description : Provide you the current paper mode.
0 means the paper size will be provided by the printer driver
1 means the paper size is user defined and will fit the paper size when you will print, preserving the ratio horizontal and vertical
2 means the paper size is user defined and will completely fit the paper size when you will print
4 means the paper size is user defined and no ratio changes will be applied when printing.

When you change the value from "printer defined" to "user defined" the current paper size become the paper size defined by the printer.

Score.PaperWidth

Type : number in 10th of millimeters
Description : Current paper width. You can change this value to adjust the paper size.

Score.PaperHeight

Type : Number in 10th of millimeters
Description : Provide you the current paper height. You can change this value to adjust the paper size.

Score.PaperLeftMargin

Type : Number in 10th of millimeters
Description : Provide you the current paper left margin. You can change this value to adjust the usable area on the paper.

Score.PaperRightMargin

Type : Number in 10th of millimeters
Description : Provide you the current paper right margin. You can change this value to adjust the usable area on the paper.

Score.PaperTopMargin

Type : Number in 10th of millimeters
Description : Provide you the current paper top margin. You can change this value to adjust the usable area on the paper.

Score.PaperBottomMargin

Type : Number in 10th of millimeters
Description : Provide you the current paper bottom margin. You can change this value to adjust the usable area on the paper.

4.3.1.3.15. Played bars

Score.IsForcedPlayedBarsList

Type : Boolean
Description : Indicate if the played bar list will be computed along the break symbols or will be forced. See Score.SetPlayedBars(...) (page 237) and Score.GetPlayedBars(...) (page 237)

4.3.1.3.16. Playing Modes

Theses values enable to change the playing mode. Changing such a value while score is playing affects instantly the audio result.

Score.IsPlayAccompanimentsActive

Type : Boolean
Description : True if the accompaniments are played.

Score.IsPlayRythmsActive

Type : Boolean
Description : True if the rythms are played.

Score.IsPlaySelectionActive

Type : Boolean
Description : True if the selection is played.

Score.IsLoopPlayActive

Type : Boolean
Description : True if the music will restart at the beginning (or a the beginning of the selection if Score.IsPlaySelectionActive (above) is true).

Score.IsPlayViewActive

Type : Boolean
Description : True if the only the current view is played.

4.3.1.3.17. Playing music

Score.CurrentPlayedTime

Type : Time position
Description : Provide the current played time on the score. If the score is not currently played the value is -1.

Score.GeneralVolume

Type : Number
Description : The general score volume from 0 (no sound) to 127 (full volume).

Score.IsMusicPaused

Type : Boolean
Description : True if music is in pause mode. Use Score.PauseMusic to pause or unpause the music.

4.3.1.3.18. Printing

Score.NumberOfPrintedPages

Type : Number
Description : Number of printed pages in the document.

4.3.1.3.19. Score appearance

These values allow to adjust finely the score display setup, as the user can do with Score>Display setup

Score.DisplayGhostRests

Type : Boolean
Description : True if ghost rests are displayed while editing.(DEPRECATED, see [Staff.GhostRestsDisplayMode])

Score.AccidentalAppliesToAllOctaves

Type : Boolean
Description : True if accidental will apply to all octaves.

Score.JoinParts

Type : Boolean
Description : True if a line will join automatically the parts.

Score.ExtendParts

Type : Boolean
Description : True if a line will extended automatically the not joined parts.

Score.OldQuarterRest

Type : Boolean
Description : True if the old quarter rest notation is used.

Score.IsAmbitusEnabled

Type : Boolean
Description : True if the ambitus notation is used.

Score.HorizontalMirror

Type : Boolean
Description : True if the notation is inverted left-right : hebrew and arabic mode.

Score.OldAccidentalCancellation

Type : Boolean
Description : True if the XIX century notation is used: when the pitch change a natural is displayed.

Score.SpaceBetweenAccidentalAndNote

Type : Number
Description : Space between the accidental and the note head.

Score.SpaceBetweenAccidentalOnNotes

Type : Number
Description : When two accidentals of a chord overlap, extra space added to avoid this overlapping.

Score.MinimumVerticalSpacingBetweenAccidental

Type : Number
Description : When two accidentals of a chord are closest as this value they are considered as overlapping.

Score.SpaceBetweenDotAndNote

Type : Number
Description : Space between the dot and the note head symbol.

Score.SpaceBetweenDotAndRest

Type : Number
Description : Space between the dot and the rest symbol.

Score.SpaceBetweenDotAndDot

Type : Number
Description : When a symbol is double doted, space between the two dots.

Score.HorizontalDotAdjustementForUpstemFlag

Type : Number
Description : When a note stem have an upstem flag, extra space to avoid overlapping between dot and flag.

Score.VerticalDotOffset

Type : Number
Description : Vertical dot offset.

Score.DotScale

Type : Number
Description : Dot scale in percent (by default 100)

Score.BreakSymbolHeavyLineThickNess

Type : Number
Description : For bar line break symbol: thickness of heavy line.

Score.BreakSymbolThinLineThickNess

Type : Number
Description : For bar line break symbol: thickness of thin line.

Score.BreakSymbolSpaceBetweenDoubleBarlines

Type : Number
Description : For bar line break symbol: space between double bar lines.

Score.BreakSymbolSpaceBetweenHeavyLineAndThinLine

Type : Number
Description : For bar line break symbol: space between heavy line and thin line.

Score.BreakSymbolDashLineLength

Type : Number
Description : For bar line break symbol: length of the dash.

Score.BreakSymbolDashSpaceLength

Type : Number
Description : For bar line break symbol: space between dashes.

Score.BreakSymbolBracket

Type : Number
Description : For bar line break symbol: bracket on the bold line.

Score.SpaceBeforeClef

Type : Number
Description : Space between left edge of the bar and clef.

Score.SpaceBeforeKeySignature

Type : Number
Description : Space between clef and key signature.

Score.SpaceAfterKeySignature

Type : Number
Description : Space after key signature .

Score.SpaceBeforeTimeSignature

Type : Number
Description : Space before time signature.

Score.SpaceAfterTimeSignature

Type : Number
Description : Space after time signature.

Score.SpaceBetweenKeySignatureAccidentals

Type : Number
Description : Space between accidentals in key signature.

Score.SpaceAfterCancelledKey

Type : Number
Description : Space after natural symbols in key signature.

Score.GraceNoteSize

Type : Number
Description : Size of grace notes.

Score.StemLineThickNess

Type : Number
Description : Note stem thickness.

Score.StemOffsetLength

Type : Number
Description : Offset for default stem length.

Score.BeamSplitOnHalfBeat

Type : Number
Description : Beam split mode on half beat (0,1,2)

Score.TupletLineThickNess

Type : Number
Description : Tuplet bracket line thickness.

Score.SlurGraceThickNess

Type : Number
Description : Slur on graces thickness.

Score.GraceNoteGap

Type : Number
Description : Extra gap between grace notes.

Score.SpaceBeforeMultiRest

Type : Number
Description : Space before multi-rest symbol.

Score.VerticalOffsetMultiRest

Type : Number
Description : Multi-rest vertical offset.

Score.VerticalOffsetMultiRestNumber

Type : Number
Description : Multi-rest number vertical offset.

Score.MultiRestLineWidth

Type : Number
Description : Multi-rest line thickness.

Score.MultiRestEdgeLineHeight

Type : Number
Description : Multi-rest edge line height.

Score.MultiRestTextSize

Type : Number
Description : Multi-rest text size.

Score.SpaceBeforeMultiBar

Type : Number
Description : Space before multi-bar symbol.

Score.VerticalOffsetMultiBar

Type : Number
Description : Multi-bar vertical offset..

Score.VerticalOffsetMultiBarNumber

Type : Number
Description : Multi-bar number vertical offset.

Score.MultiBarTextSize

Type : Number
Description : Multi-bar text size.

Score.MultiBarSymbolScale

Type : Number
Description : Multi-bar symbol scale.

Score.StaffLineThickNess

Type : Number
Description : Staff line thickness.

Score.StaffEndingLineThickNess

Type : Number
Description : Staff ending line thickness.

Score.StaffRightLineThickNess

Type : Number
Description : Staff ending line thickness at the system right.

Score.LedgerLineThickNess

Type : Number
Description : Ledger line thickness.

Score.LyricDashLineThickNess

Type : Number
Description : Lyric dash line thickness.

Score.LyricUnderLineThickNess

Type : Number
Description : Lyric underline thickness (melisma).

Score.LyricDashLineLength

Type : Number
Description : Lyric dash line length.

Score.LyricSpaceBetweenDashLineAndLetter

Type : Number
Description : Space between dash and letter.

Score.LyricSpaceBetweenMelismaAndLetter

Type : Number
Description : Space between melisma and letter.

Score.LyricMinimalSpaceBetweenSyllable

Type : Number
Description : Minimal space between syllable. If the distance is lower, dash line is not draw.

Score.LyricMaximalSpaceBetweenDashLine

Type : Number
Description : Maximal space between dashes. If the distance is greater, several dashes are draw.

Score.StaffTabLineThickNess

Type : Number
Description : Tablature staff line thickness .

Score.TabAccordionPushPullLineThickNess

Type : Number
Description : Accordion tablature push/pull line thickness.

Score.TabHarmonicaTriangleLineThickNess

Type : Number
Description : Harmonica tablature triangle thickness.

Score.TabHarmonicaArrowLineThickNess

Type : Number
Description : Harmonica tablature line thickness.

Score.DownwardTieWidth

Type : Number
Description : For downward tie: width of the curve.

Score.DownwardTieTension

Type : Number
Description : For downward tie: tension of the curve in percent (0 to 100).

Score.DownwardTieDeltaXStart

Type : Number
Description : For downward tie: horizontal offset of the start from the note head.

Score.DownwardTieDeltaYStart

Type : Number
Description : For downward tie: vertical offset of the start from the note head.

Score.DownwardTieDeltaXEnd

Type : Number
Description : For downward tie: horizontal offset of the end from the note head..

Score.DownwardTieDeltaYEnd

Type : Number
Description : For downward tie: vertical offset of the end from the note head..

Score.DownwardTieDeltaXCurve

Type : Number
Description : For downward tie: horizontal offset of the medium point of the curve.

Score.DownwardTieDeltaYCurve

Type : Number
Description : For downward tie: vertical offset of the medium point of the curve.

Score.UpwardTieWidth

Type : Number
Description : For upward tie: width of the curve.

Score.UpwardTieTension

Type : Number
Description : For upward tie: tension of the curve.

Score.UpwardTieDeltaXStart

Type : Number
Description : For upward tie: horizontal offset of the start from the note head.

Score.UpwardTieDeltaYStart

Type : Number
Description : For upward tie: vertical offset of the start from the note head.

Score.UpwardTieDeltaXEnd

Type : Number
Description : For upward tie: horizontal offset of the end from the note head..

Score.UpwardTieDeltaYEnd

Type : Number
Description : For upward tie: vertical offset of the end from the note head..

Score.UpwardTieDeltaXCurve

Type : Number
Description : For upward tie: horizontal offset of the medium point of the curve.

Score.UpwardTieDeltaYCurve

Type : Number
Description : For upward tie: vertical offset of the medium point of the curve.

Score.SlurMinimalBend

Type : Number
Description : Slur minimal bend for automatic computing (0-100)

Score.SlurMaximalBend

Type : Number
Description : Slur maximal bend for automatic computing (0-100)

Score.SlurBendModerator

Type : Number
Description : Slur moderating coefficient for bend automatic computing (0-100).
This value enables to restrict the bending of int32_t slurs more that small ones.

Score.EffectWidth

Type : Number
Description : Effect line thickness.

Score.StaccatoWidth

Type : Number
Description : Staccato dot width.

Score.ChordDiagramLineThickNess

Type : Number
Description : Chord diagram line thickness.

Score.ChordDiagramNutSize

Type : Number
Description : Chord diagram nut size.

Score.ChordDiagramFingerTextScale

Type : Number
Description : Chord diagram finger number size.

Score.ChordDiagramFingerCircleScale

Type : Number
Description : Chord diagram finger circle scale.

Score.ChordDiagramFingerBarreScale

Type : Number
Description : Chord diagram finger barre scale.

Score.ChordDiagramBarreFretNumberTextScale

Type : Number
Description : Chord diagram fret number text scale.

Score.ChordDiagramNameTextScale

Type : Number
Description : Chord diagram name scale.

Score.ChordDiagramFontName

Type : String
Description : Font name of the font used to display chord.

Score.ChordDiagramFontFace

Type : String
Description : Face of the font used to display chord.

Score.ChordDiagramFontSize

Type : Number
Description : Size of the font used to display chord.

Score.ChordDiagramFontColor

Type : String
Description : Color of the font used to display chord (in RGB Component)

Score.ChordGridFontName

Type : String
Description : Font name of the font used to display chord grid.

Score.ChordGridFontFace

Type : String
Description : Face of the font used to display chord grid.

Score.ChordGridFontSize

Type : Number
Description : Size of the font used to display chord grid.

Score.ChordGridFontColor

Type : String
Description : Color of the font used to display chord (in RGB Component)

Score.TextFrameThickness

Type : Number
Description : Text frame thickness.

Score.TextFrameMargin

Type : Number
Description : Text frame margin.

Score.BraceWidth

Type : Number
Description : Brace symbol width.

Score.BraceCenterOffset

Type : Number
Description : Brace symbol center offset

Score.BracketWidth

Type : Number
Description : Bracket symbol width.

Score.BracketEdgeOffset

Type : Number
Description : Bracket symbol edge offset.

Score.HairPinWidth

Type : Number
Description : Harpin line width.

Score.DrumAsRect

Type : Number
Description : For drum grid: drums event are draw as round rect.

Score.DrumGridTriangleLineSize

Type : NumberNumber
Description : For drum grid: size of the triangle line.For drum grid: size of the triangle line.

Score.DrumGridTimeBarLineSize

Type : Number
Description : For drum grid: size of the line at time.

Score.DrumGridBarLineSize

Type : Number
Description : For drum grid: size of the line between each mesure.

Score.DrumGridCellBarLineSize

Type : Number
Description : For drum grid: size of the line not at time.

4.3.1.3.20. Selection

Score.IsSelected

Type : Boolean
Description : Return true if a selection is currently active on the score.

Score.IsSelectedIndividually

Type : Boolean
Description : Return true if a last one symbol is selected individually on the score.

Score.TimeBeginSelection

Type : Time position
Description : Provide the written time position of the beginning of the selection.
Note : You can access this value even if the score is protected by its author.

Score.TimeEndSelection

Type : Time position
Description : Provide the written time position of the end of the selection.
Note : You can access this value even if the score is protected by its author.

Score.FirstSelectedStaff

Type : Staff
Description : Provide the first selected staff of the score. If there is no staff, or no selected staff in the score, the value is nil.
Example :
```
myStaff=myScore.FirstSelectedStaff
```
Note : You can't access this value if the score is protected by its author.

Score.LastSelectedStaff

Type : Staff
Description : Provide the last selected staff of the score. If there is no staff, or no selected staff in the score, the value is nil.
Example :
```
myStaff=myScore.LastSelectedStaff
```
Note : You can't access this value if the score is protected by its author.

4.3.1.3.21. Staves group

Score.FirstStavesGroup

Type : StavesGroup
Description : Provides the first staves group of the score. If there is no staves group in the score, this value is nil.
Note : You cannot get this value if the score is protected by its author.

Score.StavesGroups

Type : StavesGroup
Description : Must be used as an array, by providing an index.

Score.NumberOfStavesGroups

Type : Number
Description : Provides the number of staves group in the score

4.3.1.3.22. Targets

Score.FirstTarget

Type : Target
Description : First Target (page 398) object in the score

Score.LastTarget

Type : Target
Description : Last Target (page 398) object in the score

Score.TargetCaptionsTextStyle

Type : TextStyle (page 414)
Description : Text style for targets text.

Score.AutomaticTargetCaptions

Type : Boolean
Description : If true, target captions are automatically computed.

Score.TargetCaptionsMode

Type : Number
Description : Define the way the target captions are computed. Allowed values are
1: The target name is used
2: Letters are used
3: Number are used
Applied only if Score.AutomaticTargetCaptions (above) is active.

Score.FrameTargetCaptions

Type : Boolean
Description : If true, a frame is draw around the targets captions.
Applied only if Score.AutomaticTargetCaptions (above) is active.

Score.TargetCaptionsHorizontalShift

Type : Number
Description : Target captions horizontal shift.
Applied only if Score.AutomaticTargetCaptions (above) is active.

Score.TargetCaptionsVerticalShift

Type : Number
Description : Target captions vertical shift.
Applied only if Score.AutomaticTargetCaptions (above) is active.

4.3.1.3.23. Views

Score.NumberOfViews

Type : Number
Description : Number of views in this score

Score.Views

Type : View
Description : View (page 400) objects in the score. The first element of this array is the general view.
Must be used as an array, by providing an index.
Example :

```
-- Display view names in ascending order
myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   for x=1,myScore.NumberOfViews do
      print ("View :"..myScore.Views[x].Name)
   end
end
```

Score.FirstView

Type : View
Description : First View (page 400) object in the score

Score.LastView

Type : View
Description : Last View (page 400) object in the score

4.3.1.3.24. Window appearance

Score.WindowWidth

Type : Number
Description : Width of the score window. This width doesn't include the margins of the window.Warning : Don't set a value which can be greater than the screen. Use the <ScreenSize> function to get the current screen dimension.
You can change this value even the document is protected by its author.
Range : from 0 to 10000 (in pixel)

Score.WindowHeight

Type : Number
Description : Height of the score window. This width doesn't include the margins of the window.

Score.WindowHorizontalPosition

Type : Number
Description : Horizontal position of the score window on the screen. This position doesn't include the margins of the window.

Score.WindowVerticalPosition

Type : Number
Description : Vertical position of the score window on the screen. This position doesn't include the margins of the window.

Score.WindowLeftMargin

Type : Number
Description : Provide the window left margin. You can't change this value. This margin is interface dependant and can change from an Operating System to another. You can use this value to move the window or resize it, taking account of the no-usable area.

Score.WindowRightMargin

Type : Number
Description : Provide the window right margin. You can't change this value. This margin is interface dependant and can change from an Operating System to another. You can use this value to move the window or resize it, taking account of the no-usable area.

Score.WindowTopMargin

Type : Number
Description : Provide the window top margin (the window Title height).You can't change this value. This margin is interface dependant and can change from an Operating System to another. You can use this value to move the window or resize it, taking account of the no-usable area.

Score.WindowBottomMargin

Type : Number
Description : Provide the window bottom margin.You can't change this value. This margin is interface dependant and can change from an Operating System to another. You can use this value to move the window or resize it, taking account of the no-usable area.

4.3.1.4. Score methods

4.3.1.4.1. Background picture

Score.RemoveBackGroundPicture(...)

Description : Remove the custom document background picture (if any) and set the standard document background.
Parameters : None
Return value : None
Example :

```
--- This sample remove the background of the front document
score=FrontScore()
if(score==nil) then
   Message("Please load a score")
else
   score.RemoveBackGroundPicture()
end
```

Score.SetBackGroundPicture(...)

Description : Set the custom document background picture. The entire content of the picture file is embedded in the score.
Parameters : String : The complete path name of the file
Return value : Boolean : True is the operation succeses.
Example :

```
--- This sample ask the user to select a PICT or a BMP file and set it
--- as background of the front document
score=FrontScore()
if(score==nil) then
   Message("Please load a score")
else
   myFileName=GetFileNameLoad("bmp,pct")
   if(myFileName~=nil) then
      score.SetBackGroundPicture(myFileName)
   end
end
```

4.3.1.4.2. Adding a BarAction object

Score.InsertBarAction(...)

Description : Inserts a BarAction object
Parameters : Number : Bar number
Number : BarAction kind. (See constants in MSDefine.mys)
Return value : Inserted BarAction object or nil if impossible to insert
If there is already a BarAction object of the same type at this bar, it is replaced by the new one.

The following BarAction objects are inserted at the **beginning** of the bar :

- INVISIBLE_BARLINE (can be moved to bar end)
- DOUBLE_BARLINE (can be moved to bar end)
- BOLD_BARLINE (can be moved to bar end)
- DOTTED_BARLINE (can be moved to bar end)
- PIECE_START_BARLINE
- REPEAT_START_BARLINE
- PART_START_BREAK_SYMBOL
- CODA_BREAK_SYMBOL
- SEGNO_BREAK_SYMBOL

The following BarAction objects are inserted at the **end** of the bar :

- PIECE_END_BARLINE
- REPEAT_END_BARLINE
- PART_END_BREAK_SYMBOL
- FINE_BREAK_SYMBOL
- AL_CODA_BREAK_SYMBOL
- DA_CAPO_BREAK_SYMBOL
- DA_SEGNO_BREAK_SYMBOL
- DA_CAPO_AL_FINE_BREAK_SYMBOL
- DA_SEGNO_AL_FINE_BREAK_SYMBOL
- DA_CAPO_AL_CODA_BREAK_SYMBOL
- DA_SEGNO_AL_CODA_BREAK_SYMBOL

Example :

```
Include "MSDefine"
-- Repeats bars 3 to 5 of the front score
myScore=FrontScore()
if myScore==nil then
   print("Please load a score first")
else
   --We preserve the current score state,
   --so the user will be able to undo the changes.
   myScore.Preserve()
   myScore.InsertBarAction(3,REPEAT_START_BARLINE)
   myScore.InsertBarAction(5,REPEAT_END_BARLINE)
end
```

4.3.1.4.3. Bar management

Score.DeleteBars(...)

Description : Delete bars.
Parameters : · Number : First bar to delete
Optional :
- Number of bar to delete (by default 1)
- Boolean : apply to selected staves (true by default)
- Boolean : apply to unselected staves (true by default)
- Boolean : apply to break symbols (true by default)
- Boolean : apply to accompaniements array (true by default)
- Boolean : apply to rythm array (true by default)
- Boolean : apply to chord grid (true by default)
- Boolean : apply to key changes (true by default)

Return value : True if deleted.
Example :

```
local myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   myScore.DeleteBars(1)
end
```

Score.InsertBars(...)

Description : Insert bars.
Parameters : · Number : Bar position where to insert.
Optional :
·Number of bar to insert (by default 1)
·Boolean : shift notes (true by default)
·Boolean : shift objects (true by default)
·Boolean : apply to selected staves (true by default)
·Boolean : apply to unselected staves (true by default)
·Boolean : apply to break symbols (true by default)
·Boolean : apply to accompaniements array (true by default)
·Boolean : apply to rythm array (true by default)
·Boolean : apply to chord grid (true by default)
·Boolean : apply to key changes (true by default)
Return value : True if inserted.
Example :

```
local myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   myScore.InsertBars(1)
end
```

Score.EditBarActions(...)

Description : Edit the bars.
Parameters : None
Return value : True if the user had validated the changes.

4.3.1.4.4. Beaming

Score.ComputeBeam(...)

Description : Compute the beaming of all notes in the score.
Parameters : Optional : true if stem directions will be computed too (false by default)
Return value : None

4.3.1.4.5. Chord grid

Score.EditChordGrid(...)

Description : Edit the chord grid.
Parameters : None
Return value : True if the user had validated the changes.

Score.GetBarChords(...)

Description : Gets an array of chords from the chord grid for a given bar.
Parameters : Bar number
Return value : Chord (page 386) object array
Number : Number of division of the grid (number of chord) from 1 to 4

Score.TimeToChord(...)

Description : Convert a time position in chord taken from the chord grid.
Parameters : Time position
Return value : Chord (page 386) object or nil if the time position is outside the score.
Number : Position in the bar from 1 to 4
Number : Number of division of the grid (number of chord) from 1 to 4
Example :

```
-- For a given time position provide the chord name,
-- the bar number, and the position in bar.
chord=score.TimeToChord(210600)
print(chord.Name.." "..chord.BarNumber.. " " ..chord.PositionInBar)
```

Score.AddChordDiagram(...)

Description : Adds a new local diagram object to the score grid
Parameters : Optional : Chord name (string)
Return value : Diagram (page 363) : object or nil if not added

Score.GetChordDiagramFromName(...)

Description : Search in the local chord diagrams of the grid the first chord diagram with this name
Parameters : Chord name (string)
Return value : Diagram (page 363) : object or nil if not found

Score.GetChordDiagramFromIndex(...)

Description : Provide the chord diagram from chord diagram database related to the score chord grid
Parameters : Number : index (1 based)
Return value : Diagram (page 363) : object or nil if not found
Example :

```
local score
local diagram

score=FrontScore()

for i=1,score.NumberOfChordDiagrams do
   diagram=score.GetChordDiagramFromIndex(i)
   print(diagram.ChordName.." "..diagram.StringValue)
end
```

Score.DeleteChordDiagrams(...)

Description : Delete all locals chord diagrams related to the score chord grid.
Parameters : None
Return value : None

Score.DuplicateChordDiagrams(...)

Description : Duplicate all local chords diagrams related to a chord grid to another score.
Local chords diagrams already related to target score are not affected.
You can use Score.DeleteChordDiagrams(...) (above) to clear the chords diagram.
Parameters : Target score
Return value : None

4.3.1.4.6. Converting

Score.TimeToBar(...)

Description : Convert a written time position into bar number
Parameters : the position to convert
Return value : The bar number or nil if the score object is wrong.
Example :

```
myBar=myScore.TimeToBar(myTime)
```

Note : If the time position exceed the total score duration additional bars are considered at the length of the last bar.
Use <Score.GetWrittenDuration> to get the total written music duration.
For example : lastBar=myScore.TimeToBar(myScore.GetWrittenDuration()-1)

Score.PlayedTimeToPlayedBar(...)

Description : Convert a played time position into played bar number
Parameters : the time position to convert
Return value : The bar number or nil if the score object is wrong.
Example :

```
myBar=myScore.PlayedTimeToPlayedBar(myPlayedTime)
```

Note : If the time position exceed the total score played duration additional bars are considered at the length of the last bar.
Use <Score.GetPlayedDuration> to get the total played music duration.
For example : lastPlayedBar=myScore.PlayedTimeToPlayedBar(myScore.GetPlayedDuration()-1)

Score.BarToTime(...)

Description : Convert a bar number from 1 tofirts 9999 into time position (in tick)
Parameters : the bar number
Return value : The time position of the beginning of the bar.

Score.PlayedBarToPlayedTime(...)

Description : Convert a played bar number from 1 to 9999 into its time position
Parameters : the played bar number
Return value : The played time position of the beginning of the bar.

Score.GetWrittenDuration(...)

Description : Returns the total written length of the score, in time position unit
Parameters : None
Return value : Total written length of the score, in time position unit

Score.GetPlayedDuration(...)

Description : Returns the total played length of the score, in time position unit
Parameters : None
Return value : Total played length of the score, in time position unit

Score.TimeToBeat(...)

Description : Convert a written time position in beat.
Parameters : the written time position to convert
Return value : Written beat number from the beginning of the staff. This value is not an integer. For example, 2.5 means two beats and a half from the beginning.
This value is 1 based.

Score.BeatToTime(...)

Description : Convert a beat in written time position (in tick)
Parameters : Floating number value. The beat from 1 to n.
Return value : The time position from the beginning of the staff

Score.ScoreDurationMilliSecond(...)

Description : Provide the total duration of the score in milli-seconds.
Parameters : None
Return value : The duration.

Score.PlayedTimeToWrittenTime(...)

Description : Converts a played time into written time position in the score. Played time differs from the written time position because of break symbols.
Parameters : Number: Played time
Return value : Number: written time position or -1 if this played time position is never reached
Number: number of times this written position has been played. 1 if it is the first time it is played, two if it is the second time...

Score.WrittenTimeToPlayedTime(...)

Description : Converts a written time in the score to played time position. Played time differs from the written time position because of break symbols.
Parameters : Number: Written time
(optional) Number: number of times this written position has been played. 1 if it is the first time it is played, two if it is the second time... Default = 1
Return value : Number: Played time position or -1 if this written time position and repeat number is never reached

Score.PlayedBarToWrittenBar(...)

Description : Converts a played bar number (from 1 to score.NumberOfPlayedBars) in written bar number (from 1 to score.NumberOfBars).
Parameters : Number: Played bar
Return value : Number: Written bar or 0 if this bar is never reached
Number: number of times this written bar has been played. 1 if it is the first time it is played, two if it is the second time...

Score.WrittenBarToPlayedBar(...)

Description : Converts a written bar number (from 1 to score.NumberOfBars) into played bar number (from 1 to score.NumberOfPlayedBars).
Parameters : Number: Written bar
(optional) Number: number of times this written bar has been played. 1 if it is the first time it is played, two if it is the second time... Default = 1
Return value : Number: Played bar or 0 if this bar is never reached

Score.PlayedTimeToMs(...)

Description : Converts a played time to number of millisecond elapsed from the beginning of the score. Played time differs from the written time position because of break symbols.
Parameters : Number: Played time
Return value : Number: Milliseconds from the beginning of the score or -1 if the played time position is never reached

Score.MsToPlayedTime(...)

Description : Converts number of millisecond elapsed from the beginning of the score to played time value. Played time differs from the written time position because of break symbols.
Parameters : Number: Milliseconds from the beginning of the score
Return value : Number: Played time or -1 if the played time position is never reached

Score.AbsolutePixelToWrittenTime(...)

Description : Convert an absolute pixel position to time position. The absolute pixel is the pixel position in the score when the score is in scroll mode. The pixel position include the bar margins, so you can access this area using the Pixel position.
Parameters : Number: Pixel
Return value : Number: Time position in the written music

Score.WrittenTimeToAbsolutePixel(...)

Description : Convert a time position to absolute pixel position
Parameters : Number: Time position in the written music
Return value : Number: Pixel

Score.WrittenTimeToTempo(...)

Description : Convert a time position in current tempo value
Parameters : Number: Time position in the written music
Return value : Number: Tempo

4.3.1.4.7. Engraver

Score.SaveEngraverSetup(...)

Description : Save all the engraver settings related to this score
Parameters : String: File name to load
Return value : True if loaded.
Example :

```
score=FrontScore()
path=GetUserSettingsPathName("Engraver").."MyEngraver.eng"
score.SaveEngraverSetup(path)
```

Score.LoadEngraverSetup(...)

Description : Load all the engraver settings from a file and relate them to the score.
Parameters : String: File name to load
Return value : True if loaded.

4.3.1.4.8. Exporting

Score.MusicExport(...)

Description : Export the score on disk in various format. The current file name and path related to the score is not changed.
Unlike with Score.Save(...) (page 240) method, the current file name and folder location are not changed by this method.
Parameters : Optional :
- String : The file name or ASK_FOR_NAME if the standard select file dialog box will open. The extension related to the file format will be automatically added to the file name.
- Number : the file format as defined in MSDefine (page 100)

FORMAT_STANDARD=1
FORMAT_EXPORT=2
FORMAT_MU3=3
FORMAT_MIDI0=4
FORMAT_MIDI1=5
FORMAT_KAR=6
FORMAT_RMTF=7
FORMAT_CMF=8
FORMAT_ABC=9
FORMAT_TAB=10
FORMAT_MUT=13
FORMAT_AIFF=14
FORMAT_WAV=15
FORMAT_SND=17
FORMAT_RAW=18
FORMAT_MP3=21
FORMAT_TEMPLATE=22
FORMAT_OGG=24 (experimental)
FORMAT_STAFF_TEMPLATE=25
FORMAT_XML=26
FORMAT_XML_COMPRESSED=27
FORMAT_MXL=28

by default FORMAT_STANDARD

If the select export format is a digital format (FORMAT_AIFF,FORMAT_WAV,FORMAT_SND,FORMAT_RAW, FORMAT_MP3 or FORMAT_OGG) the default export settings are : 44 khz, 16 bits, stereo, volume fixed to 64, the export will end smoothly (not exact size).
If the export is FORMAT_MP3 or FORMAT_OGG the bit rate is set to 128.

You can change these digital export default setting providing additional parameters :
- Number : Sample rate. Values allowed are : 96000 (96kHz), 48000 (48 kHz), 44100 (44 Khz) ,22050 (22 Khz), 11025 (11 Khz). Other values will be converted to the closest legal value.(ignored in MP3 export)
- Number : Bits per sample. Values allowed are : 16 or 8. (ignored in MP3 export)
- Number : Volume from (0 no sound to 127 maximal volume).
- Boolean : true if stereo, false if mono.
- Boolean : true if exact size, false if smooth end.
- Number : Deprecated. Previously bitrate.
- Boolean : true if normalise volume (the volume maximum will be computed), false by default
- Number : quality from 0 to 4. 0 is low quality, fast export, 4 high quality, slow export. Default value is 1.

If the selected export format is FORMAT_KAR, FORMAT_MIDI0,FORMAT_MIDI1 an additional parameter allow to define all staves with lyrics will be exported.
This avoid to ask the user to select the staves to export when there is more one staff with lyric in the document.
- Boolean : true if all staves will be exported, by default, false

If the selected export format is a digital format, an additional parameter allow to disable the export dialog box.
- Boolean : true if dialog box will not be displayed, by default, false

If the selected export format is MP3, additionals parameters allow to setup the ID3 tags
- Number : 0=no ID3 data, 1=ID3 data Windows encoding, 2=ID3 data Unicode 16LE encoding (by default),3= ID3 data Unicode 16LE encoding, 4= ID3 data UTF-8 encoding
- Boolean : true if synchronised lyrics are added (SYLT ID3 tag) (default is true)
- Boolean : true if raw lyrics are added (USLT ID3 tag) (default is true)
- Staff: Lyrics are extracted from this staff, if nil the first staff with lyrics is used.
- Number : ms of silence to add before the music
- Number : ms of silence to add after the music
- Number : Number of metronomes bars to add.

Return value : True if file saved, complete name of the exported file
Example :

```
-- Save the front myScore in Wav file format in the "Music Export" folder
-- then, open the result file using the standard Wav application
Include "MSDefine"
local ok,exportname,theCompleteFileName
```

```
-- Getting the front score
myScore=FrontScore()
if(myScore ~= nil) then
   -- Creating complete file name in the export folder
   theCompleteFileName=GetUserExportPathName("Music Export").."MyExport.wav"
   -- Exporting the file
   ok,exportname=myScore.MusicExport(theCompleteFileName,FORMAT_WAV,44100,16,64,false,true)
   if(ok==true) then
      -- Opening the result file
      OpenFileWithDefaultApplication(exportname)
   end
else
   Message("Please load a score");
end
```

Score.WebExport(...)

Description : Export the score on disk for web sharing. The file can include music data, score pictures,etc
Parameters : · String : The file name or ASK_FOR_NAME if the standard select file dialog box will open. The extension related to the file format will be automatically added to the file name.
Optional :

· Number : the file format as defined in MSDefine (page 100)
FORMAT_MYRWEB=29, this is binary format for internal use, not documented, sorry.
FORMAT_HTML=30, this is the standalone format which can be directly used by a web browser by default FORMAT_HTML
· Boolean : true if the file will be generated only if not already exist (by default false)
· String : Advanced setup, all settings separated by ';',for example : playmarker=bar;volume=80;
· Boolean : true if the score will be displayed on the web page (by default the last value defined in WebExportSteup, by default true)
· Boolean : true if all views will be included, false only main view (by default the last value defined in WebExportSteup, by default true)
· Boolean : true if lyrics could be displayed in a separate page (as a karaoke) (by default the last value defined in WebExportSteup, by default true)
· Boolean : true if music could be played (by default the last value defined in WebExportSteup, by default true)
· Boolean : true if piano keyboard could be displayed (by default the last value defined in WebExportSteup, by default true)
· Boolean : true if metronome could be played (by default the last value defined in WebExportSteup, by default true)
· Number : music quality from 0 (low quality, shortest file) to 4 (by default the last value defined in WebExportSteup, high quality , biggest file) (by default 2)
· Boolean : true if myr file will be embedded (by default the last value defined in WebExportSteup, by default true)
· Boolean : true if data will be compressed (by default the last value defined in WebExportSteup, by default true)
· Number : automatic group of staves. On the web page the user can select which group is played. (by default the last value defined in WebExportSteup, by default 0)
0 = No automatic group, use Staff.MyrWebGroup (page 268) to group the staves in a same audio track.
1 = First staff in a track, all others in another track.
2 = All staves with lyrics in a track, all others staves in another track.
3 = Each staves with lyrics in a separate track, all others staves in another track.
4 = Each staves in a separate track.
Return value : String : Complete file name of generated file.

Related methods and values:
Application.WebExportSetup(...) (page 195)
Application.GetWebExportSetup(...) (page 195)
Staff.MyrWebGroup (page 268)
Example :

```
-- Save the front myScore in MyrWeb file format in the "Music Export" folder
Include "MSDefine"
local myScore
local theCompleteFileName

-- Getting the front score
myScore=FrontScore()
if(myScore ~= nil) then
   -- Creating complete file name targetting the export folder
   theCompleteFileName=GetUserExportPathName("Music Export").."MyExport"
   -- Exporting the file
   myScore.WebExport(theCompleteFileName)
else
   Message("Please load a score");
end
```

Score.MovieExport(...)

Description : Export the score as a movie.
Parameters : · String : The file name or ASK_FOR_NAME if the standard select file dialog box will open. The extension specifie the file format to generate : mp4, mov, avi, mkv...
Optional :
· String : Text fore color (default is "FFFFE0")
· String : Text back color (default is "000080")
· Number : Display text duration in seconds (default is 3)
· Number : Crossfade text duration in seconds (default is 1) 0 mean no cross fade
· Number : Fade duration in seconds (default is 4) 0 mean no fade.
· Boolean : Add composer,title, remarks first (default is true)
· Boolean : Add audio track (default is true)
· Boolean : Played symbols in color (default is true)
· String : String to add before
· String : String to add before
· String : String to add before
· String : String to add after
· String : String to add after
· String : String to add after
Return value : String : Complete file name of generated file if no error. nil if error
Example :

```
-- Export the front myScore in .mp4 file format in the "Music Export" folder
local myScore
local theCompleteFileName

-- Getting the front score
myScore=FrontScore()
if(myScore ~= nil) then
   -- Creating complete file name targetting the export folder
   theCompleteFileName=GetUserExportPathName("Music Export").."MyExport.mp4"
   -- Exporting the file
   myScore.MovieExport(theCompleteFileName)
else
   Message("Please load a score");
end
```

```
function ProcessScores(CurrentFile)
    local score
    local result

    print("Processing "..CurrentFile)
    -- Loading score
    score=Load(CurrentFile)
    if score ~=nil then
        -- Set the window frame for export
            score.SizeWindow(600,400)
            score.AutoScale(false)
        -- Creating complete file name targetting the export folder
            thePath,theName,theExtension=SplitName(CurrentFile)
            theCompleteFileName=GetUserExportPathName("Music Export")..theName..".mp4"
        -- Exporting the file
            result=score.MovieExport(theCompleteFileName)
        -- Dispose score
            score.Dispose()
        -- Error
            if result==nil then
                -- Stop process
                return false
            end
    end
    -- Cont process
    return true
end

BrowseFolder(GetFolderName(GetUserMusicPathName()),"*.myr", ProcessScores,true)
```

Score.GraphicalExport(...)

Description : Export graphically the score. The preview of the score is provided and the user can change the page setup, export one or more pages, select the export file format. The page setup applied to the score is stored in the Score.ExportSetup (page 208) value of the Score object. You can edit it or apply changes before cal the GraphicalExport method.
Parameters : Optional :

- Number. The file format to export. In this case, the preview is not displayed, the user is prompted from a file name and all the pages are exported.

Allowed file formats are :

EXPORT_EPS=1 Standard EPS format
EXPORT_FPCT=2 FPCT file format
EXPORT_PICT=3 Pict file format
EXPORT_BMP=4 BMP file format

EXPORT_WMF=5 WMF file format (available only on Windows)

EXPORT_EMF=6 EMF file format (available only on Windows)
EXPORT_PNG=7 PNG file format
EXPORT_JPG=8 JPG file format (available only on Windows and Mac/OS)
EXPORT_GIF=9 GIF file format (available only on Windows)
EXPORT_SVG=10 SVG file format
EXPORT_SVG=11 multipage SVG file format
EXPORT_PDF=12 PDF file format

- String. The complete file name to export, including path. In this case,the preview is not displayed, the files are saved (one file per page). The number of the page will be added to the file name.

The standard extension related to the file type will be added to the name, so you can omit the extension. If the string is empty (""), or nil the file name is asked to the user.

- Number. The width of the output in 10th of millimeter, or 0 for default value. By default the current paper size for this document.
- Number. The height of the output in 10th of millimeter,or 0 for default value. By default the current paper size for this document.
- Number. The precision of the output in dot per inches,or 0 for default value. By default the maximum printer resolution.
- Number. First page to export or 0 for default value. (by default 1)
- Number. Last page to export or 0 for default value. (by default number of pages in the document)
- For JPEG export, quality default 1
- Boolean: true if a confirmation is asked if the target file exist (by defualt false)

Return value : String : The complete file name (including path) of the first page exported, nil if the user stopped the export
Example :

```
-- This sample will export the first page of the front score in
-- BMP file format in the "Graphical Exports" folder
-- of the user with the name "MyExport.bmp" and open it with the
-- current BMP viewer application.
local myScore
local theCompleteFileName
local theResultFileName
local theNumberOfThePage

myScore=FrontScore()
if(myScore ~= nil) then
   -- We will export the page # 1
   theNumberOfThePage=1
   --- Selecting path and file name
   theCompleteFileName=GetUserExportPathName("Graphics Export").."MyExport"
   --- Exporting and getting the file name as result
   theResultFileName=myScore.GraphicalExport(4,theCompleteFileName,0,0,0,theNumberOfThePage,
   theNumberOfThePage)
   if(theResultFileName~=nil) then
      -- Opening the picture file
      OpenFileWithDefaultApplication(theResultFileName);
   end
else
   Message("Please load a score.")
end
```

Score.SelectPaperArea(...)

Description : Open the paper area selection box
Parameters : None
Return value : None

4.3.1.4.9. FreeObject

Score.NewFreeObject(...)

Description : Inserts a FreeObject object
Parameters : Number : FreeObject kind. (See constants in MSDefine.mys)
Return value : Inserted FreeObject object or nil if impossible to insert

4.3.1.4.10. Global bar settings

Score.GetGlobalBarSetting(...)

Description : Gets the global bar setting object for a given bar.
Parameters : Bar number
Return value : GlobalBarSetting (page 318) object

Score.GetBarTimeSignatureChange(...)

Description : Gets the bar number at which a change occurs in time signature (or time signature is asked to be displayed).
Parameters : Number : Bar number to start search from
Return value : First bar in which time signature is different from the one at the given bar.
Returns Score.NumberOfBars+1 if there is no change till the end of the score
Returns nil if the provided bar number is outside the score bar number range or score object is deprecated.

Score.PropagateGlobalBarSetting(...)

Description : Copies the GlobalBarSetting of a bar to next bars (except the visibility booleans)
All affected time signatures are set invisible, and the one of next bar after the end of the copy is set as visible.
Parameters : Number : Bar number to copy from
[optional]
Number: Bar number to stop copying to (end of score if omitted)
Return value : None

4.3.1.4.11. Information

Score.DisplayComments(...)

Description : Open the standard dialog box showing the comments related to the score.
Parameters : None
Return value : None

Score.EditComments(...)

Description : Open the standard dialog box for editing the comments related to the score.
Parameters : None
Return value : None

4.3.1.4.12. Local metrics

In a displayed staff, you can see various area: the staff lines with the symbols, the tablature, the bar numbers, the lyrics, the chord grid.
The staff object own miscellaneous values which allow to adjust the position of these area in the main staff area.
For example Staff.BaseLine (page 270) define the global base line for a staff.
By the same way you can adjust the staff area height, using Staff.Height (page 270) value.
Introduced with Harmony 9.1, you can now adjust separately the position of the area for each bar.
As a system is constituted of several bars, when different values are defined in the same system, the first value is taken.
So, typically, these settings are used in conjunction with the methods which extract system informations from the score: Score.BarRangeInSystem(...) (page 245),Score.GetStaffInSystem(...) (page 245).

While calling one of these local metrics methods to change some values, the page setup is not computed, so you can assume the systems are consistent.
The methods related to the tablature are defined in the Tablature object : tablature.SetLocalBaseLine(...) (page 373), tablature.GetLocalBaseLine(...) (page 373)
After applying all changes you want to the staff, you ought to call Staff.AdjustLocalMetrics(...) (page 274) or Score.AdjustLocalMetrics(...) (below), to adjust all values.
The staff heights are computed to reflect the changes, the useless changes are deleted, etc. and the score is tagged for update.
Two methods allow you define the displayed lyrics lines: Staff.GetLocalLyricLineVisibility(...) (page 276) and Staff.SetLocalLyricLineVisibility(...) (page 276)

Note: In case of merged (multi-voices) staves, the display settings are taken from the first merged staff.

Score.AdjustLocalMetrics(...)

Description : After applying changes to local metrics for example with it is recommanded to call this method to adjust all values:
The staff heights are computed to reflect the changes, the useless changes are deleted, etc.
Parameters : None
Return value : None

Score.DeleteLocalMetrics(...)

Description : Delete all local metrics related to this score.
Parameters : None
Return value : None

Score.ComputeSystemsLeftMargin(...)

Description : The left margin of all systems will be computed as the name of staves will not overlap the paper left area.
Parameters : Optional Boolean : true if a confirmation will be asked to the user (false by default)
Optional Boolean : true if a minimal margin will be computed and applied (false by default)
Return value : None

4.3.1.4.13. Managing

Score.Dispose(...)

Description : Closes the score and releases related data. If the score is playing, music stops.
Parameters : Optional : Boolean value for asking a confirmation before closing a modified score.
If the parameter is true, a confirmation is asked, if the parameter is false, score is closed immediately without confirmation.
(default : true)
Return value : True if the score has been actually closed, false if the user canceled the operation.

Score.CopyTo(...)

Description : Copy all the score settings and objects in another score.
Parameters : The target score.
Return value : None

4.3.1.4.14. Managing instruments

Score.AddInstrument(...)

Description : Adds an Instrument (page 428) object to the score's instrument list
Parameters : Instrument name
Optional :
Instrument sound ID
Return value : Instrument: Added instrument
Example :

```
-- Adds a 'My Piano" instrument if not already existing
-- The instrument will be related to the 'Harpsichord' sound
Include "MSSounds"
local myScore
local name
name="My Piano"
myScore=FrontScore()
if myScore~=nil then
    if myScore.GetInstrumentByName(name)==nil then
        instrument=myScore.AddInstrument(name,SoundArray["Piano-Harpsichord"])
    else
        Message("A '",name,"' instrument already exists")
    end
else
    Message("You must load a score first")
end
```

Score.GetInstrumentByName(...)

Description : Gets the Instrument (page 428) object, given its name
Parameters : Instrument name
Return value : Instrument: instrument or nil if not found

Score.DeleteUnusedInstruments(...)

Description : Deletes all the unused instruments from a score.
An instrument is considered as unused when not used by any staff (including its rules) or accompaniment, and not used by the metronome.
Parameters : none
Return value : Number of deleted instruments

4.3.1.4.15. Managing Metronome

Score.EditMetronome(...)

Description : Edit the metronome.
Parameters : None
Return value : True if the user had validated the changes.

4.3.1.4.16. Managing Slurs

Score.AdjustSlurs(...)

Description : Adjust the slurs appearence of the whole score to avoid overlapping
Parameters : None
Return value : None

4.3.1.4.17. Played bars

Score.SetPlayedBars(...)

Description : Force the list of played bars. If active, the break symbols will not be interpreted and the played bars will be the bars defined in the list
See Score.GetPlayedBars(...) (below) to get the actual list of played bars and Score.IsForcedPlayedBarsList (pag 10) to know the actual mode.
Parameters : String : List of bars to play. Bar number start at 1 and are separated by comma symbol. You can define a range using hyphen.
For example : 1,5,7,1-4,1-8
To switch off this mode send an empty string.
Return value : Boolean. True if the list of bars is valid, false otherwise. If the list is not valid, nothing is done.

Score.GetPlayedBars(...)

Description : rovide the list of played bars
Parameters : None
Return value : String : List of bars to play.

Score.EditGlobalDigitalEffects(...)

Description : Edit the global digital effects.
Parameters : None
Return value : None

Score.OutputLevel(...)

Description : Provide the current output levels.
Parameters : None
Return value : Left, right, back left and back right level from 0 to 255
Example :

```
left,right,backleft,backright=myScore.OutputLevel()
```

4.3.1.4.18. Playing music

Score.PlayMusic(...)

Description : Launch the music on the score. If a score is currently playing the music is stopped.
You can get the currently playing score using the Application.CurrentPlayingScore (page 179) value.
Parameters : Optional
Boolean: true if the lyrics will be displayed in a karaoke window, by default, the user preference.
Boolean: true if music will scroll, by default, the user preference.
Boolean: true if the follow tune dialog box will open, by default, the user preference.
Number: number of metronome bars to play before launching the music, by default, the user preference.
Number: indexed color of the played note, 0= no color, by default, the user preference.
Return value : The score which was currently played or nil if none.
See also : Target.PlayMusic(...) (page 399)

Score.SetPlayMark(...)

Description : Set/reset the play mark.
Parameters : Number : Measure number (0 to remove the play mark)
Optional : Number, line to start from 1 to number of lyrics line (if there is more than one line of lyric because of break symbols)
Return value : None

Score.GetPlayMark(...)

Description : Get the current play mark position.
Parameters : None
Return value : Number : Measure number
Number, line to start from 1 to number of lyrics line (if there is more than one line of lyric because of break symbols)

Score.ScrollMusic(...)

Description : Activate/deactivate the scroll while music is playing. If the score is not currently playing do nothing.

The default scroll mode can only be changed by the user in the General Setup dialog box.
Parameters : Boolean. true if the score ought to scroll, false otherwise
Return value : Boolean : The previous scroll mode status.

Score.StopMusic(...)

Description : Stop the music. If this score is playing the music is stoped. If the score is not playing, do nothing.
Parameters : None.
Return value : Nothing

Score.PauseMusic(...)

Description : Pause, unpause the music. If the score is not currently played, do nothing.
Parameters : Boolean : true if the music will be paused, false if the music will be unpaused
Return value : None
Example :

```
score=FrontScore()
if score~=nil then
    if score.IsMusicPaused==true  then
        score.PauseMusic(false)
    else
        score.PauseMusic(true)
    end
end
```

Score.ComputeNormalizedVolume(...)

Description : Compute the normalized volume for the score (digital output). The score is virtualy played with the default settings and the optimal volume is computed.
If a music is currently playing, the music is stopped.
This method can take several second.
Parameters : None
Return value : Number : the volume (0 to 127)
Example :

```
-- Save the front score, asking for a new name
score=FrontScore()
if(score ~= nil) then
    score.GeneralVolume=score.ComputeNormalizedVolume()
end
```

4.3.1.4.19. Positioning

Score.AutoView(...)

Description : Scroll the score (if necessary) to show the position.
Parameters : The staff to show, the time position to show

Score.PositionView(...)

Description : Scroll the score to align the position on screen.
Parameters : The staff to view, the time position to view
optional : horizontal position and vertical position in the window (by default, the center)
Warning: this positioning is approximate. The staff/time position shown at the required pixel might not exactly reflect the parameters you set.

Score.JumpToPage(...)

Description : Show the page.
Parameters : The number of the page to show.

Score.JumpToBar(...)

Description : Show the bar.
Parameters : The number of the bar to show.

Score.SetPositionAtPage(...)

Description : Sets the score window scroll bars so that the origin of the given page is displayed on top-left corner.
Parameters : Number : Page number from 1 to Score.NumberOfPrintedPages (page 211)
(Optional) Number : X offset in 72dpi pixel
(Optional) Number : Y offset in 72dpi pixel
Return value : none

Score.GetViewPosition(...)

Description : Returns the nearest time position and staff at a given pixel position in document window
Parameters : optional: horizontal position and vertical position in the window (by default, the center)
Return value : written time position in the score
nearest staff
pixel distance between the point and the staff area (0 if the point is in the area)
Warning: this calculation is approximate. The staff/time position returned for the required pixel might not exactly reflect the actual display.

4.3.1.4.20. Printing

Score.Print(...)

Description : Print the score.
Parameters : Optional :
- ·a boolean telling if a print preview will be displayed (by default the value defined in the General Configuration)
- ·a boolean telling if the printing option dialog box will be displayed (true by default) If the printing option is displayed, the score is preserved and the settings

can be undo using Score.Undo(...) (page 249)
- ·a boolean telling if the printer job dialog box will open (true by default)
- · number. if all previous parameters are false, you can define the first page to print
- · number. if all previous parameters are false, you can define the last page to print
- · Boolean. If true, print all views. True by default

Return value : None

Score.DoPrinterJob(...)

Description : Open the standard printer dialog job. The user can select many printing setup : number of copies, page range to print.
The number of copies is managed by the program. The script ought to manage the page range to print.
Parameters : Optional : Number of pages to be printed
Return value : Boolean : true if the user validate his choice false otherwise
Number : first page number to print
Number : last page number to print

Score.PreparePageForPrinting(...)

Description : Print the page in a local spool. The page is not sent to the printer. You can collect more than one page in the local spool then print all the pages using Score.PrintPendingPages(...) (below)
Parameters : · Number : Page to print. If the page is out of range a blank page is printed. Use Score.NumberOfPrintedPages (page 211) to know the number of pages.
Return value : Boolean : true if the page has been successfully spooled.

Score.PrintPendingPages(...)

Description : Print the page in the local spool. The pages was added to the local spool using Score.PreparePageForPrinting(...) (above) method.
Parameters : None
Return value : None

Score.ComputePageSetup(...)

Description : This method must be called so that changes in page setup are taken into account and pages are recalculated.
Parameters : None.
Return value : None

Score.SubstTokenString(...)

Description : Substitute (if any) the tokens $T,$M,$R,etc with their current value.
Parameters : The styled string (page 177)
Return value : Result styled string (page 177)

4.3.1.4.21. Protection

Score.AskPassword(...)

Description : If the score is protected by the author (Score.Protected) open the password input dialog box. If the score is not protected do nothing.
Parameters : None
Return value : True if the score is not protected, false if the score is still protected against changes.

4.3.1.4.22. Saving

Score.Save(...)

Description : Save the score on disk.
Unlike with Score.MusicExport(...) (page 229) method, the current file name and folder location can be changed b this method.
Parameters : Optional :
· String : The file name by default (the current file name). If a file name is provided, the current score file name become this file name.
or
· Number : ASK_FOR_NAME. The user will input a new file name and will be able to select the target location of the file.
· Boolean : true if the file name will be added to the the File menu (false by default)
Return value : True if file saved, complete name of the saved file
Example :

```
Include "MSDefine"
-- Save the front score, asking for a new name
score=FrontScore()
if(score ~= nil) then
    score.Save(ASK_FOR_NAME)
end
```

4.3.1.4.23. Selection

Score.SelectAll(...)

Description : Select all the score
Parameters : None
Return value : None

Score.DeselectAll(...)

Description : Deselect all the score
Parameters : None
Return value : None

Score.DeselectIndividually(...)

Description : Deselect all individually selected symbols
Parameters : None
Return value : None

Score.CutSelection(...)

Description : Copy all the selected objects in the scrap book and erase them.
Parameters : Optional : Value indicating if the copy will be applied only the notes, texts, breaks, etc.
Each symbol kind can be activated using a bit :
1 = notes and rests
2 = texts related to the staff
4 = lyrics
8 = chords
16 = chords diagrams
32 = graphics
64 = dynamics
128 = FXs
256 = break symbols
512 = time signatures
1024 = key signatures and clefs
By default, to all except breaks, time signature, keys and clefs.
Return value : None

Score.CopySelection(...)

Description : Copy all the selected objects in the scrap book.
Parameters : Optional : Value indicating if the copy will be applied only the notes, texts, breaks, etc.
Each symbol kind can be activated using a bit :
1 = notes and rests
2 = texts related to the staff
4 = lyrics
8 = chords
16 = chords diagrams
32 = graphics
64 = dynamics
128 = FXs
256 = break symbols
512 = time signatures
1024 = key signatures and clefs
By default, to all except breaks, time signature, keys and clefs.
Return value : None

Score.PasteSelection(...)

Description : Paste all the objects from the scrap book to the score at the current selection location.
Parameters : None
Return value : None

Score.EraseSelection(...)

Description : Erase all the selected objects.
Parameters : Optional : Value indicating if the copy will be applied only the notes, texts, breaks, etc.
Each symbol kind can be activated using a bit :
1 = notes and rests
2 = texts related to the staff
4 = lyrics
8 = chords
16 = chords diagrams
32 = graphics
64 = dynamics
128 = FXs
256 = break symbols
512 = time signatures
1024 = key signatures and clefs
By default, to all except breaks, time signature, keys and clefs.
Return value : None

Score.AddSelection(...)

Description : Add the content of the scrap book to the current selection
Parameters : None
Return value : None

Score.QuantizeSelection(...)

Description : Quantizes all the selected objects.
Parameters : Optional : Quantize settings :
Number : Quantize Power from 0 to 4
Boolean : true if triplets are managed
Boolean : true if grace notes are managed
Number : Max size (in ms) of grace notes. Only meaningful if previous boolean is true. Notes longer than this value are considered as real notes instead of grace notes
Boolean : true if staccato marks are added
Boolean : true if tenuto marks are added
Return value : None

Score.HumanizeSelection(...)

Description : Sets a random delay to all selected notes (makes them sound more "human"), and alter the notes to make them swing
Parameters : Optional :
Number : Humanizing Power from 0 (no inaccuracy) to 10 (maximum inaccuracy). Default is 3.
Number : Swing Power from 0 (no swing) to 4 (strong swing). Default is 0.
REMARKS: Score.HumanizeSelection(0,0) clears all delays and resets all pressure time to 100%, which is equivalent to "Mechanize".
Return value : None

Score.BeamSelection(...)

Description : Beams notes in the selection range.
Parameters : Optional :
Boolean : false (default) if beam all notes
true if beam according to beat
Return value : None

4.3.1.4.24. Staves

Score.UserChooseStaves(...)

Description : Ask the user to choose one or multiple staves. You will use Staff.InUserChoice (page 270) value to check the choosen staves.
Parameters : Optional : Boolean : true if several staves can be chosen (default), false if only one staff can be chosen.
Optional : Boolean : true if this affect only visible staves (by default false).
Return value : True if at least one staff was choosen, false if no staves selected.
Example :

```
-- Asks the user to choose staves
-- and display their names
local myScore=FrontScore()
local myStaff
if myScore==nil then
   print("You must open a score first")
else
   if(myScore.UserChooseStaves()==true) then
      --- Browsing the staves
      myStaff=myScore.FirstStaff
      while(myStaff~=nil) do
         if(myStaff.InUserChoice==true) then
            print("Choosen staff name is : "..myStaff.Title)
            end
         myStaff=myStaff.Next
      end
   end
end
```

Score.DeleteTemporaryStaves(...)

Description : Delete all temporary staves created with Score.AddTemporaryStaff(...) (page 244).
Please note, when a script is closed, all temporary staves are automaticaly deleted.
You will use this method, for example, in a floating window when you created temporary staves \in the method OnStartPlay and when the method OnStopPlay is called.
-- These methods take place in the general methods of a floating window
-- When the music start, a temporary staff is created
function OnStartPlay(dialog,score)
staff=score.AddTemporaryStaff()
if staff ~= nil then
for i=0,20 do
staff.InsertNote(DURATION_QUARTER*i,DURATION_QUARTER,5*12+i)
end
end
end
-- When the music stop, the temporary staff is removed
function OnStopPlay(dialog,score)
score.DeleteTemporaryStaves()
end
Parameters : None.
Return value : None

Score.DeleteEmptyStaves(...)

Description : Delete all empty staves.
Parameters : Optional: true if only empty multi-voices staves are processed. (false by default)
Return value : None

Score.AddStaff(...)

Description : Create a new empty staff.
Parameters : Optional :
· staff type as defined in MSDefine (page 100)
STAFF_TYPE_STANDARD=0
STAFF_TYPE_DRUM_GRID=2
STAFF_TYPE_TEXT=3
STAFF_TYPE_TEMPO=4
STAFF_TYPE_TABLATURE=7
STAFF_TYPE_DRUM=8
STAFF_TYPE_GREGORIAN=9
STAFF_TYPE_DIGITAL_TRACK=10
by default STAFF_TYPE_STANDARD
· Clef type as defined in MSDefine (page 100)
CLEF_TYPE_TREBLE=1
CLEF_TYPE_BASS=2
CLEF_TYPE_UT=3
· String : the title of the staff
· Number : the standard id of the instrument. A new instrument with this ID is created and related to the staff.
· Boolean : true if this staff has to be displayed right now, false otherwise. False is quicker (default)
Return value : A Staff object.
Example :

```
Include "MSDefine"
-- Adds a staff in the frontmost score
myScore=FrontScore()
if myScore==nil then
    print("You must open a score first")
else
    myStaff=myScore.AddStaff(STAFF_TYPE_STANDARD,CLEF_TYPE_TREBLE,"new staff",0)
    if myStaff==nil then
        print("Impossible to add staff")
    end
end
```

Score.AddTemporaryStaff(...)

Description : Create a new empty invisible staff. This staff will be deleted when the script will end. Typically this kind of Temporary staff is used for local works on the score.
Parameters : Optional :
· staff type as defined in MSDefine (page 100)
STAFF_TYPE_STANDARD=0
STAFF_TYPE_DRUM_GRID=2
STAFF_TYPE_TEXT=3
STAFF_TYPE_TEMPO=4
STAFF_TYPE_TABLATURE=7
STAFF_TYPE_DRUM=8
STAFF_TYPE_GREGORIAN=9
STAFF_TYPE_DIGITAL_TRACK=10
by default STAFF_TYPE_STANDARD
· Clef type as defined in MSDefine (page 100)
CLEF_TYPE_TREBLE=1
CLEF_TYPE_BASS=2
CLEF_TYPE_UT=3
· String : the title of the staff
· Number : the standard id of the instrument. A new instrument with this ID is created and related to the staff.
Return value : A Staff object.

4.3.1.4.25. Staves group

Score.AddStavesGroup(...)

Description : Create a new staves group.
Parameters : All optionals
Number : The symbol displayed. By default SYMBOL_BRACE.
The following constants are defined in MSDefine :
SYMBOL_BRACE a brace
SYMBOL_BRACKET a bracket
SYMBOL_SUB_BRACKET a sub-bracket
SYMBOL_BRACKET_UP a bracket but only with the up part
SYMBOL_BRACKET_DOWN a bracket but only with the down part
SYMBOL_LINE a line (in fact a bracket without up and down
SYMBOL_NONE no symbol displayed
Number : The join mode,. By default JOIN_LINE_NONE.
The following constants are defined in MSDefine :
JOIN_LINE_NONE : no line between staves
JOIN_LINE_STANDARD : a line join the staves inside the group
JOIN_LINE_MENSURSTRICH : a line join the staves, but outside the staff lines. No ba$r line is displayed on the staff except for a repeat symbol.
Staff : The staff where begins the group, by default first staff of the score
Staff : The staff where ends the group, by default the last staff of the score
String : Group name, par default empty string
String : Group abreviated name, par default empty string
Number: The horizontal offset of the group from the left of the staff area. A positive value.
If this value is not provided, a default value is computed to avoid overlapping between groups symbols.
Return value : A staves group object.

4.3.1.4.26. Systems

Score.NumberOfSystemsInPage(...)

Description : Provide the number of systems in a given page. This depends of the page setup.
Parameters : Page number from 1 to Score.NumberOfPrintedPages (page 211)
Return value : Number of systems in this page
Example :

```
--Extracting score
score=FrontScore()
-- Displaying number of printed pages
print("There is "..score.NumberOfPrintedPages.." page(s) in the score.")
-- Dumping the number of systems by page
for numPage=1,score.NumberOfPrintedPages do
   print("There is "..score.NumberOfSystemsInPage(numPage).." system(s) in page "..numPage)
end
```

Score.NumberOfStavesInSystem(...)

Description : Provide the number of staves in a given system. This depends of the page setup: for example staves can be hidden for printing.
Parameters : Number : Page number from 1 to Score.NumberOfPrintedPages (page 211)
Number : System number from 1 to Score.NumberOfSystemsInPage(...) (page 244)
Return value : Number of staves in this system
Example :

```
--Extracting score
score=FrontScore()
-- Displaying number of printed pages
print("There is "..score.NumberOfPrintedPages.." page(s) in the score.")
-- Dumping the number of systems by page
for numPage=1,score.NumberOfPrintedPages do
   print("  There is "..score.NumberOfSystemsInPage(numPage).." system(s) in page "..numPage)
   for numSystem=1,score.NumberOfSystemsInPage(numPage) do
      print("   There is "..score.NumberOfStavesInSystem(numPage,numSystem).." staves in system "..
      numSystem.." in page "..numPage)
   end
end
```

Score.BarRangeInSystem(...)

Description : Provide the range of bars for a given system.
Parameters : Number : Page number from 1 to Score.NumberOfPrintedPages (page 211)
Number : System number from 1 to Score.NumberOfStavesInSystem(...) (above)
Return value : Number: first bar of this system
Number: last bar of this system
Example :

```
--Extracting score
score=FrontScore()
-- Displaying number of printed pages
print("There is "..score.NumberOfPrintedPages.." page(s) in the score.")
-- Dumping the number of systems by page
for numPage=1,score.NumberOfPrintedPages do
   print("  There is "..score.NumberOfSystemsInPage(numPage).." system(s) in page "..numPage)
   for numSystem=1,score.NumberOfSystemsInPage(numPage) do
      firstMesure,lastMesure=score.BarRangeInSystem(numPage,numSystem)
      print("   System "..numSystem.." page "..numPage.." start at bar "..firstMesure.." and end at bar "..
      lastMesure)
   end
end
```

Score.GetStaffInSystem(...)

Description : Provide the staff from its index in a given system.
Parameters : Number : Page number from 1 to Score.NumberOfPrintedPages (page 211)
Number : System number from 1 to Score.NumberOfSystemsInPage(...) (page 244)
Number : Staff number from 1 to Score.NumberOfStavesInSystem(...) (above)
Return value : A staff or nil if the staff or the system is outside valid range.
Example :

```
--Extracting score
score=FrontScore()
-- Displaying number of printed pages
print("There is "..score.NumberOfPrintedPages.." page(s) in the score.")
-- Dumping the number of systems by page
for numPage=1,score.NumberOfPrintedPages do
   print("  There is "..score.NumberOfSystemsInPage(numPage).." system(s) in page "..numPage)
   for numSystem=1,score.NumberOfSystemsInPage(numPage) do
      print("   There is "..score.NumberOfStavesInSystem(numPage,numSystem).." staves in system ".. numSystem.." in page "..numPage)
      for numStaff=1,score.NumberOfStavesInSystem(numPage,numSystem) do
         staff=score.GetStaffInSystem(numPage,numSystem,numStaff)
         print("    Staff "..numStaff.." of system "..numSystem.." of page "..numPage.." have title : "..staff.Title)
      end
   end
end
```

Score.SystemArea(...)

Description : Provide the area in 72dpi pixel of the system on the page, relative to the paper area.
Parameters : Number : Page number from 1 to Score.NumberOfPrintedPages (page 211)
Number : System number from 1 to Score.NumberOfStavesInSystem(...) (page 245)
Return value : Numbers: left,top,right,bottom
Example :

```
--Extracting score
score=FrontScore()
-- Displaying number of printed pages
print("There is "..score.NumberOfPrintedPages.." page(s) in the score.")
-- Dumping the number of systems by page
for numPage=1,score.NumberOfPrintedPages do
   print("  There is "..score.NumberOfSystemsInPage(numPage).." system(s) in page "..numPage)
   for numSystem=1,score.NumberOfSystemsInPage(numPage) do
      left,top,right,bottom=score.SystemArea(numPage,numSystem)
      print("Left="..left.." Top="..top.." Right="..right.." Bottom="..bottom)
   end
end
```

Score.BarToSystem(...)

Description : Provide the page and system where the bar is displayed.
Parameters : Number : written bar number
Return value : Number : Page number from 1 to Score.NumberOfPrintedPages (page 211)
Number : System number from 1 to Score.NumberOfStavesInSystem(...) (page 245)
Example :

```
--Extracting score
score=FrontScore()
-- Dumping the number of systems by page
for numBar=1,score.NumberOfBars do
   numPage,numSystem=score.BarToSystem(numBar)
   print("  Bar # "..numBar.." is displayed page #"..numPage.." system "..numSystem)
end
```

4.3.1.4.27. Tags

Some objects on the score can be tagged with a string : Symbols, Free Objects, Pictures.
This allow the script to find later the objects. The tags are saved in the .myr file.

Score.SearchSymbolsByTag(...)

Description : Browse the score, search the provided tag (case sensitive) and return an array of the symbols.
Parameters : String : The tag to search
Return value : A lua table with all matching symbols.

Score.ClearTagSymbols(...)

Description : Erase all tags related to the symbols.
Parameters : None
Return value : None

Score.SearchFreeObjectsByTag(...)

Description : Browse the score, search the provided tag (case sensitive) and return an array of the free objects.
Parameters : String : The tag to search.
Return value : A lua table with all matching free objects.

Score.ClearTagFreeObjects(...)

Description : Erase all tags related to the free objects.
Parameters : None
Return value : None

Score.SearchPicturesByTag(...)

Description : Browse the score, search the provided tag (case sensitive) and return an array of the pictures.
Parameters : String : The tag to search
Return value : A lua table with all matching pictures.

Score.ClearTagPictures(...)

Description : Erase all tags related to the pictures.
Parameters : None
Return value : None

Score.SearchOrnamentsByTag(...)

Description : Browse the score, search the provided tag (case sensitive) in ornaments related to notes and return an array of the ornaments.
Parameters : String : The tag to search
Return value : A lua table with all matching ornaments.

Score.ClearTagOrnaments(...)

Description : Erase all tags related to the ornaments.
Parameters : None
Return value : None

4.3.1.4.28. Targets

Score.InsertTarget(...)

Description : Inserts a Target (page 398) object
Parameters : Number : Bar number
String : Name (optional)
Return value : Inserted Target object or nil if impossible to insert

If Score.AutomaticTargetCaption is true, the name is computed except if a name is provided.
If a target already exist at this bar number localtion it is deleted.
Example :

```
Include "MSDefine"
-- Repeats bars 3 to 5 of the front score
myScore=FrontScore()
if myScore==nil then
   print("Please load a score first")
else
   --We preserve the current score state,
   --so the user will be able to undo the changes.
   myScore.Preserve()
   myScore.InsertTarget(1)
end
```

Score.RegenerateTargets(...)

Description : Regenerate all targets contents
Parameters : None
Return value : None
If Score.AutomaticTargetCaption is false, no operation.

4.3.1.4.29. Tempo

Score.GetGeneralTempo(...)

Description : This is the general tempo of the score. This general tempo can be changed into the score by tempo objects.
Parameters : None
Return value : The tempo (floating point number)
Example :

```
score=FrontScore()
if(score~=nil) then
   Message("Tempo is :"..score.GetGeneralTempo())
end
```

Score.GetCurrentTempo(...)

Description : This is the current tempo of the score. If the music is playing, general tempo can be changed localy by tempo object.
If the music is not playing return the general tempo.
Parameters : None
Return value : The current tempo (floating point number)
Example :

```
score=FrontScore()
if(score~=nil) then
   Message("Tempo is :"..score.GetCurrentTempo())
end
```

Score.SetGeneralTempo(...)

Description : Change the general tempo of the score. If you apply change to score while it is playing, the tempo change in real-time.
Parameters : The tempo (floating point number)
Optional : a boolean indicating if the tempo objects will be adapted, true by default.
Return value : None
Example :

```
score=FrontScore()
if(score~=nil) then
   score.SetGeneralTempo(score.GetGeneralTempo()+2.125)
   Message("Tempo is now : "..score.GetGeneralTempo())
end
```

Score.EditGeneralTempo(...)

Description : Edit the general tempo.
Parameters : None
Return value : True if the user had validated the changes.

4.3.1.4.30. Undo/redo

Score.Preserve(...)

Description : Preserve the current status of the score so, the user will be able to undo the changes the script will apply to the score. The use of this method is **strongly** recommended before any action on the score.
Parameters : String : the text which will be displayed in the menu
Return value : Boolean true if ok.

Score.Undo(...)

Description : Go back the previous status of the score by undoing all the changes applied since the last Preserve.
After undo, no symbol or staff object gotten from the previous state of the score remains valid.
Parameters : Optional
Boolean: Redo disabled after undo.
If this parameter is false (default), then the user will be able to redo this operation through the "Edit" menu.
If this parameter is true, then the user won't be able to redo this operation through the "Edit" menu.
Return value : None

4.3.1.4.31. Views

Score.GetCurrentView(...)

Description : Gets the current view
Parameters : None
Return value : View object currently selected for the score
Example :

```
-- Displays the name of the currently selected view
myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   myView=myScore.GetCurrentView()
   print(myView.Name)
end
```

Score.SetCurrentView(...)

Description : Sets the current view
Parameters : View: view to set as current
Boolean: true if the score has to be redrawn
False if the change of view is for working on the score internal data only (default).
Return value : None
Example :

```
-- Change current view to next one
myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   myView=myScore.GetCurrentView().Next
   if myView==nil then      -- No next view
      myView=myScore.Views[1] -- loops to general
   end
   myScore.SetCurrentView(myView,true);
   print("Current view is now "..myView.Name)
end
```

Score.NewView(...)

Description : Creates a new view
Parameters : String: name of the new view
(Optional) View: view used as a template (duplicated to create the new view) default: general view
(Optional) Number: Position (index) where to insert the new view in the score "Views" array. default: at the end of the list
Return value : View object newly created
Example :

```
-- Create a new view at the end of the list
myScore=FrontScore()
if myScore==nil then
    print("You must open a score first")
else
    myView=myScore.NewView("New");
    print("New view has been created at position "..myView.Index)
end
```

Score.EditViews(...)

Description : Edit the views.
Parameters : None
Return value : True if the user had validated the changes.

4.3.1.4.32. Window

Score.BringToFront(...)

Description : Bring the score window in front all other window (except the palette windows).
Parameters : None
Return value : None

Score.MoveWindow(...)

Description : Move the score window on the screen.
Parameters : Two numbers : the horizontal position in pixel, the vertical position in pixel. These coordinates don't take account of the window margin and refer to the struct area of the window (the area where the score is displayed). You can also use Score.WindowHorizontalPosition or Score.WindowVerticalPosition.
Return value : None

Score.SizeWindow(...)

Description : Change the score window size.
Parameters : Two numbers : the horizontal size in pixel, the vertical size in pixel. These coordinates don't take account of the window margin and refer to the struct area of the window (the area where the score is displayed). You can also use Score.WindowWidth or Score.WindowHeight.
Return value : None

Score.AutoSizeWindow(...)

Description : Set the score window size to an ideal size which will fill the usable area on the screen.
Parameters : None
Return value : None

Score.AutoScale(...)

Description : Change the display scale of the score in such a way than the page will fill the screen.Only works if the score is in page mode.
Parameters : Optional : Boolean: true if full window, false if full width. By default full window.
Optional: Number: minimum scale
Optional: Number: maximum scale
Return value : None

Score.ShowWindow(...)

Description : If the score window is hidden, show it.
Parameters : None
Return value : None

Score.HideWindow(...)

Description : Hide the score window.
Parameters : None
Return value : None

Score.GetPageMode(...)

Description : Tests the current display mode of the score.
Parameters : None
Return value : true if the score is currently displayed in page mode, false if in scroll mode

Score.GetPrintMode(...)

Description : Tests if the score is currently in printing mode.
Parameters : None
Return value : true if the score is currently in printing mode.

Score.SetPageMode(...)

Description : Switch the display of the score to "page mode".
Parameters : None
Return value : None

Score.SetScrollMode(...)

Description : Switch the display of the score to "scroll mode".
Parameters : None
Return value : None

4.3.2. Staff Object Type

General description of this object type : A staff in the score.
Related methods :
To get the first staff in a score, use Score.FirstStaff (page 208)
To add a staff to a score use Score.AddStaff(...) (page 243)
To add a temporary staff to a score use Score.AddTemporaryStaff(...) (page 244)
To ask the user to select one or more staves Score.UserChooseStaves(...) (page 242)

4.3.2.1. Staff values

4.3.2.1.1. Browsing staves

Staff.Next

Type : Staff
Description : Next staff in the score
If there is no next staff, value is nil.
Example :

```
-- Displays staff names of the frontmost score
myScore=FrontScore()
if(myScore ~= nil) then
    myStaff=myScore.FirstStaff
    while myStaff~=nil do
        print (myStaff.Title)
        myStaff=myStaff.Next
    end
end
```

Note : You cannot use this field if the score is protected by its author.

Staff.Previous

Type : Staff
Description : Previous staff in the score
If there is no previous staff, value is nil.
Example :

```
-- Displays staff names of the front score in reverse order
myScore=FrontScore()
if(myScore ~= nil) then
    myStaff=myScore.LastStaff
    while myStaff~=nil do
        print (myStaff.Title)
        myStaff=myStaff.Previous
    end
end
```

Note : You cannot use this field if the score is protected by its author.

Staff.Index

Type : Number
Description : Staff index from 1 to Score.NumberOfStaves (page 208)

4.3.2.1.2. Staff title

Staff.Title

Type : Styled string (page 177)
Description : Staff "long" title.
Example :

```
myScore=FrontScore()
if(myScore ~= nil) then
   -- Renames all staves
   i=1
   myStaff=myScore.FirstStaff
   while myStaff~=nil do
      myStaff.Title="Staff "..i
      i=i+1
      myStaff=myStaff.Next
   end
end
```

Note : You cannot use this field if the score is protected by its author.

Staff.TitleRotation

Type : Number
Description : Staff title rotation (in degree).

Staff.TitleAlignment

Type : Number
Description : Alignment of staff title text in its area (If the text have more than one line). 0=left,1=center,-1=right
This apply to short title and int32_t title.

Staff.TitleOrigin

Type : Number
Description : Origin of staff title area. 0= origin is left,1= center, -1= origin is right. This apply to short title and int32_t title.

Staff.TitleXOffset

Type : Number
Description : Staff "long" title X offset (72 dpi pixel) from its regular position

Staff.TitleYOffset

Type : Number
Description : Staff "long" title Y offset (72 dpi pixel) from its regular position

Staff.TitleFont

Type : String
Description : Staff title font name. Not used if title is styled.

Staff.TitleFace

Type : String
Description : Staff title font face (See <TextStyle> object for an explanation of Face string). Not used if title is styled.

Staff.TitleSize

Type : Number
Description : Staff title font size (72 dpi pixel). Not used if title is styled.

Staff.TitleRGBColor

Type : String
Description : Staff title font color (RGB components) Not used if title is styled.
EXAMPLE : -- Sets the staff title to red
myStaff.TitleRGBColor="FF0000"

Staff.ShortTitle

Type : String
Description : Staff "short" title.
Short title is an abbreviated version of the staff title, to be displayed on all systems but the first(indented) one.
If empty, then the int32_t name is used instead.

Staff.ShortTitleRotation

Type : Number
Description : Staff title rotation (in degree).

Staff.ShortTitleXOffset

Type : Number
Description : Staff "short" title X offset (72 dpi pixel) from its regular position

Staff.ShortTitleYOffset

Type : Number
Description : Staff "short" title Y offset (72 dpi pixel) from its regular position

Staff.ShortTitleFont

Type : Styled string (page 177)
Description : Staff "short" title font name

Staff.ShortTitleFace

Type : String
Description : Staff "short" title font face (See <TextStyle> object for an explanation of Face string)

Staff.ShortTitleSize

Type : Number
Description : Staff "Short" title font size (72 dpi pixel)

Staff.ShortTitleRGBColor

Type : String
Description : Staff "Short" title font color (RGB components)

Staff.TitlePrintMode

Type : Number
Description : Staff title printing mode.
Can be (see constants in MSDefine (page 100).mys)
NEVER_PRINT_NAME Never printed
PRINT_NAME_ONCE Printed on first system only
ALWAYS_PRINT_NAME Printed at each system

4.3.2.1.3. Bar numbering

Staff.BarNumberingType

Type : Number
Description : Bar numbering type.
Can be (see constants in MSDefine (page 100).mys)
BAR_NUMBERING_NONE : No bar number
BAR_NUMBERING_NUMBER : Bar numbering using numbers (1,2,...)
BAR_NUMBERING_LETTER : Bar numbering using letters (A,B,...)

Staff.BarNumberingFont

Type : String
Description : Bar numbering font name

Staff.BarNumberingFace

Type : String
Description : Bar numbering font face (See <TextStyle> object for an explanation of Face string)

Staff.BarNumberingSize

Type : Number
Description : Bar numbering font size (72 dpi pixel)

Staff.BarNumberingRGBColor

Type : String
Description : Bar numbering font color (RGB components)

Staff.BarNumberingFirstBar

Type : Number
Description : First bar to be numbered

Staff.BarNumberingOffset

Type : Number
Description : Value to be added to the real bar number to get the bar number to be displayed

Staff.IsBarNumberingStartPage

Type : Boolean
Description : true if bars are numbered only at the beginning of each system (staff line on printed page)

Staff.BarNumberingPrefix

Type : String (One character only)
Description : Character to be displayed just before the bar number

Staff.BarNumberingPlaceToPlace

Type : Number
Description : Bars can be numbered each 1, 2, 3,.. or n bars. This is what this field represents.

Staff.IsBarNumberingFrame

Type : Boolean
Description : true if a frame is drawn around bar number

Staff.IsBarNumberingAfterKey

Type : Boolean
Description : true if bar number is drawn after the clef, key and time signature when the bar includes such object.
false if bar number has to be drawn always just after the barline.

Staff.IsDisplayOnlyIfFirstStaffOfGroup

Type : Boolean
Description : true if bar number is drawn only if the staff is the first displayed staff in the group.

Staff.BarNumberingYPosition

Type : Number
Description : All bar numbers are written on a same line. This field defines the 72 DPI pixel offset between this line and the top of the staff area.
See also Staff.SetLocalBarNumberingYPosition(...) (page 276) to change localy this value.

4.3.2.1.4. Digital Track

Staff.DigitalTrack

Type : DigitalTrack
Description : if Staff.Type (page 269) is STAFF_TYPE_DIGITAL_TRACK (see constant in MSDefine (page 100) returns the related DigitalTrack (page 449) object.
Otherwise, returns nil.

4.3.2.1.5. Display

Staff.GraphRecord (R)

Type : Graph
Description : Current recorded graphical context (created with Staff.BeginRecord(...)).

4.3.2.1.6. Lyrics

There are two ways to get or set lyrics :
a) You can get or set the whole lyrics for the staff using the Staff.LyricsText (page 257) field
(you can also get the displayable lyrics through the Staff.DisplayableLyricsText (page 257) field.
b) You can access the lyrics cell by cell, using the dedicated Staff methods.

Staff.IsLyrics

Type : Boolean
Description : true if lyrics are related to this staff, false otherwise.

Staff.LyricsTitle

Type : String
Description : Lyrics line title

Staff.LyricsTitleFont

Type : String
Description : Lyrics line title font name

Staff.LyricsTitleFace

Type : String
Description : Lyrics line title font face

Staff.LyricsTitleSize

Type : Number
Description : Lyrics line title font size (72 DPI pixel)

Staff.LyricsTitleRGBColor

Type : String
Description : Lyrics line title RGB color

Staff.LyricsFont

Type : String
Description : Lyrics font name for displaying lyrics on score

Staff.LyricsFace

Type : String
Description : Lyrics font face for displaying lyrics on score

Staff.LyricsSize

Type : Number
Description : Lyrics line font size (72 DPI pixel) for displaying lyrics on score

Staff.LyricsRGBColor

Type : String
Description : Lyrics line RGB color for displaying lyrics on score

Staff.LyricsAlignment

Type : Number
Description : Lyrics alignment in its cell. Use constants in MSDefine (page 100).mys : ALIGN_RIGHT=-1, ALIGN_LEFT=0, ALIGN_CENTER=1

Staff.LyricsYPosition

Type : Number
Description : Lyrics line Y position (72 DPI pixel) relatively the the top of the staff area. See also Staff.SetLocalLyricsYPosition(...) (page 276) to change localy this value.

Staff.LyricsVerticalSize (R)

Type : Number
Description : Lyrics line vertical size (72 DPI pixel). Because of descent and leading this is different of LyricsSize.

Staff.LyricsNumberOfLines (R)

Type : Number
Description : Because of repeat, several lyrics lines can be related to a staff. This value provide the number of lines.
So, for example, the total area height for lyrics will be staff.LyricsVerticalSize*staff.LyricsNumberOfLines

Staff.IsLyricsAutoFormat

Type : Boolean
Description : True if automatic format is active.

Staff.IsLyricsFrame

Type : Boolean
Description : true if a frame is drawn around lyrics line.

Staff.LyricsText

Type : String
Description : Complete lyrics in original format, i.e. this string can include unbreakable spaces (^), unbreakable hyphens (=) and the miscellaneous commands (brackets) that can have been entered. A "/" character separates each played bar.
Note : If lyrics are disabled on this staff, this value is nil.

Staff.AutoSizeLyrics

Type : Boolean
Description : true if the lyrics size follow the staff size.

Staff.BeamFlag

Type : Boolean
Description : true if the flags as displayed with beam.

Staff.DisplayableLyricsText (R)

Type : String
Description : Complete lyrics in displayable format, i.e. this unbreakable spaces (^) have been replaced by regular spaces, unbreakable hyphens (=) by hyphens, and the miscellaneous commands (brackets) that can have been entered have been deleted
Note : If lyrics are disabled on this staff, this value is nil.

4.3.2.1.7. Managing chord diagrams

Staff.NumberOfDiagrams (R)

Type : Number
Description : Number of chord diagrams in this staff

Staff.NumberOfChordDiagrams (R)

Type : Number
Description : Number of custom chord diagrams related to the chord line

Staff.Diagrams (R)

Type : Diagram (page 363)
Description : Diagram (page 363) objects in the staff. Valid even if there is no such object
Must be used as an array, by providing an index.

Staff.FirstDiagram (R)

Type : Diagram (page 363)
Description : First Diagram (page 363) object in the staff.

Staff.LastDiagram (R)

Type : Diagram (page 363)
Description : Last Diagram (page 363) object in the staff.

4.3.2.1.8. Managing chord line

Staff.IsChord

Type : Boolean
Description : true if a chord line has to be displayed on this staff, false otherwise.

Staff.IsAutoChord

Type : Boolean
Description : true if chords are computed automatically according to notes in the staff, false if set "manually"

Staff.IsAutoTranspose

Type : Boolean
Description : true if manual chords are recomputed automatically when a transpose is applied.

Staff.AutoChordNumberOfNotes

Type : Number
Description : Number of notes (3,4 or 5) to be used to build a chord

Staff.IsAutoChordExtendedSearch

Type : Boolean
Description : true if chords are searched outside the current key modal array

Staff.IsAutoChordOnePerBar

Type : Boolean
Description : true if the same chord will be used for the whole bar

Staff.IsDrawChordName

Type : Boolean
Description : true if chord names are drawn

Staff.IsDrawChordDiagram

Type : Boolean
Description : true if chord diagrams (for fretted instrument) have to be drawn

Staff.IsAutoChordFromTablature

Type : Boolean
Description : true if chord line is computed from tablature

Staff.IsAutoChordBarreFirst

Type : Boolean
Description : true if barre chord diagrams will be choosen first

Staff.DiagramColor

Type : Number
Description : Diagram color index (use <RGBToIndex> and <IndexToRGB> global functions to convert back and forth between color index and RGB components)

Staff.ChordDiagramNeck

Type : Boolean
Description : Kind of fretted instrument used to draw chord diagrams.

Staff.IsDrawChordDiagramAbove

Type : Boolean
Description : true if chord diagrams have to be drawn above the chord names. false if below

Staff.IsPlayChord

Type : Boolean
Description : true if chords are played

Staff.ChordInstrument

Type : Instrument
Description : Instrument to play chords with

Staff.ChordArpeggioDelay

Type : Number
Description : Arpeggio delay for playing chords

Staff.ChordTitle

Type : String
Description : Chord line title

Staff.ChordTitleFont

Type : String
Description : Chord line title font name

Staff.ChordTitleFace

Type : String
Description : Chord line title font face

Staff.ChordTitleSize

Type : Number
Description : Chord line title font size (72 DPI pixel)

Staff.ChordTitleRGBColor

Type : String
Description : Chord line title RGB color

Staff.ChordFont

Type : String
Description : Chord font name for displaying chord line on score

Staff.ChordFace

Type : String
Description : Chord font face for displaying chord line on score

Staff.ChordSize

Type : Number
Description : Chord line font size (72 DPI pixel) for displaying on score

Staff.ChordRGBColor

Type : String
Description : Chord line RGB color for displaying on score

Staff.ChordAlignment

Type : Number
Description : Chord alignment in its cell. Use constants in MSDefine (page 100).mys
ALIGN_RIGHT=-1
ALIGN_LEFT=0
ALIGN_CENTER=1

Staff.ChordYPosition

Type : Number
Description : Chord line Y position (72 DPI pixel) relatively the the top of the staff area. See also Staff.SetLocalChordYPosition(...) (page 275) to change localy this value.

Staff.ChordVerticalSize (R)

Type : Number
Description : Chord line vertical size (72 DPI pixel) Because of descent and leading this is different of ChordSize.

Staff.IsChordAutoFormat

Type : Boolean

Staff.IsChordFrame

Type : Boolean
Description : true if a frame is drawn around chord line.

Staff.ChordFrequencyType

Type : Number
Description : Chord frequency defines how many chords cells are drawn per bar.
chord frequency type can be (see constants in MSDefine (page 100).mys) :
CELL_EVERY_N_BARS : one cell every N bars
CELL_N_TIMES_PER_BAR : several times per bar
CELL_EACH_BEAT : each beat
CELL_EACH_NOTE : each note
CELL_EACH_DIVISION : each division, given by the time signature denominator. Equal to beat except for compound time signatures.

Staff.ChordFrequencyValue

Type : Number
Description : parameter "N" for CELL_EVERY_N_BARS and CELL_N_TIMES_PER_BAR chord frequency types

Staff.ChordText

Type : String
Description : The whole set of chord names. Chord names are separated by a space, or "/" each time bar number changes.Note : Setting this value switches off the automatic chord calculation mode.
Note : If chord line is disabled on this staff, this value is nil

Staff.NumberOfChords (R)

Type : Number
Description : Number of chords in chord line.

4.3.2.1.9. Managing Clef, and Key signature changes

Staff.BarSettings

Type : BarSetting (page 332)
Description : Bar settings "array" for this staff. Must be used as an array, by providing a bar number as index.
This pseudo-array provides the BarSetting object currently used at the given bar.
Because effect of a BarSetting object propagates until a new object is encountered, the pseudo-array will provide reference to
the BarSetting object whose location is below or equal to the required bar.
To know whether the BarSetting object is located exactly at the bar you requested (numBar), use : if staff.BarSettings[numBar].Bar==numBar then...

Staff.FirstBarSetting

Type : BarSetting (page 332)
Description : First BarSetting object in the staff.

Staff.LastBarSetting

Type : BarSetting (page 332)
Description : Last BarSetting object in the staff.

4.3.2.1.10. Managing Dynamics/Tempo marks

Staff.NumberOfDynamics

Type : Number
Description : Number of dynamics/tempo marks in this staff

Staff.Dynamics

Type : Dynamic
Description : Dynamic/tempo objects in the staff. Valid even if there is no such object
Must be used as an array, by providing an index.
Example :

```
-- Gets type of the 4th dynamic/tempo object in this staff
                if myStaff.NumberOfDynamics>=4 then myFourthDyn=myStaff.Dynamics[4].Type end
```

Staff.FirstDynamic

Type : Dynamic
Description : First Dynamic/tempo object in the staff.

Staff.LastDynamic

Type : Dynamic
Description : Last Dynamic/tempo object in the staff.

4.3.2.1.11. Managing FX processors

Staff.NumberOfFXProcessor

Type : Number
Description : Number of FX processor objects in this staff

Staff.FXProcessors

Type : FXProcessor
Description : FX processor objects in the staff. Valid even if there is no FX processor.
Must be used as an array, by providing an index.
Example :

```
-- Gets the 4th FX processor in this staff
                if myStaff.NumberOfFXProcessor>=4 then myFourthFX=myStaff.FXProcessors[4].Title end
```

Staff.FirstFXProcessor

Type : FXProcessor
Description : First FX processor object in the staff.

Staff.LastFXProcessor

Type : FXProcessor
Description : Last FX processor object in the staff.

4.3.2.1.12. Managing parameter curves

Staff.Curves

Type : Curve (page 471)
Description : Curves attached to this staff. From this field, you can index using curve type constants (see MSDefine (page 100).mys) to access a given curve.
If no curve of this type is currently attached to the staff, will then return nil.
Example :

```
volumeCurve=myStaff.Curves[VOLUME_CURVE]
if volumeCurve then
    print("Volume curve is defined in staff",myStaff.Title")
 end
```

Staff.FirstCurve

Type : Curve (page 471)
Description : First curve attached to this staff. Curves are not sorted.

Staff.LastCurve

Type : Curve (page 471)
Description : Last curve attached to this staff. Curves are not sorted.

Staff.DisplayedCurve

Type : String
Description : Currently displayed curve type for this staff.
Curve types are 4-character string identifiers, defined in MSDefine (page 100).mys.
Here is the list of available curve types, followed by their internal value.

- VOLUME_CURVE "Volu"
- PANNING_CURVE "Ster"
- FREQUENCY_CURVE "Freq"
- RESONATOR_RESONANCE_CURVE "Resr"
- RESONATOR_FREQUENCY_CURVE "Resf"
- OVERDRIVE_POWER_CURVE "Ovep"
- OVERDRIVE_TONE_CURVE "Ovet"
- FLANGER_POWER_CURVE "Flap"
- EQUALIZER_TREBLE_CURVE "Equa"
- EQUALIZER_BASS_CURVE "Equg"
- REVERBERATION_POWER_CURVE "Revp"
- MIDI1_CURVE "Mid1"
- MIDI2_CURVE "Mid2"
- MIDI3_CURVE "Mid3"
- MIDI4_CURVE "Mid4"
- MIDI5_CURVE "Mid5"

In addition, three other types are available. Properly speaking, they are not real "curves" but alternate way of displaying staff :
"none" no curve displayed, regular display of the staff
"dlay" note delay and pressure time display
"vlct" note velocity display

4.3.2.1.13. Managing Pictures

Staff.NumberOfPictures

Type : Number
Description : Number of pictures in this staff

Staff.Pictures

Type : Picture
Description : Picture (page 358) objects in the staff. Valid even if there is no such object
Must be used as an array, by providing an index.

Staff.FirstPicture

Type : Picture
Description : First Picture object in the staff.

Staff.LastPicture

Type : Picture
Description : Last Picture object in the staff.

4.3.2.1.14. Managing Rules

Staff.NumberOfRules

Type : Number
Description : Number of rules in this staff

Staff.Rules

Type : Rule
Description : Rule (page 416)s in this staff. Valid even if there is no such object
Must be used as an array, by providing an index.
Example :

```
-- Gets the 4th rule of this staff
                if myStaff.NumberOfRules>=4 then myFourthRule=myStaff.Rules[4] end
```

Staff.FirstRule

Type : Rule
Description : First rule object in the staff.

Staff.LastRule

Type : Rule
Description : Last Rule object in the staff.

4.3.2.1.15. Managing staff lines and barlines

Staff.IsHideBarlines

Type : Boolean
Description : true if barlines are hidden

Staff.BreakSymbolColor

Type : Number
Description : Break Symbol color index (use <RGBToIndex> and <IndexToRGB> global functions to convert back and forth between color index and RGB components)
Note : Break symbols are applied to all staves at the same time. They are managed through the <Score> object type.

Staff.IsDrawLeftLimit

Type : Boolean
Description : true if left limit of the staff is drawn at the beginning of each system

Staff.IsDrawRightLimit

Type : Boolean
Description : true if right limit of the staff is drawn at the end of each system

Staff.LowerBarlineLimit

Type : Number
Description : Lower barline limit (in staff lines relatively to the base line of the staff)

Staff.UpperBarlineLimit

Type : Number
Description : Upper barline limit (in staff lines relatively to the base line of the staff)

Staff.LedgerLineSize

Type : Number
Description : Ledger line size (in 72 DPI pixels)

Staff.Lines

Type : String
Description : Staff line settings.
Each character matches a staff line, from bottom to top.
The 5 first characters represent 5 lines below the staff base line.
The next characters represents the staff base line
The 10 next characters represent 10 lines above the staff base line.
When character is a dot (.) or omitted, the matching line is not drawn. When it is an "X", the matching line is drawn.
Example :

```
-- example 1: standard lines for a regular staff
myStaff.Lines=".....XXXXX"
-- example 2: one-line staff (the only line to be drawn is the middle one)
myStaff.Lines="......X"
```

Staff.LineColor

Type : Number
Description : Staff line color index (use <RGBToIndex> and <IndexToRGB> global functions to convert back and forth between color index and RGB components)

4.3.2.1.16. Managing symbols

Symbols are notes, rests, ghost rests or bar repeat (%) that can appear on a staff.
Most of the management has to be performed through the <Symbol> object members.
From the staff, you can just get a first symbol for starting to browse them all, and get or set global settings for symbol display

Staff.NumberOfSymbols (R)

Type : Number
Description : Number of symbols (notes, rests, ghost rests and bar repeat symbol (%))in this staff

Staff.FirstSymbol (R)

Type : Symbol
Description : First symbol in this staff (nil if none)

Staff.LastSymbol (R)

Type : Symbol
Description : Last symbol in this staff (nil if none)

Staff.SymbolDefaultSize

Type : Number
Description : Default symbol size for this staff, from 10 to 200
Regular values are : Small=42, Medium=54, Standard=72, Big=108

Staff.SymbolDefaultColor

Type : Number
Description : Default symbol color index (use <RGBToIndex> and <IndexToRGB> global functions to convert back and forth between color index and RGB components)
Note : This color can be overriden by the color index of each symbol. It only affects symbols of color index #0

Staff.ShapeNoteType

Type : Number
Description : kind of "shape note", for FASOLA notation (See constants in MSDefine (page 100).mys)
NO_SHAPE_NOTE : no shape note system (default)
SHAPE_4_NOTE : 4-shape note notation
SHAPE_7_NOTE_AIKEN : 7-shape note, Aiken notation
SHAPE_7_NOTE_SWAN : 7-shape note, Swan notation
SHAPE_7_NOTE_WALKER : 7-shape note, Walker notation
SHAPE_7_NOTE_FUNK : 7-shape note, Funk notation
SHAPE_7_NOTE_JOHNSON : 7-shape note, Johnson notation

Staff.IsUncutNotes

Type : Boolean
Description : true if notes are not splitted when playing across a barline

Staff.IsHideRests

Type : Boolean
Description : true if rests are hidden

Staff.IsDisplayWholeRests

Type : Boolean
Description : true if a rest that fills completely a bar is displayed as a whole rest, whatever the time signature is

Staff.IsRestCentered

Type : Boolean
Description : true if a rest that fills completely a bar is displayed centered on the measure

Staff.IsGhostRestsDisplayedAsWholeRests

Type : Boolean
Description : true if ghost rests are displayed as a whole rests* (Deprecated use GhostRestsDisplayMode)

Staff.GhostRestsDisplayMode

Type : Number
Description : 0 if ghost rests are not displayed, 1 displayed as arrow, 2 displayed as pause, 3 displayed as real rests/

Staff.IsDrawOrnaments

Type : Boolean
Description : true if symbol ornaments are drawn

Staff.CharSet

Type : String
Description : the charset used to display musical symbols

Staff.IsNoteBeamAligned

Type : Boolean
Description : true if note beams are aligned to the middle of the staff

Staff.IsGraceNoteStemsAligned

Type : Boolean
Description : true if grace note stem ends are aligned to the middle of the staff

Staff.MaximumBeamTilt

Type : Number
Description : Maximum tilt value for beams from 0 (horizontal beams) to 100 (no constraint)

Staff.BeamThickness

Type : Number
Description : Beam thickness from 1 to 31

Staff.BeamGap

Type : Number
Description : Gap between beams from 1 to 7

Staff.SecondaryBeamMaxSize

Type : Number
Description : Maximum horizontal size of secondary beam line

Staff.UnbreakableSpaceSlurWidth

Type : Number
Description : Width of the slur below unbreakable space in lyric line (0 mean no slur)

Staff.ExtraSpaceBetweenLyricLines

Type : Number
Description : Extra space between lyric lines

Staff.IsStemCenteredOnHead

Type : Boolean
Description : true if stem is centered on the note head

Staff.DefaultStemDirection

Type : Number
Description : Default stem direction for notes in this staff.
Only notes that have their StemType property to 0=automatic are affected by this setting.
Possible values are:
0 = standard (stem up below middle line, stem down above)
1 = always down
2 = always up

4.3.2.1.17. Managing Text elements

This notion includes all text elements that are displayed on a score.
It therefore means chord line, as well as lyrics, will also be listed as a text elements, even if specific members exist to process this kind of object in a more convenient way.

Staff.NumberOfText

Type : Number
Description : Number of text elements in this staff

Staff.Texts

Type : StaffText
Description : Texts in the staff. Valid even if there are no texts.
Must be used as an array, by providing an index.
Example :

```
-- Gets the 4th text in this staff
                if myStaff.NumberOfText>=4 then myFourthText=myStaff.Texts[4].Content end
```

Staff.FirstText

Type : StaffText
Description : First text in the staff.

Staff.LastText

Type : StaffText
Description : Last text in the staff.

Staff.IsHideLyric

Type : Number
Description : true if lyrics are completely hidden in this staff

Staff.IsHideChord

Type : Number
Description : true if chord lines are hidden in this staff

Staff.IsHideText

Type : Number
Description : true if all text elements other than lyrics and chords are hidden in this staff

Staff.IsHideChordDiagram

Type : Number
Description : true if chords diagrams are hidden in this staff

Staff.IsHideGraphic

Type : Number
Description : true if graphics are hidden in this staff

Staff.IsHideDynamic

Type : Number
Description : true if dynamics are hidden in this staff

Staff.IsHideFXProcessor

Type : Number
Description : true if dynamics are hidden in this staff

4.3.2.1.18. Managing Virtual Singer voice

Staff.VirtualSingerVoice

Type : VSVoice (page 476)
Description : Virtual Singer voice related to the staff.

4.3.2.1.19. Miscellaneous

Staff.Score

Type : Score
Description : <Score> object this staff belongs to.

Staff.Instrument

Type : Instrument
Description : <Instrument> object this staff is played with.

Staff.Tablature

Type : Tablature
Description : Tablature settings for this staff (nil if no tablature).

Staff.StorageArea

Type : StorageArea
Description : StorageArea object for this staff

Staff.HiddenTransposition

Type : Number
Description : Transposition value (in semitone) this staff will receive when switching the score back to transposed mode.
This value cannot be read or written until the Score.TransposedMode (page 201) value is set to false.
It can also be changed when applying a regular transposition to a staff while the Score.TransposedMode (page 2 value is false.
When switching back into transposed mode (either when the user does it through the menu, or the script by setting Score.TransposedMode (page 201) to true),
this value is used to set the staff to the right key signature, according to the amount of transposition that has been set here.

Staff.IsChanged

Type : Boolean
Description : True if the staff was changed

Staff.NeverHide

Type : Boolean
Description : True if the staff was always printed, even if the mode "Don't print empty staves" is enabled and the staff is empty.

Staff.IsAmbitusEnabled

Type : Boolean
Description : True if the ambitus is displayed on the staff. The ambitus must be enabled on the document.

Staff.MyrWebGroup

Type : Number
Description : Value of web group for web export, all staves with the same value will be exported in a separate audio track
Related methods:
Application.WebExportSetup(...) (page 195)
Application.GetWebExportSetup(...) (page 195)
Score.WebExport(...) (page 230)

4.3.2.1.20. Multi-voices

Staff.IsMultiVoice (R)

Type : Boolean
Description : True if the staff is a part of a multi-voice staff (This is a merged staff)

Staff.IsLayerActive

Type : Boolean
Description : True if the staff is an active layer.

Staff.FirstMergedStaff (R)

Type : Staff
Description : Provide the first staff of the merged staves if the staff is merged, the staff itself if the staff is not merged.

Staff.LastMergedStaff (R)

Type : Staff
Description : Provide the next .

Staff.IsMergedWithNext (R)

Type : Boolean
Description : True if the staff is merged with the next staff.

4.3.2.1.21. Selection

Staff.FirstSelectedSymbol

Type : Symbol
Description : First symbol in this staff that is included in the selection range (nil if none)

Staff.LastSelectedSymbol

Type : Symbol
Description : Last symbol in this staff that belongs to the selection range (nil if none)

Staff.NumberOfSelectedSymbols ®

Type : Number
Description : Provide the number of selected symbols in the staff: symbols in the select area.

Staff.NumberOfSelectedSymbolsIndividually ®

Type : Number
Description : Provide the number of selected symbols individually in the staff.

4.3.2.1.22. Spoken Human Voice

Staff.SpokenVoiceName

Type : String
Description : The name of the spoken voice related to the staff. (Sara, Marcel, Samantha...)

Staff.SpokenVoiceLanguage

Type : String
Description : The language of the spoken voice related to the staff. (en, fr, it...)

Staff.SpokenVoiceGender

Type : Number
Description : The gender of the spoken voice related to the staff. 0=neutral, 1=male, 2=female

Staff.SpokenVoiceRate

Type : Number
Description : The word rate of the spoken voice related to the staff. from 100 to 300

Staff.SpokenVoiceMaximumRate

Type : Number
Description : The maximal word rate of the spoken voice related to the staff. from 100 to 300

Staff.SpokenVoicePitchOffset

Type : Number
Description : Offset of the pitch of the spoken voice related to the staff. from -200 to 200

4.3.2.1.23. Staff Attributes

Staff.Type

Type : Number
Description : Staff type (see constants in MSDefine (page 100).mys)
STAFF_TYPE_STANDARD : Staff for standard notation (bass/treble clef...)
STAFF_TYPE_DRUM_GRID : Drum grid (in drum grid staves, note pitches are always 0, and there are no chords)
STAFF_TYPE_TEXT : Text staff (no symbol)
STAFF_TYPE_TABLATURE : Tablature-oriented staff
STAFF_TYPE_DRUM : Drum staff
STAFF_TYPE_GREGORIAN : Gregorian staff
STAFF_TYPE_DIGITAL_TRACK : Digital track. When converting a staff to digital track, the content of the staff (notes, rests) is converted into digital data. The contrary is not true.

Staff.IsPlayed

Type : Boolean
Description : true if the staff is played (little louspeaker icon "on" in its margin)

Staff.IsPrinted

Type : Boolean
Description : true if the staff will appear on printing output (and in page mode)

Staff.IsSelected

Type : Boolean
Description : true if the staff is included in the selection range

Staff.InChordGrid

Type : Boolean
Description : true if the staff processed for building the global chord grid (see <Score> object type)

Staff.InUserChoice

Type : Boolean
Description : true if the staff has been selected through a user choice (See <Score> object type)

Staff.DefaultDrumDivision

Type : Number
Description : Default number of divisions if the staff is (or becomes) a drum grid

4.3.2.1.24. Staff metrics

Staff.Height

Type : Number
Description : Staff area height (in 72 DPI pixels). See also Staff.SetLocalHeight(...) (page 274) to change localy value.
Example :

```
--- Increases height of the first staff of the frontmost score
local myScore
local myStaff

--- Getting the score and its first staff
myScore=FrontScore()
if(myScore~=nil) then
   myStaff=myScore.FirstStaff
   if(myStaff ~=nil) then
      -- Preserves to undo changes
      myScore.Preserve()
      myStaff.Height=myStaff.Height+1
   end
else
   Message("Please load a score");
end
```

Staff.BaseLine

Type : Number
Description : Staff base line Y position (in 72 DPI pixels) relatively to the top of its area. See also Staff.SetLocalBaseLine(...) (page 275) to change localy this value.

Staff.LineSpacing

Type : Number
Description : Staff line spacing (in 72 DPI pixel).
Cannot be set, because it depends on the default symbol size that has been set for this staff (see <Staff.SymbolDefaultSize>)

Staff.GroupsMargin

Type : Number
Description : Space left in (72 dpi) at the left of the staff to display the group symbols (brace, bracket,etc)

4.3.2.2. Staff methods

4.3.2.2.1. Adding a BarSetting object

Staff.InsertBarSetting(...)

Description : Inserts a BarSetting object
Parameters : Number : Bar number
Return value : Inserted BarSetting object or nil if impossible to insert
If there is already a BarSetting object at this position on this staff, the existing object is not cleared. Only its display settings are changed so that all becomes visible.
Example :

```
-- Inserts a BarSetting at bar#2 of the first staff
myScore=FrontScore()
if myScore==nil then
   print("Please load a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in this score")
   else
      --We preserve the current score state,
      --so the user will be able to undo the changes.
      myScore.Preserve()
      myStaff.InsertBarSetting(2);
   end
end
```

Note : The newly inserted object keeps the same clef and key signature as what was set before (no change in notes), and displays clef, key and time.

4.3.2.2.2. Adding a Curve object

Staff.AddCurve(...)

Description : Adds a Curve object
Parameters : Number : Curve type to insert.
Curve types are 4-character string identifiers, defined in MSDefine (page 100).mys.

Here is the list of available curve types, followed by their internal value.
It is recommended to use one of these constants. Using directly a non-listed 4-character identifier enables to create custom curves, that are neither interpreted, nor available for the user to edit.
Note : three other identifiers are reserved and shouldn't be used : "none","vlct" and "dlay"

- VOLUME_CURVE "Volu"
- PANNING_CURVE "Ster"
- FREQUENCY_CURVE "Freq"
- RESONATOR_RESONANCE_CURVE "Resr"
- RESONATOR_FREQUENCY_CURVE "Resf"
- OVERDRIVE_POWER_CURVE "Ovep"
- OVERDRIVE_TONE_CURVE "Ovet"
- FLANGER_POWER_CURVE "Flap"
- EQUALIZER_TREBLE_CURVE "Equa"
- EQUALIZER_BASS_CURVE "Equg"
- REVERBERATION_POWER_CURVE "Revp"
- MIDI1_CURVE "Mid1"
- MIDI2_CURVE "Mid2"
- MIDI3_CURVE "Mid3"
- MIDI4_CURVE "Mid4"
- MIDI5_CURVE "Mid5"

Return value : Inserted Curve object or nil if impossible to insert
If there is already a Curve object of this type in the staff, the existing object is not cleared, and is returned just as if it has been just created.
Example :

```
Include "MSDefine"
-- Inserts a Volume Curve in the first staff
myScore=FrontScore()
if myScore==nil then
   print("Please load a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in this score")
   else
      volumeCurve=myStaff.AddCurve(VOLUME_CURVE)
   end
end
```

4.3.2.2.3. Beaming

Staff.ComputeBeam(...)

Description : Compute the beaming of all notes in the staff.
Parameters : Optional : true if stem directions will be computed too (false by default)
Return value : None

4.3.2.2.4. Deleting a staff

Staff.Delete(...)

Description : Deletes the whole staff.
Parameters : None
Return value : None
Example :

```
-- Deletes the first staff of the frontmost score
myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in the document")
   else
      myStaff.Delete()
   end
end
```

4.3.2.2.5. Drum Staff

Staff.CreateDefaultDrumRules(...)

Description : If the staff is a STAFF_TYPE_DRUM, create the default rules. In other case do nothing
Parameters : Optional, the set to apply. If the parameter is equal to -1, the select set dialog box will open.
The possible set values are :
0=standard set
8=room set
16=power set
24=electronic set
25=TR808 set
26=dance set
32=jazz set
40=brush set
48=orchestra set
56=SFX set
By default, 0.
Return value : None

4.3.2.2.6. Edit

Staff.EditDisplayMode(...)

Description : Ask the user to edit the staff display mode
Parameters : None
Return value : True if the user click "ok"

Staff.EditChordDisplay(...)

Description : Ask the user to edit the chord line display mode
Parameters : None
Return value : True if the user click "ok"

Staff.EditLyrics(...)

Description : Ask the user to edit the lyrics
Parameters : Optional
Number : measure to hilite
Return value : True if the user click "ok"

Staff.EditBarNumbering(...)

Description : Ask the user to edit the bar numbering
Parameters : Boolean : true if the user can apply the changes to all staves. (false by default)
Return value : True if the user click "ok"

Staff.EditChordLine(...)

Description : Ask the user to edit the chord line content as text
Parameters : None
Return value : True if the user click "ok"

Staff.EditTablature(...)

Description : Ask the user to edit the tablature setup
Parameters : None
Return value : True if the user click "ok"

4.3.2.2.7. Local metrics

In a displayed staff, you can see various area: the staff lines with the symbols, the tablature, the bar numbers, the lyrics, the chord grid.
The staff object own miscellaneous values which allow to adjust the position of these area in the main staff area.
For example Staff.BaseLine (page 270) define the global base line for a staff.
By the same way you can adjust the staff area height, using Staff.Height (page 270) value.
Introduced with Harmony 9.1, you can now adjust separately the position of the area for each bar.
As a system is constituted of several bars, when different values are defined in the same system, the first value is taken.
So, typically, these settings are used in conjunction with the methods which extract system informations from the score: Score.BarRangeInSystem(...) (page 245),Score.GetStaffInSystem(...) (page 245).

While calling one of these local metrics methods to change some values, the page setup is not computed, so you can assume the systems are consistent.
The methods related to the tablature are defined in the Tablature object : tablature.SetLocalBaseLine(...) (page 373), tablature.GetLocalBaseLine(...) (page 373)
After applying all changes you want to the staff, you ought to call Staff.AdjustLocalMetrics(...) (below) or Score.AdjustLocalMetrics(...) (page 235), to adjust all values.
The staff heights are computed to reflect the changes, the useless changes are deleted, etc. and the score is tagged for update.
Two methods allow you define the displayed lyrics lines: Staff.GetLocalLyricLineVisibility(...) (page 276) and Staff.SetLocalLyricLineVisibility(...) (page 276)

Note: In case of merged (multi-voices) staves, the display settings are taken from the first merged staff.

Staff.AdjustLocalMetrics(...)

Description : After applying changes to local metrics it is recommanded to call this method or Score.AdjustLocalMetrics(...) (page 235) to adjust all values:
The staff heights are computed to reflect the changes, the useless changes are deleted, etc.
Parameters : None
Return value : None

Staff.DeleteLocalMetrics(...)

Description : Delete all local metrics related to this staff
Parameters : None
Return value : None

Staff.SetLocalHeight(...)

Description : The value Staff.Height (page 270) allow you to define the general height of the staff.
With Staff.SetLocalHeight you can adjust localy the height of a staff in a system for example. This will only affect the height of the staff in this system.
Parameters : Number : Local staff area height (in 72 DPI pixels)
Number: First bar to apply
Number: Last bar to apply
Return value : None

Staff.GetLocalHeight(...)

Description : This will provide the local height of staff in a bar range.
Parameters : Number: First bar
Number: Last bar
Return value : Number : Local staff area height (in 72 DPI pixels), nil if the value can't be extracted
Example :

```
--Extracting score
score=FrontScore()
-- Displaying number of printed pages
print("There is "..score.NumberOfPrintedPages.." page(s) in the score.")
-- Dumping the number of systems by page
-- For all pages
for numPage=1,score.NumberOfPrintedPages do
   -- For all systems in page
   for numSystem=1,score.NumberOfSystemsInPage(numPage) do
      -- Extracting start bar and end bar for each system
      firstMesure,lastMesure=score.BarRangeInSystem(numPage,numSystem)
      -- For each staff in system
      for numStaff=1,score.NumberOfStavesInSystem(numPage,numSystem) do
         -- Extracting staff
         staff=score.GetStaffInSystem(numPage,numSystem,numStaff)
         height=staff.GetLocalHeight(firstMesure,lastMesure)
         if height==staff.Height then
            print("    Staff "..numStaff.." of system "..numSystem.." of page "..numPage.." have a default height ("..
            height..")")
         else
            print("    Staff "..numStaff.." of system "..numSystem.." of page "..numPage.." have a local height of : "..
            height)
         end
      end
   end
end
```

Staff.SetLocalBaseLine(...)

Description : The value Staff.BaseLine (page 270) allow you to define the general base line of the staff,relatively the top of its area.
With Staff.SetLocalBaseLine you can adjust localy the base line of a staff in a system for example. This will only affect the base line of the staff in this system.
Parameters : Number : Local staff base line Y position (in 72 DPI pixels) relatively to the top of the staff area.
Number: First bar to apply
Number: Last bar to apply
Return value : None

Staff.GetLocalBaseLine(...)

Description : This will provide the local base line of staff in a bar range.
Parameters : Number: First bar
Number: Last bar
Return value : Number : Local base line (in 72 DPI pixels), nil if the value can't be extracted

Staff.SetLocalChordYPosition(...)

Description : The value Staff.ChordYPosition (page 260) allow you to define the general chord line vertical positi relatively to the top of the staff area.
With Staff.SetLocalChordYPosition you can adjust localy the chord line vertical position of a staff in a system for example. This will only affect the chord line vertical position of the staff in this system.
Parameters : Number : Chord line vertical position (in 72 DPI pixels) relatively to the top of the staff area.
Number: First bar to apply
Number: Last bar to apply
Return value : None

Staff.GetLocalChordYPosition(...)

Description : This will provide the chord line position inside a staff for a bar range.
Parameters : Number: First bar
Number: Last bar
Return value : Number : Chord line vertical position (in 72 DPI pixels), nil if the value can't be extracted

Staff.SetLocalLyricsYPosition(...)

Description : The value Staff.LyricsYPosition (page 257) allow you to define the general lyrics line vertical positic relatively to the top of the staff area.
With Staff.SetLocalLyricsYPosition you can adjust localy the lyrics line vertical position of a staff in a system for example. This will only affect the lyrics line vertical position of the staff in this system.
Parameters : Number : Lyrics line vertical position (in 72 DPI pixels) relatively to the top of the staff area.
Number: First bar to apply
Number: Last bar to apply
Return value : None

Staff.GetLocalLyricsYPosition(...)

Description : This will provide the lyrics line position inside a staff for a bar range.
Parameters : Number: First bar
Number: Last bar
Return value : Number : Lyrics line vertical position (in 72 DPI pixels), nil if the value can't be extracted

Staff.SetLocalBarNumberingYPosition(...)

Description : The value Staff.BarNumberingYPosition (page 255) allow you to define the bar number vertical position, relatively to the top of the staff area.
With Staff.SetLocalBarNumberingYPosition you can adjust bar number vertical position of a staff in a system for example. This will only affect the bar number vertical position of the staff in this system.
Parameters : Number : Bar number vertical position (in 72 DPI pixels) relatively to the top of the staff area.
Number: First bar to apply
Number: Last bar to apply
Return value : None

Staff.GetLocalBarNumberingYPosition(...)

Description : This will provide the bar number vertical position inside a staff for a bar range.
Parameters : Number: First bar
Number: Last bar
Return value : Number : Bar number vertical position (in 72 DPI pixels), nil if the value can't be extracted

Staff.SetLocalLyricLineVisibility(...)

Description : Enable/disable the lyric line visibility, for a bar range. For example, this allow to hide unused lyrics line inside a system.
Note: the lyrics inside the line are not deleted, just not displayed. The following lines are shifted on the score.
Parameters : Number : Lyric line number (from 1 to 32)
Boolean: Visibility. True, the line will be visible, false the line will be invisible.
Number: First bar to apply
Number: Last bar to apply
Return value : None

Staff.GetLocalLyricLineVisibility(...)

Description : This will provide the lyric line visibility inside a staff for a bar range.
Parameters : Number: Line number from 1 to 32
Number: First bar
Number: Last bar
Return value : Boolean, true if the line is visible.

Staff.SetLocalLeftMargin(...)

Description : This will adjust the left margin of the system which own the staff, if the staff is the first displayed in this system.
Parameters : Number : Left margin, positive or negative (in 72 DPI pixels) relative to the left page margin
Number: First bar to apply
Number: Last bar to apply
Return value : None

Staff.GetLocalLeftMargin(...)

Description : This will provide the left margin of the system which own the staff if the staff is the first displayed in this system.
Parameters : Number: First bar
Number: Last bar
Return value : Number : Offset from the left margin (in 72 DPI pixels), nil if the value can't be extracted

Staff.SetLocalRightMargin(...)

Description : This will adjust the right margin of the system which own the staff, if the staff is the first displayed in this system and if the print setup ask to justify bars.
Parameters : Number : Right margin, positive or negative (in 72 DPI pixels) relative to the right page margin
Number: First bar to apply
Number: Last bar to apply
Return value : None

Staff.GetLocalRightMargin(...)

Description : This will provide the right margin of the system which own the staff if the staff is the first displayed in this systems.
Parameters : Number: First bar
Number: Last bar
Return value : Number : Offset from the right margin (in 72 DPI pixels), nil if the value can't be extracted

Staff.SetLocalTopMargin(...)

Description : This will adjust the top margin of the system which own the staff, if the staff is the first displayed in this system and if the system is the first of the page.
Parameters : Number : Top margin, positive or negative (in 72 DPI pixels).n
Number: First bar to apply
Number: Last bar to apply
Return value : None

Staff.GetLocalTopMargin(...)

Description : This will provide the top margin of the system which own the staff if the staff is the first displayed in this system and if the system is the first of the page.
Parameters : Number: First bar
Number: Last bar
Return value : Number : Offset (in 72 DPI pixels), nil if the value can't be extracted

4.3.2.2.8. Managing chord diagrams

Staff.AddDiagram(...)

Description : Adds a Diagram object to the staff
Parameters : All parameters are optional
Number : Time position. Default: 0
Number : Y position, in 72 DPI pixel relatively to the staff base line. Default:0
Return value : Diagram (page 363) : added object or nil if not added
Example :
Adding a Diagram in a staff

```
Include "MSDefine"
-- Adds a Diagram in the first staff of the frontmost score
myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in the document")
   else
      -- Adds a new Diagram at the beginning of 2nd bar
      newTempo=myStaff.AddDiagram(DURATION_WHOLE,-40,DURATION_QUARTER)
   end
end
```

4.3.2.2.9. Managing chord line

Staff.GetChordIndexAtTime(...)

Description : Returns the chord index in chord line at the given time position.
Parameters : Number : written time position
Return value : Number : Chord index in the chord line
or nil if no chord line for this staff, or no chord at this time position
Example :

```
index=myStaff.GetChordIndexAtTime(timePosition)
```

Note : Time position is the **written** time position in score. It means it follows a linear progression from the beginning of the written tune (bar 1) to theend. If break symbols (repeat barlines, parts, jumps, etc) are present in the score, they are not taken into account.

Staff.GetChordIndexAtBar(...)

Description : Returns the first chord index in chord line in the given written bar
Parameters : Number : bar number
Return value : Number : Chord index in the chord line
or nil if no chord line for this staff, or no chord at this bar
Example :

```
index=myStaff.GetChordIndexAtBar(barNumber)
```

Note : Bar number is the **written** bar in score. It means it follows a linear progression from the beginning of the written tune (bar 1) to theend. If break symbols (repeat barlines, parts, jumps, etc) are present in the score, they are not taken into account.

Staff.GetChordTime(...)

Description : Returns the indexed chord time position
Parameters : Number : chord index in chord line
Return value : Number : time position
or nil if the chord index cannot be found in this staff
Example :

```
timePosition=myStaff.GetChordTime(chordIndex)
```

Staff.GetChordBar(...)

Description : Returns the indexed chord bar number
Parameters : Number : chord index in chord line
Return value : Number : bar number
or nil if the chord index cannot be found in this staff
Example :

```
barNumber=myStaff.GetChordBar(chordIndex)
```

Staff.GetChordName(...)

Description : Gets the indexed chord native name
Parameters : Number : chord index in chord line
Return value : String : Chord name, in native format (depends on the Anglo-Saxon/Latin setting in Global setup)
or nil if the chord index cannot be found in this staff
Example :

```
chordName=myStaff.GetChordName(chordIndex)
```

Staff.SetChordName(...)

Description : Sets the indexed chord native name
Parameters : Number : chord index in chord line
String : chord name to set, in native format (depends on the Anglo-Saxon/Latin setting in Global setup)
Return value : None
Example :

```
myStaff.SetChordName(chordIndex,chordName)
```

Note : For Diminished 5th Chords (notated "-5"), the dash must be replaced by a "=" prior calling this method, in order not to be considered as a separator. You can use for example: nameToSet=gsub(name,"-","=")

Staff.GetChordDiagram(...)

Description : Gets the indexed chord diagram
Parameters : Number : chord index in chord line
Return value : Diagram (page 363)
Example :

```
local staff
local diagram

score=FrontScore()
staff=score.FirstStaff
for i=1,999 do
   diagram=staff.GetChordDiagram(i)
   if diagram~=nil then
      print("First case:".. diagram.FirstCase)
      for j=1,diagram.NumberOfFingers do
         case,string1,string2=diagram.GetFinger(j)
         print("finger "..case.." from string "..string1.." to "..string2)
      end
   print(diagram.StringValue)
   end
end
```

Staff.AddChordDiagram(...)

Description : Adds a new diagram object to the chord line of the staff
Parameters : Optional : Chord name (string)
Return value : Diagram (page 363) : object or nil if not added

Staff.GetChordDiagramFromName(...)

Description : Search in the local chord diagrams the first chord diagram with this name
Parameters : Chord name (string)
Return value : Diagram (page 363) : object or nil if not found

Staff.GetChordDiagramFromIndex(...)

Description : Provide the chord diagram from chord diagram database related to the staff
Parameters : Number : index (1 based)
Return value : Diagram (page 363) : object or nil if not found
Example :

```
local staff
local diagram

score=FrontScore()
staff=score.FirstStaff
for i=1,staff.NumberOfChordDiagrams do
   diagram=staff.GetChordDiagramFromIndex(i)
   print(diagram.StringValue)
end
```

Staff.DeleteChordDiagrams(...)

Description : Delete all locals chord diagrams related to the chord line.
Parameters : None
Return value : None

Staff.DuplicateChordDiagrams(...)

Description : Duplicate all local chords diagrams related to a chord line to another staff.
Local chords diagrams already related to target staff are not affected.
You can use Staff.DeleteChordDiagrams(...) (above) to clear the chords diagram.
Parameters : Target staff
Return value : None

4.3.2.2.10. Managing Dynamics/Tempo marks

Staff.AddDynamic(...)

Description : Adds a dynamic/tempo object
Parameters : All parameters are optional
Number : type (see constants in MSDefine (page 100).mys). Default : DYNAMIC_TEMPO
Number : Time position. Default: 0
Number : Y position, in 72 DPI pixel relatively to the staff base line. Default:0
Number : First parameter (note duration if type is DYNAMIC_TEMPO). Default: DURATION_QUARTER
Number : Second parameter (tempo value if type is DYNAMIC_TEMPO). Default: 120
Return value : Dynamic: added object or nil if not added
Example :

```
Include "MSDefine"
-- Adds a tempo change in the first staff of the frontmost score
myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in the document")
   else
      -- Adds a new tempo change of quarter=122.5 at the beginning of 2nd bar
      newTempo=myStaff.AddDynamic(DYNAMIC_TEMPO,DURATION_WHOLE,-40,DURATION_QUARTER,
      122.5)
   end
end
```

4.3.2.2.11. Managing FX processors

Staff.AddFXProcessor(...)

Description : Adds a new FXProcessor
Parameters : All parameters are optional
Number : Time position
Number : Y position, in 72 DPI pixel relatively to the staff base line
Return value : FXProcessor: added object or nil if not added
Example :

```
Include "MSDefine"
-- Adds an FX processor in the first staff of the frontmost score
myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in the document")
   else
      -- Adds a new FX Processor
      newFx=myStaff.AddFXProcessor(DURATION_WHOLE,-20)
      if newFx~=nil then
         -- Changes its name
         newFx.Title="My new FX"
      end
   end
end
```

4.3.2.2.12. Managing lyric lines

Staff.InsertLineInLyrics(...)

Description : Insert a blank line before a lyric line.
Parameters : Number : Start measure (1 based)
Number : End measure (1 based)
Number : first line to shift (1 based)
Return value : none
Example :

```
-- Shift lyrics line on a score
myScore=FrontScore()
if myScore==nil then
    print("Please load a score first")
else
    myStaff=myScore.FirstStaff
    if myStaff==nil then
        print("There is no staff in this score")
    else
        --We preserve the current score state,
        --so the user will be able to undo the changes.
        myScore.Preserve()
        myStaff.InsertLineInLyrics(2,5,2)
    end
end
```

Staff.HideLineInLyrics(...)

Description : Hide the lyric line
Parameters : Number : Start measure (1 based)
Number : End measure (1 based)
Number : Line to hide (1 based)
Return value : none

Staff.IsHideLineInLyrics(...)

Description : Tells if a lyrics cell is hidden
Parameters : Number : measure (1 based)
Number : Line to test (1 based)
Return value : true if lyrics are hidden at this measure and line number

Staff.CompactLinesInLyrics(...)

Description : For the given measures range, compact all lyrics lines.
Parameters : Number : Start measure (1 based)
Number : End measure (1 based)
Return value : none

Staff.UnHideLinesInLyrics(...)

Description : For the given measures range, show all hidden lyrics lines.
Parameters : Number : Start measure (1 based)
Number : End measure (1 based)
Return value : none

4.3.2.2.13. Managing lyrics

Staff.GetLyricsCellIndexAtTime(...)

Description : Returns the lyrics index in lyrics line at the given time position.
Parameters : Number : played time position
Return value : Number : lyric index from 1 to n
or nil if no lyrics for this staff, or no lyrics at this time position
Example :

```
index=myStaff.GetLyricsCellIndexAtTime(timePosition)
```

Note : Time position is the **played** time position in score. It means it starts from the first played bar and follows a linear progression along the played locations.
If break symbols (repeat barlines, parts, jumps, etc) are present in the score, they are taken into account.
For example, if the first bar is repeated three time, bar 2 of the written score will be considered to be bar 4 according to the played location.

Staff.GetLyricsCellTime(...)

Description : Returns the played time of the indexed lyrics element.
Parameters : Number : lyric index from 1 to n
Return value : Number : lyrics played time position
or nil if index is outside the index range for this staff
Note : Time position is the **played** time position in score. It means it starts from the first played bar and follows a linear progression along the played locations.
If break symbols (repeat barlines, parts, jumps, etc) are present in the score, they are taken into account.
For example, if the first bar is repeated three time, bar 2 of the written score will be considered to be bar 4 according to the played location.

Staff.GetLyricsCellContent(...)

Description : Returns the content (string) of the indexed lyrics element.
The returned string contains the raw string for this part of lyrics, i.e. it includes unbreakable spaces (^), unbreakable hyphens (=) and the miscellaneous commands (brackets) that can have been entered.
Take also a look to Staff.GetLyricAtTime(...) (page 283) which is a faster way to do a similar thing.
Parameters : Number : lyrics index
Return value : Styled string (page 177): lyrics
or nil if index is outside the index range for this staff
Example :

```
-- Display the lyrics of the first staff of the front score
-- in  a text window
local score
local staff
local str
local index

-- Creating text window
MyTextWindow=NewTextWindow("Lyrics")
--Extracting score
score=FrontScore()
if(score~=nil) then
   --Extracting staff
   staff=score.FirstStaff
   if(staff~=nil) then
      index=1
      --Extracting lyrics until end of score
      repeat
         str=staff.GetLyricsCellContent(index)
         if(str~=nil) then
            MyTextWindow.Write(PurgeStringStyle(str),false)
            index=index+1
         end
      until str==nil
   end
else
   Message("Load a score please.")
end
```

Staff.SetLyricsCellContent(...)

Description : Sets the content (string) of the indexed lyrics element.
The string parameter contains the raw string for this part of lyrics, i.e. it can include unbreakable spaces (^), unbreakable hyphens (=) and the miscellaneous commands (brackets) for Virtual Singer.
The string MUST NOT include syllable separator characters (space, hyphen, underline), because it would shift the next syllables in the bar.
The last character MUST be such a character in order to separate the syllable from the next one.
Parameters : Number : lyric index from 1 to n
Styled string (page 177) : string to set
Return value : Boolean: true if successfully set, false otherwise.
Example :

```
-- Replace the lyrics of the first staff of the front score
-- With the fragments
local score
local staff
local index

--Extracting score
score=FrontScore()
if(score~=nil) then
   score.Preserve()
   --Extracting staff
   staff=score.FirstStaff
   if(staff~=nil) then
      index=1
      lyrics=staff.CreateLyricsArray()
      if(lyrics) then
         --Extracting lyrics until end of score
         repeat
            if(lyrics[index]~=nil) then
               if(lyrics[index].Lyric~="") then
                  print(lyrics[index].Lyric.." ("..lyrics[index].Fragment..")")
                  staff.SetLyricsCellContent(index,"[#"..lyrics[index].Fragment.."]")
               end
               index=index+1
            end
         until lyrics[index]==nil
      else
               Message("There is no lyrics line on the first staff of this score.")
      end
   end
else
   Message("Load a score please.")
end
```

Staff.GetLyricAtTime(...)

Description : Returns the content (string) of the lyric at a written time position and lyric line.
The returned string contains the raw string for this part of lyrics, i.e. it includes unbreakable spaces (^), unbreakable hyphens (=) and the miscellaneous commands (brackets) that can have been entered.
This is a faster way to extract lyrics than GetLyricsCellContent.
However there is a difference, when lyrics are empty because repeated from the previous line, GetLyricAtTime returns an empty string.
Parameters : Number : Written Time In general, the time position of a note.
Optional : Line number from 1 to n. By default 1.
Optional : Boolean which indicate if special sequences between brackets will be purged or not. (false by default)
Return value : Styled string (page 177): lyrics played at this time position
Empty string if no lyric at this time position
Example :

```
-- Display the lyrics of the first staff of the front score
-- in  a text window
local score
local staff
local str
local line,nbLines

--Extracting score
score=FrontScore()
if(score~=nil) then
   --Extracting staff
   staff=score.FirstStaff
   if staff~=nil then
      nbLines=staff.LyricsNumberOfLines
      for line=1,nbLines do
         symbol=staff.FirstSymbol
         while symbol~=nil do
            --- Extracting from notes only (there is no data related to rests and this can provide bad data)
            if symbol.IsNote==true then
               str=staff.GetLyricAtTime(symbol.Time,line,true)
               if str~="" then print("Line:"..line.." '"..str.."'") end
            end
            symbol=symbol.Next
         end
      end
   end
else
   Message("Load a score please.")
end
```

Staff.SetLyricAtTime(...)

Description : Set the content (styled string (page 177)) of the lyric at a written time position and lyric line.
This is a faster way to set lyrics than SetLyricsCellContent.
In general, the time position is the time position of a note.
After all changes have been made to lyrics through this method, Staff.RemoveUnusedLyrics(...) (page 287) must called, to get rid of unused words, recompute lyrics cell height and various other display settings.
Parameters : Number : Written Time
Styled string (page 177): string to set
Optional : Line number from 1 to n. By default 1.
Return value : None

Staff.GetLyricsAtBar(...)

Description : Returns the entire content (string) of the lyric at a given measure.
The returned string contains the raw string for this part of lyrics, i.e. it includes unbreakable spaces (^), unbreakable hyphens (=) and the miscellaneous commands (brackets) that can have been entered.
Parameters : Number : Measure number
Optional : Line number from 1 to n. By default 1.
Return value : Styled string (page 177): lyrics played for this bar
Empty string if no lyric at this bar
Example :

```
score=FrontScore()
staff=score.FirstStaff
for i=1,score.NumberOfBars do
   for j=1,staff.LyricsNumberOfLines do
      str=staff.GetLyricsAtBar(i,j)
      if str~="" then
         print(str)
      end
   end
end
```

Staff.SetLyricsAtBar(...)

Description : Set the entire content (styled string (page 177)) of the lyric at a given measure.
After all changes have been made to lyrics through this method, Staff.RemoveUnusedLyrics(...) (page 287) must called, to get rid of unused words, recompute lyrics cell height and various other display settings.
Parameters : Number : Measure number
Styled string (page 177) : string to set
Optional : Line number from 1 to n. By default 1.
Return value : None

Staff.CreateLyricsArray(...)

Description : Copy many informations about lyrics in an array for fastest and more precise access.
As you saw, methods described in this section allow you to access individualy to texts related to a time position. When you use method Staff.GetLyricsCellContent(...) (page 282), Staff.GetLyricsCellTime(...) (page 282), Staff.GetLyricsCellIndexAtTime(...) (page 281) the entire lyrics are computed for the staff. This is slow and don't allow you to respond quickly, for example if you want to display lyrics while music is playing.
The method Staff.CreateLyricsArray perform the computation one time (this can take some seconds) and provide an array where each element describe the lyrics played at a time position.
Much more, this method provide you informations like the Sampa notation for the lyrics.
The array begin at index 1. You can browse all the items until the returned value is nil (end of array). Items are sorted by ascending order of played time position.
Each item in the array provide :

- Time : the played time position.
- Lyric : the lyric text (styled string (page 177))
- Sampa : the Sampa notation for this lyric.
- Fragment : the fragment list for this lyric as "+aa+bb+cc" where aa,bb,cc are the fragments names.

Note 1 : As the array is a copy of the current lyrics related to the staff, if you apply changes to the lyrics, you ought to recompute the array.
*Note 2 : This method need to access to the Virtual Singer data, so an error message will be displayed if you use it on a computer where Virtual Singer is **not** installed. You can use Application.IsVirtualSingerInstalled (page 182) check if Virtual Singer is installed before calling the method.*
Parameters : None
Return value : An array.
Example :

```
-- Display the lyrics of the first staff of the front score
-- in  a text window
-- Demontration of "Lyric" field use
local score
local staff
local str
local index
local lyrics

-- Creating text window
MyTextWindow=NewTextWindow("Lyrics")
--Extracting score
score=FrontScore()
if(score~=nil) then
   --Extracting staff
   staff=score.FirstStaff
   if(staff~=nil) then
      index=1
      lyrics=staff.CreateLyricsArray()
      if(lyrics) then
         --Extracting lyrics until end of score
         repeat
            if(lyrics[index]~=nil) then
               MyTextWindow.Write(lyrics[index].Lyric,false)
               index=index+1
            end
         until lyrics[index]==nil
      else
         Message("There is no lyrics line on the first staff of this score.")
      end
   end
else
   Message("Load a score please.")
end
```

```
-- Display the lyrics and the sampa of the first staff of the front score
-- Demontration of "Sampa" field use
local score
local staff
local index

--Extracting score
score=FrontScore()
if(score~=nil) then
   --Extracting staff
   staff=score.FirstStaff
   if(staff~=nil) then
      index=1
      lyrics=staff.CreateLyricsArray()
      if(lyrics) then
         --Extracting lyrics until end of score
         repeat
            if(lyrics[index]~=nil) then
               if(lyrics[index].Lyric~="") then
                  print(lyrics[index].Lyric.." ("..lyrics[index].Sampa..")")
               end
               index=index+1
            end
         until lyrics[index]==nil
      else
         Message("There is no lyrics line on the first staff of this score.")
      end
   end
else
   Message("Load a score please.")
end
```

```
-- Display the played sampa
-- Demontration of "Time" field use
local score
local staff
local index
```

```
--Extracting score
score=FrontScore()
if(score~=nil) then
   --Extracting staff
   staff=score.FirstStaff
   if(staff~=nil) then
      index=1
      lyrics=staff.CreateLyricsArray()
      if(lyrics) then
         score.PlayMusic()
         --Extracting lyrics until end of score
         repeat
            Application.UserBreak()
            time=score.CurrentPlayedTime
            while(lyrics[index].Time<time) do
               if lyrics[index].Sampa ~= "" then
                  print("Sampla played is : "..lyrics[index].Sampa)
               end
               index=index+1
            end
         until lyrics[index]==nil
      else
         Message("There is no lyrics line on the first staff of this score.")
      end
   end
else
   Message("Load a score please.")
end
```

Staff.RemoveUnusedLyrics(...)

Description : Remove all unused lyrics : lyrics not related with a played bar. Lyrics cell height is also computed, as well as other display settings.
Parameters : None
Return value : none

4.3.2.2.14. Managing Pictures

Staff.AddPicture(...)

Description : Adds a Picture object to the staff
Parameters :
String : Full path on the picture file. The file can be a BMP,PICT,GIF,JPEG,PNG,TIFF.
Optional:
Number : Time position. Default: 0
Number : Y position, in 72 DPI pixel relatively to the staff base line. Default:0
Boolean : True if the horizontal size is independant from the bar scale, false if the horizontal size of the picture is related to the bar scale (true by default)
Boolean : True if the background is erased before the picture is draw (false by default)
Number : Horizontal scale from 1 to 1000 (by default 100)
Number : Vertical scale from 1 to 1000 (by default 100)
Number : Index of the color to apply (by default 0)
Number : Rotation (only used if the source is a graphic file other than PICT)
Return value : Picture (page 358) : added object or nil if not added
Example :

```
Include "MSDefine"
-- Adds a picture in the first staff of the frontmost score
myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in the document")
   else
      --- Selecting the file
      myFileName=GetFileNameLoad("bmp,pct")
      if(myFileName~=nil) then
         -- Adds a new picture at the beginning of 2nd bar
         newPicture=myStaff.AddPicture(myFileName,DURATION_WHOLE,-40)
      end
   end
end
```

Staff.BeginRecord(...)

Description : Beginning the recording of a picture. All following graph methods will be recorded in this picture. Only one recording can be active at the same time. All recording must be close using the Staff.EndRecord(...) (pa 289) method.
If you want to get the current graphic context where the graphic commands are recorded, you can use Staff. GraphRecord.
This is particulary useful if you want to call a function in a floating window using dialog.functionName because this kind of call set the current graphic context to the dialog graphic context.
In this case you just have to call the function with Staff.GraphRecord as parameter and use the Graph object.
Parameters : Optional :
- Number : horizontal size of the picture, in pixel (by default 100)
- Number : vertical size of the picture, in pixel (by default 100)

Return value : Boolean : true if the recording was inited, false if it failed (for example a record picture is pending)

Example :

```
Include "MSDefine"
-- Create a picture and add it to the first staff of the frontmost score
myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in the document")
   else
      --- Beginning the recording process
      myStaff.BeginRecord(64,64)
      --- Draw the picture
      Graph.FrameRect(0,0,64,64);
      Graph.PaintRect(8,8,64-8,64-8);
      Graph.EraseRect(16,16,64-16,64-16);
      --- Close the recording and add the picture
      myStaff.EndRecord(100,-40)
   end
end
```

Staff.EndRecord(...)

Description : End the recording of a picture and add the resulting picture to the staff
Parameters : Optional :
Optional:
Number : Absolute pixel position of the left of the picture. Default: 0
Number : Y position, in 72 DPI pixel relatively to the staff base line. Default:0
Boolean : True if the horizontal size is independant from the bar scale, false if the horizontal size of the picture is related to the bar scale (true by default)
Boolean : True if the background is erased before the picture is draw (false by default)
Number : Horizontal scale from 1 to 1000 (by default 100)
Number : Vertical scale from 1 to 1000 (by default 100)
Number : Index of the color to apply (by default 0)
Return value : Picture (page 358) : added object or nil if not added

Staff.AddGraphicLine(...)

Description : Adds a graphical line
Parameters :
Optional:
Number : Kind of the line : SOLID_LINE, DASHED_LINE, DOTTED_LINE,WAVY_LINE,SOLID_CURVE, DOTTED_CURVE. By default SOLID_LINE
Number : Kind of beginning of line : END_NONE, END_JOG_UP, END_JOG_DOWN, END_JOG_BOTH, END_ARROW. By default END_NONE
Number : Time position of the beginning of the line. Default: 0
Number : Y position of the beginning of the line, in 72 DPI pixel relatively to the staff base line. Default: 0
Number : Kind of end of line : END_NONE, END_JOG_UP, END_JOG_DOWN, END_JOG_BOTH, END_ARROW. By default END_NONE
Number : Time position of the end of the line. Default: 0
Number : Y position of the end of the line, in 72 DPI pixel relatively to the staff base line. Default: 0
Number : Line width in 72 dpi. Default: 1.
Number : Bracket height. Default: 8
Number : Index of the color to apply (by default 0)
Return value : Picture (page 358) : added object or nil if not added

4.3.2.2.15. Managing rules

Staff.AddRule(...)

Description : Adds a new rule in the rule list
Parameters : Optional : Boolean true if the rule will be unique. The rule parameters will be computed to be different of all other rules. By default false.
Optional : SoundID of the instrument which will be related to this rule. If the instrument don't exist it will be created.
Return value : Rule : added object or nil if not added
Example :

```
-- Adds a rule in the first staff of the frontmost score
myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in the document")
   else
      -- Inserts a new rule
      rule=myStaff.AddRule()
      if rule~=nil then
         -- Changes its name
         rule.Title="My rule"
      end
   end
end
```

4.3.2.2.16. Managing Staves

Staff.Duplicate(...)

Description : Duplicates the staff.
Parameters : Optional : Boolean is true if the staff is created as a volatile staff. This staff will be deleted when the script will end.
Return value : Staff : New staff, which is a copy of the source staff
Example :

```
-- Duplicates the first staff of the frontmost score
myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in the document")
   else
      myStaff2=myStaff.Duplicate()
      -- duplicate instrument
      myStaff2.Instrument=myStaff.Instrument.Duplicate()
   end
end
```

Note : The instrument for the staff is not duplicated. Any operation on this instrument will affect both staves. If you need an independent instrument, use Instrument.Duplicate()

Staff.MoveBefore(...)

Description : Changes the relative order of staves in the score.
The current staff is graphically moved just before (above) the staff in parameter
Moving a staff before itself (or the next one) does nothing.
Parameters : Staff : staff to be moved before
if nil or omitted, moves after the last staff in the score.
Return value : None
Example :

```
-- Moves the first staff of the frontmost score at the end
myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in the document")
   else
      myStaff.MoveBefore(nil)
   end
end
```

4.3.2.2.17. Managing symbols

Staff.OverlayNote(...)

Description : Adds a new note at the required time position, without moving notes around.
As a result, the inserted note will be placed at the exact time position you specified, and some rests might have been added in chord with the previous note in order not to move other notes in the staff.
Special consideration : only notes (not rests) can be overlayed
In the provided example below, a new note will be added
Parameters : Number : Time position
Number : Symbol duration, if 0 the current selected duration in note duration palette
Following parameters are optional
Number : Note pitch
Boolean : Insert mode.
[.] If true (default): when adding a note,
- rests in chord might be deleted if useless
- uncomplete bars are filled in with true rests

it results in an automatically good-looking score, but during the insert, some existing rests can be deleted in order to keep the score simple
[.] If false: when adding a note,
- as far as possible, existing rests are kept as is
- uncomplete bars are filled with ghost rests

This mode is closer to a manual insert, with less rest deletion, and has to be used when you exactly know how the result should look like.

Return value : Symbol : Added note
or nil if the note could not been added.
Example :

```
note1=myStaff.InsertNote(0,DURATION_QUARTER,5*12) -- Insert a quarter C at the beginning of the staff
if note1~=nil then
   note2=myStaff.OverlayNote(note1.Time+DURATION_EIGHTH,DURATION_EIGHTH,5*12+2)
end
-- will result in : Time position 0              : A quarter C in chord with an eighth rest
--                  Time position DURATION_EIGHTH  : An eighth D
```

Staff.InsertNote(...)

Description : Inserts a new note just after the symbol that precedes the given time position.
If there is already a symbol at the given time position, the new note is inserted in chord with this symbol
If the time position lies between two symbols, the symbol will be inserted after the whole duration of the previous one, and will shift the next symbol to the right of the score.
In fact, InsertNote mimics the way a symbol is inserted when you click between two notes on a score, using the note insert tool.
Parameters : Number : Time position
Number : Symbol duration, if 0 the current selected duration in note duration palette
Following parameter are optional
Number : Note pitch
Boolean : true if insert operation is limited to one bar, false otherwise
Special Considerations :
·When the last parameter is true, the insert operation behaves like a manual symbol insert when "Options> Limit editing to one bar" setting is on.
·When the last parameter is false, the insert operation behaves like a manual symbol insert when "Options> Limit editing to one bar" setting is off.
·When the last parameter is omitted, the insert operation behaves like a manual symbol insert, using the current setting in "Options>Limit editing to one bar"
Return value : Symbol : Added note
or nil if the note could not been added.
Example :

```
note1=myStaff.InsertNote(0,DURATION_QUARTER,5*12,false) -- Insert a quarter C at the beginning of the staff
if note1~=nil then
   note2=myStaff.InsertNote(note1.Time+DURATION_EIGHTH,DURATION_EIGHTH,5*12+2,false)
end
-- will result in : Time position 0              : A quarter C
--                  Time position DURATION_QUARTER : An eighth D
```

Staff.InsertRest(...)

Description : Inserts a rest just after the symbol that precedes the given time position.
If there is already a symbol at the given time position, the new rest is inserted in chord with this symbol.
If the time position lies between two symbols, the rest will be inserted after the whole duration of the previous symbol, and will shift the next symbol to the right of the score.
In fact, InsertRest mimics the way a rest is inserted when you click between two notes on a score, using the rest insert tool.
Parameters : Number : Time position
Number : Rest duration, if 0 the current selected duration in note duration palette
Optional :
Boolean : true if insert operation is limited to one bar, false otherwise
Special Considerations :
- When the last parameter is true, the insert operation behaves like a manual symbol insert when "Options> Limit editing to one bar" setting is on.
- When the last parameter is false, the insert operation behaves like a manual symbol insert when "Options> Limit editing to one bar" setting is off.
- When the last parameter is omitted, the insert operation behaves like a manual symbol insert, using the current setting in "Options>Limit editing to one bar"

Return value : Symbol : Added note
or nil if the note could not been added.
Example :

```
note1=myStaff.InsertNote(0,DURATION_QUARTER,5*12) -- Insert a quarter C at the beginning of the staff
if note1~=nil then
    note2=myStaff.InsertRest(note1.Time+DURATION_EIGHTH,DURATION_EIGHTH)
end
-- will result in : Time position 0                : A quarter C
--                  Time position DURATION_QUARTER : An eighth rest
```

Staff.GetIndexedSymbol(...)

Description : Gets a symbol from its position in the staff.
Through this method, you can get the Nth symbol in a staff.
Parameters : Number : Symbol index in the staff
Return value : Symbol : Note at this index
or nil if there is no note at this index (index outsite a range from 1 to Staff.NumberOfSymbols (page 264))
Example :

```
-- Displays pitch of 5th symbol
myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in the document")
   else
      symbol=myStaff.GetIndexedSymbol(5)
      if symbol~=nil then
         print("5th symbol is : ",symbol.Name)
      else
         print("There is no 5th symbol in 1st staff");
      end
   end
end
```

Staff.DeleteAllSymbols(...)

Description : Clears the staff of all its symbols.
Parameters : None
Return value : None
Example :

```
-- Deletes all symbols in the first staff of the frontmost score
myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in the document")
   else
      myStaff.DeleteAllSymbols()
   end
end
```

Staff.TimeToSymbol(...)

Description : Returns the first symbol whose time position is greater or equal to the required position.
Parameters : Number : Time position
Return value : Symbol : First symbol whose time position is greater or equal to the required position.
Example :

```
      mySymbol=myStaff.TimeToSymbol(DURATION_WHOLE)
      -- Returns the first note in bar 2 (if time signature is 4/4)
```

4.3.2.2.18. Managing texts

Staff.AddText(...)

Description : Inserts a new text
Parameters : All parameters are optional
Number : Text type : COMMENT_STAFF_TEXT=0, LYRICS_STAFF_TEXT=1, CHORD_STAFF_TEXT=2
String : Text content
Number : Left of the text area (time position)
Number : Top of the text area (72 DPI pixel relatively to the staff base line)
String : text font
Number : text size
String : text face
Return value : StaffText (page 382) : StaffText object or nil if not added
Example :

```
Include "MSDefine.mys"
-- Inserts a text object in the first staff of the frontmost score
myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in the document")
   else
      -- Inserts a new text
      newText=myStaff.AddText(COMMENT_STAFF_TEXT,"Hello",DURATION_WHOLE,-40)
      if newText~=nil then
         -- Changes its color to blue
         newText.RGBColor="0000FF"
      end
   end
end
```

4.3.2.2.19. Managing transposing instruments

Staff.ApplyTransposingInstrument(...)

Description : Apply a transposing instrument to a whole staff.
This method changes the key shift of the instrument, as well as all the key signatures on the staff.
Existing notes can be changed according to the input parameter.
Parameters : Number : semitone offset (number of semitone that will be added to the written note when playing)
Number: Transposing type
Can be :
· TRANSPOSE_UPWARD
· TRANSPOSE_DOWNWARD
· TRANSPOSE_DISPLAY_UNCHANGED
Return value : None.
Note : To cancel the effect of an already existing transposing instrument on a staff, use this method with a semitone of 0 and a transposing type of TRANSPOSE_UPWARD.

4.3.2.2.20. Merging Staves

Staff.MergeWithNext(...)

Description : Merge the staff with the next staff
Parameters : None
Return value : Boolean : true if the staff was successfuly merged, false otherwise.

Staff.Unmerge(...)

Description : Unmerge the staff with the next staff
Parameters : None
Return value : Boolean : true if the staff was successfuly unmerged, false otherwise.

4.3.2.2.21. Miscellaneous

Staff.ComputeStemsDirection(...)

Description : All stems direction of the notes owned but the staff will be computed.
Parameters : None
Return value : none

Staff.ComputePitchDisplayedHeight(...)

Description : From a pitch and a time position, compute the note position in interline and the accidental.
Parameters : Number: pitch
Number: time position
Return value : Number: interline from baseline
Number: accidental

Staff.ComputeNotesCaptionAlongRules(...)

Description : From a pitch and a time position, compute the note position in interline and the accidental.
Parameters : Number: pitch
Number: time position
Return value : Number: interline from baseline
Number: accidental

4.3.2.2.22. Page setup

Staff.GetPage(...)

Description : From a given staff and written bar, provides the page number where this bar will be printed/displayed.
Parameters : The bar number from 1 to the number of written bar.
Return value : The page number from 1 to the number of pages or 0 if error.
Example :

```
score=FrontScore()
staff=score.FirstStaff
for numbar=1,score.NumberOfBars do
   print("Bar # "..numbar.." is in page "..staff.GetPage(numbar))
end
```

4.3.2.2.23. Playing notes

Staff.PlayNotes(...)

Description : Play a note or a chord whith a given duration and velocity with the instrument related to the staff.
Parameters : · Time position in the staff : this time position will be used to convert notes duration to real duration using the current tempo at this position.
You can send 0
· Duration of the chord
· Velocity of the notes
· List of pitches. You can provide up to 19 notes to be played simultenaously. A negative value or no value end the list.
Return value : None
Example :

```
myStaff.PlayNotes(0,DURATION_QUARTER,64,4*12,4*12+5,4*12+7)
```

Staff.OutputLevel(...)

Description : Provide the current output levels.
Parameters : None
Return value : Left, right, back left and back right level from 0 to 255
Example :

```
left,right,backleft,backright=myStaff.OutputLevel()
```

4.3.2.2.24. Tablature

Staff.AddTablature(...)

Description : Relate a new tablature to the staff. If a tablature is already related to the staff, you can use this method to change the tablature type, priority and standard staff display mode.
You can use the Tablature.IsActive (page 370) value to check if the tablature is already active before calling this method.
Only standard staff can have a tablature related to.
Parameters : Optional :
· Number : the tablature type
· GUITAR_TABLATURE=0 (by default)
· HARMONICA_TABLATURE=1
· ACCORDION_TABLATURE=2
· Boolean : true if priority is set to tablature, false if not. (false by default)
· Boolean : if the tablature has priority, define if the standard staff will be displayed or not. (true by default). The standard staff is always displayed if priority is not set to tablature.
Return value : True if a new tablature is related to the staff. False otherwise.
Example :

```
--- Relate a guitar tablature to the first staff of the front score
local myScore
local myStaff

myScore=FrontScore()
if(myScore~=nil) then
   myStaff=myScore.FirstStaff
   if(myStaff ~=nil) then
      myStaff.AddTablature()
   end
else
   Message("Please load a score");
end
```

Staff.SetDefaultTablature(...)

Description : Set the tablature to default value along the current tablature kind. If the user had defined custom default values, they can be applied.
If no tablature is related to the staff, do nothing
Parameters : Optional :
Boolean : true if user custom values are applied. (true by default)
Return value : None

4.3.2.2.25. Tags

Some objects on the score can be tagged with a string : Symbols, Free Objects, Pictures.
This allow the script to find later the objects. The tags are saved in the .myr file.

Staff.SearchSymbolsByTag(...)

Description : Browse the staff, search the provided tag (case sensitive) and return an array of the symbols.
Parameters : String : The tag to search
Return value : A lua table with all matching symbols.

Staff.ClearTagSymbols(...)

Description : Erase all tags related to the symbols
Parameters : None
Return value : None

Staff.SearchPicturesByTag(...)

Description : Browse the staff, search the provided tag (case sensitive) and return an array of the pictures.
Parameters : String : The tag to search
Return value : A lua table with all matching symbols.

Staff.ClearTagPictures(...)

Description : Erase all tags related to the picturesErase all tags related to the pictures
Parameters : NoneNone
Return value : NoneNone

Staff.SearchOrnamentsByTag(...)

Description : Browse the staff, search the provided tag (case sensitive) in ornaments related to notes and return an array of the ornaments.
Parameters : String : The tag to search
Return value : A lua table with all matching ornaments.

Staff.ClearTagOrnaments(...)

Description : Erase all tags related to the ornaments
Parameters : None
Return value : None

4.3.2.2.26. Virtual Singer

Staff.VirtualSingerVoiceKind(...)

Description : Provide the kind of voice related to a staff.
Parameters : None
Return value : 0 if no voice, 1 if VirtualSinger voice, 2 if RealSinger voice.

Staff.SaveVirtualSingerVoice(...)

Description : Save all the Virtual Singer settings related to this staff
A lyric line (a singer) must be related to the staff.
Parameters : String: File name to save. No extension is needed: the extension will be added following the voice kind (.rsv or .prv)
Return value : True if saved.
Example :

```
staff=FrontScore().FirstStaff
path=GetUserSettingsPathName("Voices").."MyVoice.prv"
staff.SaveVirtualSingerVoice(path)
```

Staff.LoadVirtualSingerVoice(...)

Description : Load all the Virtual Singer settings from a file and relate them to the staff.
A lyric line (a singer) must be related to the staff.
Parameters : String: File name to load
Return value : True if loaded.

4.3.3. Symbol Object Type

General description of this object type : A Symbol object is a music symbol that can be displayed on a staff.
It includes : notes, rests, ghost rests and bar repeat (%) symbols
From the symbol, depending on its type, you can manage ornaments, grace notes and slurs
Other items (clefs, keys, barlines, dynamics, etc) must be accessed through the dedicated values and methods of a Staff (page 252) object.

Note pitch

Parameter that is actually stored for a note pitch represents the way this note is PLAYED.
It means, pitch is stored in semitone, not in terms of graphical position on score.
Then additional parameters can be set to the symbol to tell the program how to display this note on score.
For example a C# will be stored 61 (5*12 = 5th octave, +1 = 1 semitone above C)
To force it to be displayed as a Db, set a Flat accidental in its "Accidental" value.
It won't change the note actual (played) pitch, but only the way note is drawn on score.
Advanced values and methods (ex : Symbol.DisplayedAccidental) enable however to force the pitch according to what we need to be displayed on score.
It is functionnally equivalent to the user injecting an accidental in a note through the injector tool.

Symbol duration

In the same way, duration is stored in an internal unit (see constants in MSDefine (page 100).mys), and the program tr to display this duration the best way it can.
You can set additional parameters (split duration, tuplets, etc) to make this note be displayed a different way.
Just keep in mind setting a 3:2 triplet setting to a note won't change the actual note duration.
It will just tell the program to DISPLAY it as a triplet, if the actual duration of the note enables this.

Time position

Time position uses the same unit than symbol duration.
For example, if a note is the 3rd note in a staff, and the two first notes are both quarters (DURATION_QUARTER), its time position will be 2*DURATION_QUARTER or DURATION_HALF
There is no gap in time position on a staff. It means a note is always positioned just after the previous one ends.
Thus, time position is a read-only value. To change time position for a note, you must delete it (or replace it by a rest) from where it is, then insert a new one where you want it to be.

Absolute pixel position

This is the absolute pixel position of the beginning of the symbol, computed as if the score was in scroll mode. The origin (0 value) is the beginning of the staff.
This take account of the bar scale, the engraver settings, the horizontal shifting of the note head. This is independant of the view scale of the score.

4.3.3.5. Symbol global functions

4.3.3.5.1. Converting

PitchToString(...)

Description : Converting a string in pitch following the note name mode.
Parameters : Number, the pitch to convert.
Optional : Boolean, note name mode, true if ABC, false if Do,Re,Mi...., by default : the current note name mode selected in the general setup.
Return value : String, the string.
Example :

```
pitch=6*12+1
Message("Pitch name of "..pitch.." is "..PitchToString(pitch))
```

StringToPitch(...)

Description : Converting a pitch in string following the current note name mode selected in the general setup.
Parameters : String, the string to convert
Optional : Boolean, note name mode, true if ABC, false if Do,Re,Mi...., by default : the current note name mode selected in the general setup.
Return value : Number, the pitch
Example :

```
myString="C#4"
Message("Pitch of "..myString.." is "..StringToPitch(myString,true))
```

PitchToFrequency(...)

Description : Convert a pitch in frequency
Parameters : Number : The pitch to convert
Return value : Number : the pitch in hertz.
Example :

```
--- This sample display the pitch of notes from C3 to C5
for pitch=StringToPitch("C3",true),StringToPitch("C5",true) do
    print("Frequency of "..PitchToString(pitch).." is "..PitchToFrequency(pitch).." Hz")
end
```

FrequencyToPitch(...)

Description : Convert a frequency in pitch.
The valid values for the pitch are from 30 hz (Do1-.49 semitones) to 4064 Hz (Si7+.49 semitones)
Parameters : Number : the value in hertz.
Return value : Number : The pitch. Nil if frequency is outside range.
Number : Shift in 100 th of semitone from the theorical value and the provided value (from -50 to 50)
Example :

```
frequency=271
pitch,shift=FrequencyToPitch(frequency)
if shift==0 then
        print(" The frequency "..frequency.." is exactly equivalent to "..PitchToString(pitch))
else
    if shift>0 then
        print(" The frequency "..frequency.." is equivalent to "..PitchToString(pitch).." plus "..(shift/100).." semitones")

    else
        print(" The frequency "..frequency.." is equivalent to "..PitchToString(pitch).." minus "..-(shift/100).."
        semitones")
    end
end
```

4.3.3.6. Symbol values

4.3.3.6.1. Browsing symbols

From a staff, you can start browsing symbol by using :Staff.FirstSymbol (page 264),Staff.LastSymbol (page 264), Staff.FirstSelectedSymbol (page 269) or Staff.LastSelectedSymbol (page 269)

Symbol.Next (R)

Type : Symbol
Description : Next symbol in this staff. nil when last symbol is reached
Example :

```
-- Displaying the time position of all the symbols
-- of the first staff of a score
local myScore
local myStaff
local mySymbol
myScore=FrontScore()
if myScore==nil then
   Message("Please load a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      Message("There is no staff in this score")
   else
      mySymbol=myStaff.FirstSymbol
      while mySymbol ~=nil do
         print(mySymbol.Time)
         mySymbol=mySymbol.Next
      end
   end
end
```

Symbol.Previous

Type : Symbol
Description : Previous symbol in this staff. nil if first symbol is reached

4.3.3.6.2. Managing note pitch

Symbol.Pitch

Type : Number
Description : Note pitch, in semitone. 60 is the pitch of the C, 4th octave. Returns nil if the symbol is not a note (rest, ghost rest or bar repeat).
This value can be set to change the note pitch.
All other values in this section are more advanced ways to get or set the Pitch value.

Symbol.PlayedPitch

Type : Number
Description : Played note pitch in semitone. 60 is the pitch of the C, 4th octave.
Differs from Symbol.Pitch by the transposing key shift given by the default instrument of the staff.
Does not manage pitch changes that could be applied through Rule (page 416)s

Symbol.Name

Type : String
Description : Note name, in "native" format (depends on Anglo-saxon/Latin preference setting)

Symbol.LatinName

Type : String
Description : Note latin name (Do, Re, ...)

Symbol.AngloSaxonName

Type : String
Description : Note Anglo-saxon name (C,D, ...)

Symbol.DiatonicDegree

Type : Number
Description : Diatonic degree (from 1 = root note to 7 = major 7th), depending on the current key signature.

Symbol.DiatonicOffset

Type : Number
Description : Semitone offset from the diatonic degree. For example, in key of Major C, Db has a diatonic degree of 2 and a diatonic offset of -1.

Symbol.DiatonicOctave

Type : Number
Description : Diatonic octave. A full octave ranges from the 1st diatonic degree to the 7th

Symbol.ChromaticDegree

Type : Number
Description : Chromatic degree (semitone offset from C).

Symbol.ChromaticOctave

Type : Number
Description : Chromatic octave.

Symbol.DisplayedHeight

Type : Number
Description : Displayed Y position, expressed in interlines from the lowest line of the staff.

Symbol.PlayedHzFrequency R

Type : Number
Description : Played note frequency, in Hz.
Depends on note pitch, transposing instrument key shift and comma ornaments.
Does not depend on bend ornaments, Rule (page 416)s or master tuning (A4 is assumed to be 440 Hz).

4.3.3.6.3. Managing time position

Symbol.Time R

Type : Number
Description : Written "time" position from the beginning of the staff. Time is expressed in the same unit as note duration (see constants in MSDefine (page 100).mys).
Use Score.CurrentPlayedTime (page 210) value to know the current played time.
Use Score.PlayedTimeToWrittenTime(...) (page 227) method to convert played time to written time.

Symbol.Bar R

Type : Number
Description : Written bar number the symbol belongs to.

Symbol.Beat R

Type : Number
Description : Written beat number from the beginning of the staff. This value is not an integer. For example, 2.5 means two beats and a half from the beginning.

4.3.3.6.4. Managing ornaments

Symbol.NumberOfOrnaments R

Type : Number
Description : Number of ornaments attached to this symbol. Ornament index can range from 1 to this value (included).

Symbol.Ornaments R

Type : Ornament array
Description : "Root" of ornament array for this symbol.
Cannot be used as is. Must be indexed.
Example :

```
--- Displays "true" if the second ornament of a symbol is a "Bend & Release"
ornament=mySymbol.Ornaments[2]
if ornament~=nil and ornament.Type==ORNAMENT_BEND_AND_RELEASE then print("true") end
```

Symbol.FirstOrnament

Type : Ornament
Description : First ornament attached to this symbol (equivalent to Symbol.Ornaments[1])
nil if there is no ornament currently attached to the symbol

Symbol.LastOrnament

Type : Ornament
Description : Last ornament attached to this symbol (equivalent to Symbol.Ornaments[Symbol.NumberOfOrnaments])
nil if there is no ornament currently attached to the symbol

4.3.3.6.5. Managing Grace notes

Symbol.NumberOfGraceNotes

Type : Number
Description : Number of grace notes attached to this symbol. Grace note index can range from 1 to this value (included).

Symbol.GraceNotes

Type : GraceNote array
Description : "Root" of grace note array for this symbol.
Cannot be used as is. Must be indexed.
Example :

```
--- Displays "true" if the second grace note of a symbol is an acciacatura (sounds after the note)
grace=mySymbol.GraceNotes[2]
if grace~=nil and grace.IsAfter==true then print("true") end
```

Symbol.FirstGraceNote

Type : GraceNote
Description : First grace note attached to this symbol (equivalent to Symbol.GraceNotes[1])
nil if there is no grace note currently attached to the symbol

Symbol.LastGraceNote

Type : GraceNote
Description : Last grace note attached to this symbol (equivalent to Symbol.GraceNote[Symbol.NumberOfGraceNotes])
nil if there is no grace note currently attached to the symbol

Symbol.GraceNoteStdPlayDuration

Type : Number
Description : Standard play duration of each grace note.

Symbol.GraceNoteStdDisplayDuration

Type : Number
Description : Standard display duration of each grace note.

Symbol.IsGraceNoteBeamHorizontal

Type : Boolean
Description : true if grace note beams are always horizontal.

Symbol.IsGraceNoteSlur

Type : Boolean
Description : true if grace note is slured to its main note.

Symbol.IsGraceNoteCrossAlone

Type : Boolean
Description : true if single grace notes are crossed.

Symbol.IsGraceNoteDisplayed

Type : Boolean
Description : true if the grace notes are displayed.

Symbol.IsGraceNoteStemsDown

Type : Boolean
Description : true if the grace notes have stems down.

Symbol.GraceNotesHorizontalShift

Type : Number
Description : Graces group horizontal shift.

4.3.3.6.6. Managing slurs

Symbol.SlurSource

Type : Symbol
Description : Symbol that supports the slur the current note is the target.
When a slur symbol ends to the current note, enables to know which symbol the slur starts from.
Setting this value to nil deletes the slur.
Note: The note provided can be on another staff. Use the Symbol.Staff (page 308) value to know the owner sta

Symbol.SlurTarget

Type : Symbol
Description : Symbol that ends the slur. When a slur symbol starts at the current note, enables to know which symbol the slur ends to.
Setting this value to nil deletes the slur.
Note: The note provided can be on another staff. Use the Symbol.Staff (page 308) value to know the owner sta

4.3.3.6.7. Managing special beams

Symbol.BeamSource

Type : Symbol
Description : Symbol that supports the beam the current note is the target.
When a special beam symbol ends to the current note, enables to know which symbol the beam starts from.
Setting this value to nil deletes the beam.
Note: The note provided can be on another staff. Use the Symbol.Staff (page 308) value to know the owner sta

Symbol.BeamTarget

Type : Symbol
Description : Symbol that ends the beam. When a beam symbol starts at the current note, enables to know which symbol the beam ends to.
Setting this value to nil deletes the beam.
Note: The note provided can be on another staff. Use the Symbol.Staff (page 308) value to know the owner sta

4.3.3.6.8. Tablature

Symbol.TabCell

Type : Number
Description : Tablature cell number for fretted instrument

Symbol.TabLine

Type : Number
Description : Tablature line number for fretted instrument

Symbol.ComputeTab

Type : Boolean
Description : If true the fingering is computed along the note pitch, if false, no computation.
If you switch to compute mode, you will have to apply manually the computation using Tablature.Compute(...) (page 372)
Setting some values set the ComputeTab to false : for example changing the symbol pitch using Symbol.Pitch (page 300), Symbol.DisplayedHeight (page 301), Symbol.PlayedPitch (page 300), Symbol.Name (page 300), Symbol.DiatonicDegree (page 300), Symbol.DiatonicOffset (page 300), Symbol.DiatonicOctave (page 301), Symbol.ChromaticDegree (page 301), Symbol.ChromaticOctave (page 301).
Setting some values set the ComputeTab to true : for example changing the symbol pitch using Symbol.TabCell (page 303), Symbol.TabLine (page 303).

Symbol.IsBlow

Type : Boolean
Description : For harmonica tablature : true if the note is blow. Changing this value don't change note pitch. Changing this value only have meaning if priority is set to tablature.
Example :

```
Include "MSDefine"
score=FrontScore()
if score ~=nil then
   staff=score.FirstStaff
   while staff ~=nil do
      if staff.Tablature~=nil and  staff.Tablature.Type==HARMONICA_TABLATURE then
         symbol=staff.FirstSymbol
         MyTextWindow=NewTextWindow("Export Harping !",true,0)
         while symbol ~=nil do
            if symbol.TabCell >=0  then
               if(symbol.IsDraw==false) then
                  MyTextWindow.Write("+",false)
               else
                  MyTextWindow.Write("-",false)
               end
               MyTextWindow.Write(symbol.HoleNumber,false)
               if(symbol.IsSlide==true) then
                  MyTextWindow.Write("s",false)
               end
               MyTextWindow.Write(" ",false)
            end
            symbol=symbol.Next
         end
         return
      end
   staff=staff.Next
   end
   Alert("No harmonica tablature in this score.")
else
      Alert("No score loaded")
end
```

Symbol.IsDraw

Type : Boolean
Description : For harmonica tablature : true if the note is draw. Changing this value don't change note pitch. Changing this value only have meaning if priority is set to tablature.

Symbol.HoleNumber

Type : Number
Description : For harmonica tablature : the hole number

Symbol.IsSlide

Type : Boolean
Description : For chromatic harmonica tablature : true if slide bar is pushed.

Symbol.IsAccordionPush

Type : Boolean
Description : For accordion tablature, true if the accordion is pushed
Example :

```
Include "MSDefine"
score=FrontScore()
if score ~=nil then
   staff=score.FirstStaff
   while staff ~=nil do
      if staff.Tablature~=nil and  staff.Tablature.Type==ACCORDION_TABLATURE then
         symbol=staff.FirstSymbol
         MyTextWindow=NewTextWindow("Display accordion symbols",true,0)
         while symbol ~=nil do
            print (symbol.TabLine.. " ".. symbol.TabCell)
            if symbol.AccordionLine >0  then
               if(symbol.IsAccordionPush==true) then
                  MyTextWindow.Write("P",false)
               else
                  MyTextWindow.Write("T",false)
               end
               MyTextWindow.Write(symbol.AccordionRow..",",false)
               MyTextWindow.Write(symbol.AccordionLine,false)
               MyTextWindow.Write(" ",false)
            end
            symbol=symbol.Next
         end
         return
      end
   staff=staff.Next
   end
   Alert("No accordion tablature in this score.")
else
      Alert("No score loaded")
end
```

Symbol.IsAccordionPull

Type : Boolean
Description : For accordion tablature, true if the accordion is pulled

Symbol.AccordionRow

Type : Number
Description : For accordion tablature, the row, right hand. (from 1 to 16)

Symbol.AccordionLine

Type : Number
Description : For accordion tablature, the line, right hand. (from 1 to 16)

4.3.3.6.9. Note miscellaneous parameters

Symbol.IsSelected

Type : Boolean
Description : true if the symbol is included in the selection range, false otherwise.
This value cannot be set because a selection range must be continuous. Set Staff.FirstSelectedSymbol (page 2 and Staff.LastSelectedSymbol (page 269) instead.
Note : Because of the amount of internal calculation it involves, it is **not recommended** to use this field as a test for browsing all selected symbols in a staff.
If you are sure the last symbol won't be deleted, it is preferable to test equality of the current symbol with Staff.LastSelectedSymbol (page 269), taken into a local variable.
It means instead of writing :

```
symbol=staff.FirstSelectedSymbol
while symbol and symbol.IsSelected do
    -- my processing here
    symbol=symbol.Next
end
```

it is highly recommended to write instead :

```
symbol=staff.FirstSelectedSymbol
stopSymbol=staff.LastSelectedSymbol
-- Gets symbol after the last selected one
-- so that the last selected symbol is processed
if stopSymbol then stopSymbol=stopSymbol.Next end
while symbol and symbol~=stopSymbol do
    -- my processing here
    symbol=symbol.Next
end
```

Both of these samples perform the same job, but the second way of writing makes execution be much, much faster.

Symbol.IsSelectedIndividually

Type : Boolean
Description : A note can be selected by area or individually. This value reflect the current state.
You can use Score.DeselectIndividually(...) (page 240) to unselect all these symbols.

Symbol.Velocity

Type : Number
Description : Note velocity (nil if the symbol is not a note)

Symbol.PlayedVelocity

Type : Number
Description : Played note velocity (nil if the symbol is not a note). Takes the dynamics into account.
Note : This value is an approximation, because a different dynamic value can be applied depending on the same note depending on the repeat

Symbol.IsRest

Type : Boolean
Description : true if the symbol is a rest, false otherwise

Symbol.IsGhostRest

Type : Boolean
Description : true if the symbol is a ghost rest, false otherwise

Symbol.IsRepeatBar

Type : Boolean
Description : true if the symbol is a "repeat bar" symbol, false otherwise. Be aware if you change a symbol to " repear bar" mode, the duration of the symbol must be equal to the bar duration.

Symbol.IsRepeatTwoLastBars

Type : Boolean
Description : true if the symbol is a "repeat two last bars" symbol, false otherwise. Be aware if you change a symbol to "repeat two last barr" mode, the duration of the symbol must be equal to the bar duration.

Symbol.IsNote

Type : Boolean
Description : true if the symbol is a note, false otherwise

Symbol.IsInChord

Type : Boolean
Description : true if the symbol belongs to a chord, false if it does not, or if it is the last symbol in the chord.
When several notes are included in a same chord, they are sorted by decreasing order of duration (so longest notes are the first in chord), and for equal duration, by decreasing pitches.
All notes are marked as being "in chord" except the last one (so the shortest, more low-pitched one), which gives its duration to the whole chord.
It means, in a chord made of a quarter and an eighth, you will find :
First, the quarter, marked "in chord"
Then, the eighth, not marked "in chord", so that the next note in the score will be separated by an eighth duration from this chord.
This value cannot be set. To insert a symbol in an existing chord, insert it at the chord's time position.

Symbol.IsBeamed

Type : Boolean
Description : true if the symbol is beamed with the next one.

Symbol.IsDeleted

Type : Boolean
Description : true if the symbol is marked for deletion. It is recommended NOT TO USE this setting.

Symbol.Tag

Type : String
Description : Tag related to this symbol. Tags are saved in the file.
See also : Score.SearchSymbolsByTag(...) (page 246) Score.ClearTagSymbols(...) (page 247) Staff.SearchSymbolsByTag(...) (page 296) Staff.ClearTagSymbols(...) (page 296)

Symbol.Duration

Type : Number
Description : Symbol duration (see constants in MSDefine (page 100).mys).
Special considerations
·This value can only be changed (set) for a rest or a note. Trying to set it for a ghost rest or a bar repeat symbol does nothing.
·Setting this value does not take the "limit editing to one bar" into account.
·Setting this value can shift the time position of all following notes in the staff.
·Setting this value for a note or a rest that belongs to a chord can change the relative order of notes in this chord.Due to this, you **must not** use Symbol.Previous (page 300) and Symbol.Next (page 299) when changing o of notes that belong to a chord. Use the ProcessChord function (see MSLibrary) instead.
·A rest can only be the shortest symbol in a chord. Setting this value for such a rest can set the rest duration to its maximal possible value (duration of the shortest note in this chord) instead of the duration you specified.
Example :

```
Include "MSLibrary"
-- Doubles note durations of the first staff
myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in the document")
   else
      symbol=myStaff.FirstSymbol
      while symbol do
         symbol=ProcessChord(symbol,function(s) s.Duration=s.Duration*2 end)
      end
   end
end
```

Symbol.PressureTime

Type : Number
Description : Note pressure time. Actual length of the note, in percent of its displayed duration.

Symbol.Delay

Type : Number
Description : Note delay (256th of quarter note). Can be negative if the note starts playing before its displayed position

Symbol.Staff

Type : Staff
Description : Staff that includes the symbol.

Symbol.Score

Type : Score
Description : Score that includes the symbol.

4.3.3.6.10. Accidental

Symbol.Accidental

Type : Number
Description : Preferred accidental for this note. That does not mean this accidental is actually displayed on score

Symbol.IsAccidentalParenthesis

Type : Boolean
Description : true if parenthesis are drawn around the accidental.

Symbol.IsCautionaryAccidental

Type : Boolean
Description : true if the accidental is always displayed, even if not necessary.

Symbol.IsHiddenAccidental

Type : Boolean
Description : true if the accidental is hidden.

Symbol.DisplayedAccidental

Type : Number
Description : Actually displayed accidental.

4.3.3.6.11. Managing display

Symbol.HorizontalPixelPosition

Type : Number
Description : Absolute pixel position of the symbol. This take account of the bar scale, engraver mode, horizontal pixel shifting. This is independant of the view scale of the score.

Symbol.VerticalPixelPosition

Type : Number
Description : Relative note position of the top of the note from the staff base line.

Symbol.HorizontalPixelSize

Type : Number
Description : Absolute pixel size of the note head width. This take account of the bar scale, engraver mode, horizontal pixel shifting. This is independant of the view scale of the score.

Symbol.VerticalPixelSize

Type : Number
Description : Absolute pixel size of the note head height.

Symbol.Shape

Type : Number
Description : Note head shape (see constants in MSDefine (page 100).mys).
SHAPE_NORMAL=0 SHAPE_TRIANGLE=1 SHAPE_DIAMOND=2 SHAPE_CROSS=3
SHAPE_PLUS=4 SHAPE_SQUARE=5 SHAPE_BEAM=6 SHAPE_NONE=7
SHAPE_FLAG=8 SHAPE_HALF_CIRCLE=9 SHAPE_DROP=10 SHAPE_SAND_GLASS=11
SHAPE_ARC=12 SHAPE_EMPTY_DIAMOND=13 SHAPE_CONE=14 SHAPE_CRESCENT=15
SHAPE_LEFT_TRIANGLE=16 SHAPE_BELL=17 SHAPE_CONCAVE_TRIANGLE=18
SHAPE_CONVEX_TRIANGLE=19
SHAPE_JOHNSON_TRIANGLE=20 SHAPE_JOHNSON_SQUARE=21 SHAPE_CIRCLE_BOLD_CROSS=22
SHAPE_BOLD_CROSS=23
SHAPE_THIN_CROSS=24 SHAPE_HALF_CIRCLE_THIN_CROSS=25 SHAPE_EMPTY=26

Symbol.Color

Type : Number
Description : Note color index. Use Symbol.RGBColor (below) to use the Red, Green, Blue components instead of a color index

Symbol.RGBColor

Type : String
Description : Note RGB components. When read, functionnally equivalent to Graph.IndexToRGB(symbol.Color)

Symbol.SplitDuration

Type : Number
Description : Split duration (see constants in MSDefine (page 100).mys).

Symbol.StemHeight

Type : Number
Description : Stem height

Symbol.RestVerticalOffset

Type : Number
Description : Vertical offset (in pixel) of the rest. If the symbol is not a rest, return 0

Symbol.StemType

Type : Number
Description : Stem direction, either:
0 = automatic (depends on note pitch)
1 = forced down
2 = forced up

Symbol.AccidentalShift

Type : Number
Description : Accidental X offset from its regular position in pixel 72dpi.

Symbol.HeadShift

Type : Number
Description : Symbol X offset from its regular position, in percent. This value varie from -127 to 127.
Negative values move to the left, positive values to the right, 0 value mean centered.
Using values greater than 100 or lower than -100 can swap the notes positions: the note with a lower time position will be displayed after the next note.
This is NOT recommanded.

Symbol.TupletMult

Type : Number
Description : Tuplet first value (3 for a triplet).
Setting this value does not change the symbol duration. It just asks the software to display this duration as a tuplet.

Symbol.TupletDiv

Type : Number
Description : Tuplet second value (2 for a 3:2 triplet)

Symbol.IsTupletFull

Type : Boolean
Description : true if the tuplet is displayed in full (ex : 3:2) or abbreviated (ex : 3)

Symbol.IsTupletBracket

Type : Boolean
Description : true if an horizontal bracket or slur is drawn when several tuplets are beamed together

Symbol.IsTupletSlur

Type : Boolean
Description : true if slur will be displayed instead of a bracket

Symbol.IsTupletDontDisplayNumber

Type : Boolean
Description : true if no number will be displayed.

Symbol.IsTupletDontDisplayBeam

Type : Boolean
Description : true if the beam will not be displayed but the real note duration.

Symbol.IsTupletInvertedSymbol

Type : Boolean
Description : true if the symbol related to tuplet (bracket, brace) will be displayed near note head instead of stem.

Symbol.TupletSymbolHorizontalOffset

Type : Number
Description : Horizontal offset of the symbol related to tuplet (bracket, brace)

Symbol.TupletSymbolVerticalOffset

Type : Number
Description : Vertical offset of the symbol related to tuplet (bracket, brace)

Symbol.TupletSymbolVerticalSize

Type : Number
Description : Vertical size of the bracket related to tuplet (when tuplet are grouped through a bracket)

Symbol.TupletSymbolCurveTension

Type : Number
Description : Tension of the curve over tuplet in percent (when tuplet are grouped through a curve)

Symbol.TupletValueVerticalOffset

Type : Number
Description : Vertical offset of the displayed tuplet value

Symbol.Scale

Type : Number
Description : Symbol scale in percent

Symbol.IsStemDown

Type : Boolean
Description : true if the symbol has its stem down, false if it is upward

Symbol.IsParenthesis

Type : Boolean
Description : true if the symbol has parenthesis around the head

Symbol.IsInvertedHead

Type : Boolean
Description : true if the head of the symbol (that must be a note) is inverted

Symbol.IsInvertedHeadDisplayed

Type : Boolean
Description : true if the symbol (that must be a note) is displayed with an inverted head. In some case, like note to close in a chord,
the note head is automatically inverted. Please note, the score must be displayed at least one time because the computation of this value is done
at display time.

Symbol.IsInvertedTie

Type : Boolean
Description : true if the symbol is displayed with an inverted tie

Symbol.IsInvertedDot

Type : Boolean
Description : if true the dot will be displayed in alternate position

Symbol.IsInvisibleDot

Type : Boolean
Description : if true the dot will not be displayed

Symbol.OffsetXDot

Type : Number
Description : Horizontal offset of the dot relative to default position

Symbol.OffsetYDot

Type : Number
Description : Vertical offset of the dot relative to default position

Symbol.IsInvisibleRest

Type : Boolean
Description : true if the symbol (that must be a rest) is invisible

Symbol.IsRealDurationRest

Type : Boolean
Description : The rest will be dsplayed with its real duration independantly of the pause display mode defined into the staff.

Symbol.Id

Type : Number
Description : Internal MyrScript unique ID for this symbol

4.3.3.7. Symbol methods

4.3.3.7.1. Appearance

Symbol.ComputeStemDirection(...)

Description : From a symbol, compute the stem direction along its pitch. The stem type is set to STEM_AUTO
Parameters : None

Symbol.Draw(...)

Description : Draw the symbol
Parameters : None

4.3.3.7.2. Deleting symbols

Symbol.Delete(...)

Description : Deletes a symbol. The symbol is removed from its staff, and the following symbols are shifted left to fill in the free space.
Depending on the value of the "Limit editing to one bar" setting, this shifting propagates either to the end of the current bar, or to the end of the tune.
Parameters : Optional Boolean : true if delete operation is limited to one bar, false otherwise
Special Considerations :
- When parameter is true, the delete operation behaves like a manual symbol deletion when "Options>Limit editing to one bar" setting is on.
- When parameter is false, the delete operation behaves like a manual symbol deletion when "Options>Limit editing to one bar" setting is off.
- When parameter is omitted, the delete operation behaves like a manual symbol deletion, using the current setting in "Options>Limit editing to one bar"

Return value : true if the symbol has been successfully deleted
false if it could not be deleted (for example when attempting to delete a useful ghost rest)
nil if the symbol is invalid
Example :

```
-- Deleting the first symbol on a staff
local myScore
local myStaff
local mySymbol
local myResult
myScore=FrontScore()
if myScore==nil then
   Message("Please load a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      Message("There is no staff in this score")
   else
      mySymbol=myStaff.FirstSymbol
      if mySymbol ~=nil then
         myResult=mySymbol.Delete()
         if myResult==nil then Message("Invalid symbol")
            else if myResult==false then Message("Impossible to delete first symbol")
            else
               Application.UpdateScores()
               Message("First symbol successfully deleted")
            end
         end
      else Message("This staff is empty")
      end
   end
end
```

4.3.3.7.3. Echo

Symbol.StartEcho(...)

Description : Start the echo on the specifed note (Only if the user had enabled this feature)
Parameters : None
Return value : None

4.3.3.7.4. Managing Grace notes

Symbol.AddGraceNote(...)

Description : Adds a new grace note.
Parameters : (All parameters are optional)
GraceNote : Existing grace note BEFORE which the new one must be inserted.
nil or omitted if the grace note has to be inserted in last position
Boolean: true if the grace note has to be displayed after the note (acciacatura)
Number : Pitch offset from its refering note (semitone)
Number : Preferred accidental for displaying this grace note (see constants in MSDefine (page 100).mys)
Number : Dividing value from the regular played and displayed length of grace note (see Symbol.GraceNoteStdPlayDuration (page 302) and Symbol.GraceNoteStdDisplayDuration (page 302) values)
  0 means regular duration
  1 means halved duration
  2 means duration divided by 4
  3 means duration divided by 8
  etc
Number : Velocity from 0 to 127, or -1 if velocity is the note velocity (by default)
Boolean: true if the grace note is in chord with the next one (default: false)
Return value : GraceNote: newly inserted grace note, or nil if impossible to insert
Example :

```
myGraceNote=myNote.AddGraceNote(nil,false,-2)
```

Note : Up to 10 grace notes can be attached to the same note. Attempting to insert a 11th will fail, and return nil

Symbol.DeleteAllGraceNotes(...)

Description : Deletes all grace notes of a symbol
Parameters : None
Return value : None
Example :

```
myNote.DeleteAllGraceNotes()
```

4.3.3.7.5. Managing ornaments

Symbol.AddOrnament(...)

Description : Adds a new ornament.
Parameters : (All parameters are optional)
Number : Ornament type (see constants in MSDefine (page 100).mys)
Number : X offset in 72 dpi pixel from its default position
Number : Y offset in 72 dpi pixel from its default position
Number : Display origin (see constants in MSDefine (page 100).mys)
Boolean: true if this ornament will be played, false if it has no effect on sound output
Number : Color index for this ornament. Use Graph.RGBToIndex(...) (page 521) function to get an index from Re
Green, Blue color settings.
Boolean: true if this ornament is invisible, i.e. not displayed on score.

Next parameters are only useful for ORNAMENT_TEXT ornament type
String : Text value for this ornament
String : Font name for displaying this text
Number : Font size (72 dpi pixel) for displaying this text
String : Font face for displaying this text (see TextStyle object)
Return value : Ornament : ornament object that has just been inserted (or nil if error)
Example :

```
    myOrnament=myNote.AddOrnament(ORNAMENT_BEND)
```

Symbol.DeleteAllOrnaments(...)

Description : Deletes all the ornaments of a symbol.
Parameters : None
Return value : None
Example :

```
    myNote.DeleteAllOrnaments()
```

4.3.3.7.6. Managing Slurs

Slurs are related to two symbols (both must be notes) : the slur starting note and slur ending note.
A same note can be both the ending note of a slur, and the starting note of another one.
The starting note is considered as the slur owner. This means you must refer to this starting note to get or set slur settings.
Please note: From Harmony 8.5 (MyrScript 0.2.3) the note beginning the slur and the note ending the slur can be on **different staves**.

Symbol.GetSlurSettings(...)

Description : Gets the parameters of an existing slur
Parameters : None
Return value : Number : Kind of sliding (see constants in MSDefine (page 100).mys)
Number : Delay before slide
Number : X offset (72 dpi pixel) of the beginning of the slur symbol
Number : Y offset (72 dpi pixel) of the beginning of the slur symbol
Number : X ratio (1000th) of the middle point of the slur symbol
Number : Y ratio (1000th) of the middle point of the slur symbol
Number : X offset (72 dpi pixel) of the end of the slur symbol
Number : Y offset (72 dpi pixel) of the end of the slur symbol
Number : Slur symbol width (288 dpi pixel) O means invisible
Number : Slur color index. Use Graph.IndexToRGB(...) (page 521) function to get the Red, Green, Blue color settings from a color index
Number : Slur type SLUR_APPEARANCE_STANDARD = 0, SLUR_APPEARANCE_BRACKET=1, ...
Boolean : Automatic slur computing
Number : Slur tension
or nil if there is no slur attached to this note
Example :

```
myKindOfSlurSlide=myNote.GetSlurSettings()
```

Symbol.SetSlurSettings(...)

Description : Sets the parameters of an existing slur
Parameters : Number : Kind of sliding (see constants in MSDefine (page 100).mys)
Other parameters are optional
Number : Delay before slide
Number : X offset (72 dpi pixel) of the beginning of the slur symbol
Number : Y offset (72 dpi pixel) of the beginning of the slur symbol
Number : X ratio in 1000th of the middle point of the slur symbol. This value varies from 0 to 1000. 0 is the X value of the source note 1000 the X value of the target note.
Number : Y ratio (1000th) of the middle point of the slur symbol. This value can be positive or negative. This is the distance, in 1000th of staff line spacing, to the average of source and target note Y position.
Number : X offset (72 dpi pixel) of the end of the slur symbol
Number : Y offset (72 dpi pixel) of the end of the slur symbol
Number : Slur symbol width (288 dpi pixel)
Number : Slur color index. Use Graph.RGBToIndex(...) (page 521) function to get this value from Red, Green and Blue components
Number : Slur type SLUR_APPEARANCE_STANDARD = 0, SLUR_APPEARANCE_BRACKET=1, ...
Boolean : Automatic slur computing
Number : Slur tension
Return value : None
Example :

```
-- Sets a slur color index to 0
kind,delay,x1,y1,xm,ym,x2,y2,width=myNote.GetSlurSettings()
if kind~=nul then myNote.SetSlurSettings(kind,delay,x1,y1,xm,ym,x2,y2,width,0) end
```

Symbol.ComputeSlursApparence(...)

Description : Adjust the slurs apparence to avoid overlapping with this note. This will only affect slurs which allow automatic slur computing.
See Symbol.GetSlurSettings(...) (above) and Symbol.SetSlurSettings(...) (above)
Parameters : None
Return value : None

4.3.3.7.7. Managing special beams

Special beams are related to two symbols (both must be notes) : the beam starting note and beam ending note.
A same note can be both the ending note of a beaming, and the starting note of another one.
The starting note is considered as the beam owner. This means you must refer to this starting note to get or set beam settings.

Symbol.GetBeamSettings(...)

Description : Gets parameters for an existing beam
Parameters : None
Return value : Number : Position of the split in percent (0-100%)
Number : Number of beams in the first part of the beaming
Number : Number of beams in the second part of the beaming
Number : Beam width in pixel
Number : Distance between beams in pixel
Boolean : beams are inverted
Boolean : note is the last of a tuplet group
Number : Distance between beam edges and note stem (in pixel)
Boolean : True if the special beam is played as a tremolo

Symbol.SetBeamSettings(...)

Description : Sets the parameters for an existing beam
Parameters : Number : Position of the split point in percent (0-100%)
Number : Number of beams in the first part of the beaming
Number : Number of beams in the second part of the beaming
Number : beam width in pixel
Number : Distance between beams in pixel
Boolean : Beams are inverted
Boolean : note is the last of a tuplet group
Number : Distance between beam edges and note stem (in pixel)
Boolean : true if the special beam is played as a tremolo
Return value : None

4.3.3.7.8. Managing symbol settings

Symbol.GetAppearance(...)

Description : Gets the appearance from another symbol.
Pitch, duration, slur, special beam and tuplet information are NOT copied. All other settings are.
Parameters : Symbol to copy appearance from
Return value : None
Example :

```
myNote.GetAppearance(myOtherNote)
```

4.3.3.7.9. Managing tied notes

Symbol.GetTiedElements(...)

Description : From a symbol, gets the list of tied elements that are shown on score for this symbol
Parameters : None
Return value : A lua table, containing as many elements as the number of tied notes needed to display the symbol
Each element of the table contains two fields :
.time : time position of the element from the beginning of the score
.duration : duration of the element in time unit (sum of the durations equals duration of the original symbol)
Example :

```
-- Displays tied note information about a symbol
eltTable=mySymbol.GetTiedElements()
for index=1,getn(eltTable) do
    print("Element #"..index.." at position "..eltTable[index].time..", duration : "..eltTable[index].duration)
end
```

4.3.3.7.10. Rules

Symbol.GetRelatedRule(...)

Description : For the note provide a rule which criterion match the note parameters and related to a drum instrument.
Parameters : Optional: Number, index of the rule to extract (a note can match more than one rule) 1 by default
Return value : Rule : the rule (or nil if none)

Symbol.SetRelatedRule(...)

Description : Changes the note parameter as the note will match the rule.
Parameters : Rule
Return value : None

4.3.4. GlobalBarSetting Object Type

General description of this object type : Time signature, bar scale, bar margins, and other miscellaneous global settings for a given bar in a Score (page 198).
This object can be accessed through the Score.GetGlobalBarSetting(...) (page 234) method.

4.3.4.1. GlobalBarSetting values

4.3.4.1.1. Time Signature

GlobalBarSetting.ActualTimeSignatureNumerator

Type : Number
Description : Actual numerator for time signature
Example :

```
-- Changes time signature at bar 5
score=FrontScore()
if score then
   -- Global bar setting at bar 5
   gbs=score.GetGlobalBarSetting(5)
   -- Next time sig changes
   lastbar=score.GetBarTimeSignatureChange(5)
   -- set time sig to 2/3
   gbs.ActualTimeSignatureNumerator=8
   gbs.ActualTimeSignatureDenominator=4
   -- Force display
   gbs.IsDisplayedTimeSignature=true
   -- Propagate change till the next time sig change (or end of tune)
   score.PropagateGlobalBarSetting(5,lastbar-1)
else print("need a score for this example") end
```

Note : Changing this value resets the displayed time signature, visibility and beaming to their default values

GlobalBarSetting.ActualTimeSignatureDenominator

Type : Number
Description : Actual numerator for time signature
Note : Changing this value resets the displayed time signature, visibility and beaming to their default values

GlobalBarSetting.IsDisplayedTimeSignature

Type : Boolean
Description : true if the time signature is displayed on all staves. Can be overriden by the BarSetting (page 332) object of each staff.
Changing this value will produce a different page setup. The new page setup will be computed next time the score will be displayed.
You can force the computation using Score.ComputePageSetup(...) (page 239).

GlobalBarSetting.DisplayedTimeSignatureNumerator

Type : Number
Description : Displayed numerator for time signature

GlobalBarSetting.DisplayedTimeSignatureNumerator2

Type : Number
Description : Displayed numerator second part for time signature
For example, the actual time signature can be 8/4, but displayed as 5+3/4.
In this case, 5 is the DisplayedTimeSignatureNumerator field, 3 is the DisplayedTimeSignatureNumerator2 field, and 4 is the DisplayedTimeSignatureDenominator field.

GlobalBarSetting.DisplayedTimeSignatureDenominator

Type : Number
Description : Displayed denominator for time signature

GlobalBarSetting.IsAbbreviatedTimeSignature

Type : Boolean
Description : true if 2/4 and 4/4 time signature can be displayed using their abbreviated notation

GlobalBarSetting.NumberOfBeat

Type : Number
Description : Number of beat that matches this time signature

4.3.4.1.2. Beaming defaults

Defines the default behaviour of automatic beaming for this bar.

GlobalBarSetting.IsBeamedTuplets

Type : Boolean
Description : true if tuplets are always beamed according to the tuplet numerator, even if they are over a beat change.

GlobalBarSetting.IsBeamEighthsByFour

Type : Boolean
Description : true if eighth notes are beamed by 4 instead of 2 in 4/4 time signature.

GlobalBarSetting.BeamMask

Type : String
Description : Beaming mask for beats.
This string is made of as many characters as there are reference notes (actual time signature numerator) minus one in the time signature.
A "-" character means the note will be beamed with the next one, a 'l' character means it won't be beamed with the next one.

4.3.4.1.3. Bar scale and display size

GlobalBarSetting.Scale

Type : Number
Description : Bar scale, in 1000th, to be applied to this bar.

GlobalBarSetting.LeftMargin

Type : Number
Description : Left margin size, in 72 dpi pixels.

GlobalBarSetting.RightMargin

Type : Number
Description : Right margin size, in 72 dpi pixels.

4.3.4.1.4. Bar numbering settings

GlobalBarSetting.IsIgnoredForNumbering

Type : Boolean
Description : true if this bar is not taken into account in bar numbering

GlobalBarSetting.IsForcedDisplay

Type : Boolean
Description : true if this bar number display is forced

GlobalBarSetting.IsDisabledDisplay

Type : Boolean
Description : true if this bar number display is disabled

GlobalBarSetting.ForcedNumbering

Type : Number
Description : Forced number for this bar. If a non-nul number is set, numbering restarts from this value when the bar is reached.

4.3.4.1.5. Miscellaneous display settings

GlobalBarSetting.IsEndOfLine

Type : Boolean
Description : true if the user had forced an end of line after this bar. This bar will be the last to be printed on this line.

GlobalBarSetting.DontBreakLine

Type : Boolean
Description : true if a end of line will never occurs after this bar.

GlobalBarSetting.IsEndOfPage

Type : Boolean
Description : true if the user had forced an end of page after this bar. This bar will be the last to be printed on this page.

GlobalBarSetting.IsFirstOfLine

Type : Boolean
Description : True if the bar is the first of the line in the printing process.

GlobalBarSetting.IsSplitMultiRest

Type : BooleanBoolean
Description : true if the multi-bar rest will be split after reaching the end of this bartrue if the multi-bar rest will be split after reaching the end of this bar

GlobalBarSetting.IsSplitNotes

Type : Boolean
Description : true if notes are split by the barline at the end of this bar

GlobalBarSetting.IsInvisible

Type : Number
Description : 1 if the bar is invisible because included in a multirest bar, 0 if the bar is visible, -1 if this value can't be provided : we are in scroll mode.

4.3.4.2. GlobalBarSetting methods

4.3.4.2.1. Managing

GlobalBarSetting.CopyTo(...)

Description : Copy all the GlobalBarSetting setting in another GlobalBarSetting.
Parameters : The target GlobalBarSetting
Return value : None
Example :

```
-- Copies globalbarsetting from bar 1 to bar 2
score=FrontScore()
if score then
    -- Global bar setting at bar 1
    gbs1=score.GetGlobalBarSetting(1)
    -- GlobalBarSetting at bar 2
    gbs2=score.GetGlobalBarSetting(2)
    -- Copy 1 to 2
    gbs1.CopyTo(gbs2)
else print("need a score for this example") end
```

4.3.5. BarAction Object Type

General description of this object type : Barlines and break symbols that can be included in a score.
There are several types of such objects :

Barlines

Decorative barlines
These barlines have no effect on how the score is played.
There are 4 types of such barlines:
Invisible barline
Double barline
Bold barline
Dotted barline
The standard barline is displayed when no other barline is present.

Piece start/end barline
They indicate the first and last bar to be played.

Repeat start/end barline
They mark start and end of a set of bars to be repeated several times.

Break symbols

Part marker start/end
They indicate a set of bars to be played only at some play counter.

Segno/Coda define
They define a numbered Segno or Coda mark

Jumps and stops
They define an action to be taken when the mark is reached.
It includes symbols like DC, DS, DC Al Coda, Fine, etc...
These marks are fully configurable, and can be set up to perform various actions.

Depending on the object kind, some values will or won't be available.

4.3.5.1. BarAction values

4.3.5.1.1. Browsing BarActions

BarAction.Next

Type : BarAction
Description : Next BarAction
If there is no next BarAction, value is nil.

BarAction.Previous

Type : BarAction
Description : Previous BarAction
If there is no previous BarAction, value is nil.

4.3.5.1.2. BarActions miscellaneous values

These values can apply to any kind of BarAction object.

BarAction.Type

Type : Number
Description : BarAction type. Can be any of the constants defined in MSDefine.mys, i.e. :
- INVISIBLE_BARLINE
- DOUBLE_BARLINE
- BOLD_BARLINE
- DOTTED_BARLINE
- PIECE_START_BARLINE
- PIECE_END_BARLINE
- REPEAT_START_BARLINE
- REPEAT_END_BARLINE
- PART_START_BREAK_SYMBOL
- PART_END_BREAK_SYMBOL
- CODA_BREAK_SYMBOL
- SEGNO_BREAK_SYMBOL
- FINE_BREAK_SYMBOL
- AL_CODA_BREAK_SYMBOL
- DA_CAPO_BREAK_SYMBOL
- DA_SEGNO_BREAK_SYMBOL
- DA_CAPO_AL_FINE_BREAK_SYMBOL
- DA_SEGNO_AL_FINE_BREAK_SYMBOL
- DA_CAPO_AL_CODA_BREAK_SYMBOL
- DA_SEGNO_AL_CODA_BREAK_SYMBOL
- SIMPLE_BARLINE

Note : 1- Be careful: setting this value will reset all other values of the BarAction (except Bar number) to default
2- Because some actions are not part of the regular music notation, if you have altered any of the 8 last object types in the list above to make it perform another action (like jump to a given bar, or jump to the first played bar) reading this field will then result in : CUSTOM_BREAK_SYMBOL

BarAction.Bar

Type : Number
Description : Bar at which the BarAction object is located
Example :

```
Include "MSDefine"
-- Displays bar number for each BarAction
myScore=FrontScore()
if myScore==nil then
   print("Please load a score first")
else
   index=1
   myBarAction=myScore.FirstBarAction
   while myBarAction~=nil do
      print("BarAction #",index,"is located at bar",myBarAction.Bar);
      index=index+1
      myBarAction=myBarAction.Next
   end
end
```

BarAction.Text

Type : String
Description : Text to be displayed for this BarAction
Note : This text can contain tagged data :
$P : number of repetitions/ play count number
$[: opening half-bracket (used in part start)
$# : list of play counts this BarAction triggers
$] : closing half-bracker (used in part end)
$S : Segno symbol
$C : Coda symbol
$N : segno, coda or bar number
$T : suffix for play count number (in English "st time", "nd time", "rd time"...) depending on the play count value.
$$: $ symbol
For example, if you define a break symbol to jump to bar #5 the 3rd time, enter a text like :
PT, go to bar#$N
which will be displayed on score under the form :
3rd time, go to bar#5
Be careful, changing the text won't change the way a BarAction is actually processed when playing the piece. You are responsible for changing the BarAction settings, then changing the text so that it reflects what the BarAction actually performs.

BarAction.TextStyle

Type : TextStyle
Description : Text style for BarAction text.

BarAction.TextXOffset

Type : Number
Description : Text X offset (72 dpi pixel)

BarAction.TextYOffset

Type : Number
Description : Text Y offset (72 dpi pixel)

4.3.5.1.3. Barlines

These values only apply to barlines.

BarAction.BarlinePosition

Type : Number
Description : Barline position
Can be one of the following constants defined in MSDefine.mys :

BARLINE_POSITION_BAR_START : At the very beginning of the bar, before clef
BARLINE_POSITION_AFTER_KEY : Beginning of the bar, after clef, key or time signature
BARLINE_POSITION_AFTER_MARGIN : Beginning of the bar, after clef, key or time signature and user-defineable margin

BARLINE_POSITION_BAR_END : At the end of the bar
Note :
The three first constants can be used to set position of any barline that can be at the beginning of a bar, i.e. :
·INVISIBLE_BARLINE
·DOUBLE_BARLINE
·BOLD_BARLINE
·DOTTED_BARLINE
·PIECE_START_BARLINE
·REPEAT_START_BARLINE
The last one (BARLINE_POSITION_BAR_END) can only be used for decorative barlines :
·INVISIBLE_BARLINE
·DOUBLE_BARLINE
·BOLD_BARLINE
·DOTTED_BARLINE

Setting this value to a wrong type of BarAction do not have any effect.
Getting this value from another type of BarAction will return its default position (cannot be changed).

BarAction.BarlineColorIndex

Type : Number
Description : Barline color

4.3.5.1.4. Break symbol action

These values enable to define what action is taken when the break symbol is reached

BarAction.Action

Type : Number
Description : Action to be taken when the break symbol is reached.
It is used to define a custom break symbols, which performs an action other than pre-defined.
This value can be modofied for the following list of BarAction types
- CUSTOM_BREAK_SYMBOL
- FINE_BREAK_SYMBOL
- AL_CODA_BREAK_SYMBOL
- DA_CAPO_BREAK_SYMBOL
- DA_SEGNO_BREAK_SYMBOL
- DA_CAPO_AL_FINE_BREAK_SYMBOL
- DA_SEGNO_AL_FINE_BREAK_SYMBOL
- DA_CAPO_AL_CODA_BREAK_SYMBOL
- DA_SEGNO_AL_CODA_BREAK_SYMBOL

Action field can be set for these kinds of BarAction to one of these values :
- BREAK_SYMBOL_NO_ACTION : No action is performed
- BREAK_SYMBOL_STOP : Conditional stop (Fine)
- BREAK_SYMBOL_JUMP_TO_BAR : Jump to a given bar
- BREAK_SYMBOL_JUMP_TO_WRITTEN_START : Jump to written start (bar #1)
- BREAK_SYMBOL_JUMP_TO_PLAYED_START : Jump to played start (bar #1 or first bar with a PIECE_START_BARLINE symbol)
- BREAK_SYMBOL_JUMP_TO_SEGNO : Jump to a segno mark
- BREAK_SYMBOL_JUMP_TO_CODA : Jump to a coda mark
- BREAK_SYMBOL_DEFINE_SEGNO : Segno definition
- BREAK_SYMBOL_DEFINE_CODA : Coda definition
- BREAK_SYMBOL_DEFINE_START : Music start definition
- BREAK_SYMBOL_DEFINE_END : Music end definition
- BREAK_SYMBOL_DEFINE_PART_START : Part start
- BREAK_SYMBOL_DEFINE_PART_END : Part end
- BREAK_SYMBOL_DEFINE_REPEAT_START : Repeat start
- BREAK_SYMBOL_DEFINE_REPEAT_END : Repeat end

Once this value has been set, BarAction.Type (page 323) may change to another type of pre-defined break symb
or to CUSTOM_BREAK_SYMBOL if it matches none.
Note : The text won't be changed if you set the Action field. To change both action and text to a pre-defined value, change BarAction.Type (page 323)

BarAction.ActionParameter

Type : Number
Description : Parameter for the action. The use of this parameter depends on the action itself :
- BREAK_SYMBOL_NO_ACTION : No parameter
- BREAK_SYMBOL_STOP : No parameter
- BREAK_SYMBOL_JUMP_TO_BAR : Bar number to jump to
- BREAK_SYMBOL_JUMP_TO_WRITTEN_START : No parameter
- BREAK_SYMBOL_JUMP_TO_PLAYED_START : No parameter
- BREAK_SYMBOL_JUMP_TO_SEGNO : Segno number to jump to
- BREAK_SYMBOL_JUMP_TO_CODA : Coda number to jump to
- BREAK_SYMBOL_DEFINE_SEGNO : Segno number to define
- BREAK_SYMBOL_DEFINE_CODA : Coda number to define
- BREAK_SYMBOL_DEFINE_START : No parameter
- BREAK_SYMBOL_DEFINE_END : No parameter
- BREAK_SYMBOL_DEFINE_PART_START : No parameter
- BREAK_SYMBOL_DEFINE_PART_END : No parameter
- BREAK_SYMBOL_DEFINE_REPEAT_START : No parameter
- BREAK_SYMBOL_DEFINE_REPEAT_END : No parameter

BarAction.ActionActivation

Type : Number
Description : When the action is performed, it can also activate "Fine" or "Coda" flag.
When this flag is activated, last time a "Fine" or "Coda" jump will be encountered, it will be performed in any case.
This value can be :
- ACTIVATE_NOTHING
- ACTIVATE_CODA
- ACTIVATE_FINE

4.3.5.1.5. Break symbol action triggering

These values enable to define when the action will be performed

BarAction.TriggerType

Type : Number
Description : Defines when the action will be performed.
Can be one of the following constants (defined in MSDefine.mys)
- TRIGGER_ALWAYS : always trigger
- TRIGGER_X_TIMES : several times, then stops triggering
- TRIGGER_EACH_X_TIMES : triggers periodically
- TRIGGER_SELECTED_TIMES : triggers according to a list
- TRIGGER_LAST_TIME : triggers only the last time this bar is played

Only applies to :
- REPEAT_END_BARLINE : Always TRIGGER_X_TIMES. Cannot be changed
- PART_START_BREAK_SYMBOL : Always TRIGGER_SELECTED_TIMES. Cannot be changed
- FINE_BREAK_SYMBOL : Default TRIGGER_LAST_TIME. Can be changed
- AL_CODA_BREAK_SYMBOL : Default TRIGGER_LAST_TIME. Can be changed
- DA_CAPO_BREAK_SYMBOL : Default TRIGGER_ALWAYS. Can be changed
- DA_SEGNO_BREAK_SYMBOL : Default TRIGGER_ALWAYS. Can be changed
- DA_CAPO_AL_FINE_BREAK_SYMBOL : Default TRIGGER_ALWAYS. Can be changed
- DA_SEGNO_AL_FINE_BREAK_SYMBOL : Default TRIGGER_ALWAYS. Can be changed
- DA_CAPO_AL_CODA_BREAK_SYMBOL : Default TRIGGER_ALWAYS. Can be changed
- DA_SEGNO_AL_CODA_BREAK_SYMBOL : Default TRIGGER_ALWAYS. Can be changed

BarAction.TriggerParameter

Type : Number/String
Description : Parameter for triggering. Depends on the TriggerType above :
- TRIGGER_ALWAYS : No parameter
- TRIGGER_X_TIMES : Number of times trigger occurs before being disabled
- TRIGGER_EACH_X_TIMES : Number of times the bar must be played to make triggering occur once.
- TRIGGER_SELECTED_TIMES : String of up to 16 characters that defines selected times. A '-' character means : 'do not trigger', an 'O' means 'trigger'. If characters are missing, do not trigger.
- TRIGGER_LAST_TIME : No parameter

4.3.5.2. BarAction methods

4.3.5.2.1. Deleting a BarAction

BarAction.Delete(...)

Description : Deletes the BarAction object
Parameters : None
Return value : None
Example :

```
-- Deletes all BarActions in score
myScore=FrontScore()
if myScore==nil then
    print("Please load a score first")
else
    --We preserve the current score state,
    --so the user will be able to undo the changes.
    myScore.Preserve()
    myBarAction=myScore.LastBarAction
    while myBarAction~=nil do
        -- Once object has been deleted, Previous member could not be retrieved
        previousBarAction=myBarAction.Previous
        myBarAction.Delete()
        myBarAction=previousBarAction
    end
end
```

4.3.5.2.2. Managing

BarAction.CopyTo(...)

Description : Copy all the BarAction setting in another BarAction.
Parameters : The target BarAction
Return value : None

4.3.6. StavesGroup Object Type

General description of this object type : Staves group object allow you to manage group of staves on the score. The group can be show with a brace, a bracket, a line or can be invisible.
The following constants are defined in MSDefine :
SYMBOL_BRACE a brace
SYMBOL_BRACKET a bracket
SYMBOL_SUB_BRACKET a sub-bracket
SYMBOL_BRACKET_UP a bracket but only with the up part
SYMBOL_BRACKET_DOWN a bracket but only with the down part
SYMBOL_LINE a line (in fact a bracket without up and down
SYMBOL_NONE no symbol displayed

The invisible symbol allow you to define join mode without showing a symbol

The staves group define the way bars lines are graphically joined.
The following constants are defined in MSDefine
JOIN_LINE_NONE : no line between staves
JOIN_LINE_STANDARD : a line join the staves inside the group
JOIN_LINE_MENSURSTRICH : a line join the staves, but outside the staff lines. No bar line is displayed on the staff except for a repeat symbol.

A staves group need two staves to be correctly defined: the staff where the symbol starts, the staff where the symbol ends.
These two staves must be different, a staff group can't starts and ends from the same staff. In this case, the object will be present into the score but never displayed.

A name (complete and abreviated) can be related to the staves group. By default the name is displayed centered vertically on the staves group and right aligned to the staves group area.

A method is related to the object Score to be able to create a new staves group object: Score.AddStavesGroup(...) (page

A value is related to the object Score to be able to browser staves group Score.FirstStavesGroup (page 219)
You can also browse staves group inside a score using Score.NumberOfStavesGroups (page 220) and Score.StavesGro (page 219) as an array.

You can know the space left at the left of the staff to display the group symbols using Staff.GroupsMargin (page 271) valu

4.3.6.1. StavesGroup values

4.3.6.1.1. Browsing Staves groups

StavesGroup.Next

Type : StavesGroup
Description : Next StavesGroup
  If there is no next StavesGroup, value is nil.
Example :

```
--Extracting score
score=FrontScore()
group=score.FirstStavesGroup
--- Browing groups
while group~=nil do
   print("Group starting staff # "..group.StaffBegin.Index.." and end staff #"..group.StaffEnd.Index)
   group=group.Next

end
```

Note :
  a) StavesGroups are not sorted. They are linked in the order they have been inserted.
  b) You can browse StavesGroups through Next and Previous fields, but you can also access a StavesGroup at a given index through, for example, Score.StavesGroups (page 219)[index]

StavesGroup.Previous

Type : StavesGroup
Description : Previous StavesGroup
If there is no previous StavesGroup, value is nil.
Note :
a) StavesGroups are not sorted. They are linked in the order they have been inserted.
b) You can browse StavesGroups through Next and Previous fields, but you can also access a StavesGroup at a given index through, for example, Score.StavesGroups (page 219)[index]

StavesGroup.Index

Type : Number
Description : Index of this StavesGroup
Example :

```
-- Displays the index of the 2nd StavesGroup of a staff, if exists
score=FrontScore()
group=score.StavesGroups[2]
if group~=nil then
	print(group.Index) -- Displays "2"
end
```

4.3.6.1.2. Staves group title

StavesGroup.Title

Type : String
Description : Staves group "long" title.
Note : You cannot use this field if the score is protected by its author.

StavesGroup.TitleXOffset

Type : Number
Description : Staff "long" title X offset (72 dpi pixel) from its regular position

StavesGroup.TitleYOffset

Type : Number
Description : Staff "long" title Y offset (72 dpi pixel) from its regular position

StavesGroup.TitleFont

Type : String
Description : Staff title font name

StavesGroup.TitleFace

Type : String
Description : Staff title font face (See <TextStyle> object for an explanation of Face string)

StavesGroup.TitleSize

Type : Number
Description : staves group title font size (72 dpi pixel)

StavesGroup.TitleRGBColor

Type : String
Description : staves group title font color (RGB components)
EXAMPLE : -- Sets the staves group title to red
myStaff.TitleRGBColor="FF0000"

StavesGroup.TitleRotation

Type : Number
Description : Rotation in degree.

StavesGroup.ShortTitle

Type : String
Description : staves group "short" title.
Short title is an abbreviated version of the staves group title, to be displayed on all systems but the first(indented) one.
If empty, then the int32_t name is used instead.

StavesGroup.ShortTitleXOffset

Type : Number
Description : staves group "short" title X offset (72 dpi pixel) from its regular position

StavesGroup.ShortTitleYOffset

Type : Number
Description : staves group "short" title Y offset (72 dpi pixel) from its regular position

StavesGroup.ShortTitleFont

Type : String
Description : Staff group "short" title font name

StavesGroup.ShortTitleFace

Type : String
Description : Staff group "short" title font face (See <TextStyle> object for an explanation of Face string)

StavesGroup.ShortTitleSize

Type : Number
Description : staves group "short" title font size (72 dpi pixel)

StavesGroup.ShortTitleRGBColor

Type : String
Description : staves group "Short" title font color (RGB components)
EXAMPLE : -- Sets the staves group title to red
myStaff.TitleRGBColor="FF0000"

StavesGroup.ShortTitleRotation

Type : Number
Description : Rotation in degree.

4.3.6.1.3. Staves groups general parameters

StavesGroup.Symbol

Type : Number
Description : StavesGroup symbol, can be :
SYMBOL_BRACE a brace
SYMBOL_BRACKET a bracket
SYMBOL_SUB_BRACKET a sub-bracket
SYMBOL_BRACKET_UP a bracket but only with the up part
SYMBOL_BRACKET_DOWN a bracket but only with the down part
SYMBOL_LINE a line (in fact a bracket without up and down
SYMBOL_NONE no symbol displayed

StavesGroup.JoinMode

Type : Number
Description : StavesGroup join mode, can be :
JOIN_LINE_NONE : no line between staves
JOIN_LINE_STANDARD : a line join the staves inside the group
JOIN_LINE_MENSURSTRICH : a line join the staves, but outside the staff lines. No bar line is displayed on the staff except for a repeat symbol.

StavesGroup.AllowOnSingleStaff

Type : Boolean
Description : If true the part group symbol will be displayed even only one staff is displayed.

StavesGroup.StaffBegin

Type : Staff
Description : The staff where the staves group begins.

StavesGroup.StaffEnd

Type : Staff
Description : The staff where the staves group ends.

StavesGroup.HorizontalOffset

Type : Number
Description : Horizontal offset, in 72 dpi pixel, from the left of the staff area.

StavesGroup.VerticalTopOffset

Type : Number
Description : Vertical offset of the top end, in 72 dpi pixel, from the top of the staff area.

StavesGroup.VerticalBottomOffset

Type : Number
Description : Vertical offset of the bottom end, in 72 dpi pixel, from the bottom of the staff area.

4.3.6.2. StavesGroup methods

4.3.6.2.1. Managing

StavesGroup.Delete(...)

Description : Delete the staves group
Parameters : None
Return value : None

StavesGroup.CopyTo(...)

Description : Copy the staves group on another staves grou. You can copy staves grou even from a score to another.
Parameters : Staves group : the destination Staves group.
Return value : Boolean : true if the staves group was successfully copied.

4.3.7. BarSetting Object Type

General description of this object type : Clef, key signature and time signature display setting for a given bar in a Staff (page 252)
All these notions are named hereafter under the generic term "BarSetting"
Note: BarSettings are related to a given staff and a given bar. They are sorted in increasing order of bar number.
·Clef and key can be changed independently on each staff, at each bar.
·A time signature change affects all the staves. It can therefore be accessed through the Score.GetGlobalBarSetting(...) (page 234) method.
However, each staff can display a time signature or hide it.
BarSettings are indexed elements, related internally to the bar number.
Adding (with Staff.InsertBarSetting(...) (page 271)) or deleting (with BarSetting.Delete(...) (page 334)) BarSettings does n affect other elements, that can be kept in variables for further work.
However, adding or deleting bars shifts bar numbering, so BarSetting object variables that refer to further elements in the score can become unusable.
There is always a BarSetting object related to bar #1 of each staff. This object cannot be deleted.

Important tips :

1- Each time you need a setting for a given bar, you have to ask this question :
Does this setting always apply to all staves, or can it apply to one staff only ?
If it always applies to all staves at a time, then it is part of an object of GlobalBarSetting (page 318) type, you can get fron Score.GetGlobalBarSetting(...) (page 234)
If it can apply to only one staff, then it is part of an object of BarSetting (above) type, you can get from a Staff (page 252) object.

For example :
A time signature change always apply to all staves. It is part of a GlobalBarSetting (page 318) object type.
A clef change can apply to one staff only. It is part of a BarSetting (above) object type.

2- Even if a BarSetting object can be indended to change only one element of the clef/key pair, the other one is changed too, but set to invisible.
For instance, if, on a treble key staff you change only key signature in bar #5, there will also be an invisible clef change at that bar,
setting the clef to treble again (so no effect on score).
However, if later you want to change the staff main clef through a script (for example to bass clef), this script will be responsible for
changing also the next BarSettings that contain an invisible clef change.
Otherwise, you'll still have an invisible clef change at bar #5, that will reset the clef to treble without showing it.

3- Setting clef and key signature invisible and identical to the previous one in score results in the BarSetting object to be deleted.

4.3.7.1. BarSetting values

4.3.7.1.1. Browsing BarSettings

BarSetting.Next (R)

Type : BarSetting
Description : Next BarSetting
If there is no next BarSetting, value is nil.

BarSetting.Previous (R)

Type : BarSetting
Description : Previous BarSetting
If there is no previous BarSetting, value is nil.

4.3.7.1.2. BarSetting details

BarSetting.Clef (R)

Type : Clef (page 336)
Description : Clef change object

BarSetting.KeySignature

Type : KeySignature (page 338)
Description : Key signature change object

4.3.7.1.3. BarSettings miscellaneous values

BarSetting.Bar

Type : Number
Description : Bar at which this BarSetting is attached
Example :

```
-- Displays bar number of all BarSettings in the first staff
myScore=FrontScore()
if myScore==nil then
   print("Please load a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in this score")
   else
      BarSetting=myStaff.FirstBarSetting
      while BarSetting~=nil do
         print("BarSetting found at bar",BarSetting.Bar)
         BarSetting=BarSetting.Next
      end
   end
end
```

BarSetting.Color

Type : Number
Description : Color index of the displayed clef/key/time signature change

4.3.7.1.4. BarSettings visibility

BarSetting.IsClefVisible

Type : Boolean
Description : true if clef is displayed

BarSetting.IsKeyVisible

Type : Boolean
Description : true if key signature is displayed

BarSetting.IsTimeVisible

Type : Boolean
Description : true if time signature is displayed.
To change the time signature at a given bar, see the GlobalBarSetting (page 318) object type.

4.3.7.1.5. Managing ornaments

BarSetting.NumberOfOrnaments

Type : Number
Description : Number of ornaments attached to this bar setting. Ornament index can range from 1 to this value (included).

BarSetting.Ornaments

Type : Ornament array
Description : "Root" of ornament array for this symbol.
Cannot be used as is. Must be indexed.

BarSetting.FirstOrnament

Type : Ornament
Description : First ornament attached to this symbol (equivalent to Symbol.Ornaments[1])
nil if there is no ornament currently attached to the symbol

BarSetting.LastOrnament

Type : Ornament
Description : Last ornament attached to this symbol
nil if there is no ornament currently attached to the symbol

4.3.7.2. BarSetting methods

4.3.7.2.1. Deleting a BarSetting

BarSetting.Delete(...)

Description : Deletes the whole BarSetting (clef, key signature change and time signature display setting)
Parameters : None
Return value : true if successfully deleted
Example :

```
-- Deletes all BarSettings in the first staff
myScore=FrontScore()
if myScore==nil then
   print("Please load a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in this score")
   else
      --We preserve the current score state,
      --so the user will be able to undo the changes.
      myScore.Preserve()
      -- 1st bar setting cannot be deleted
      while myStaff.FirstBarSetting.Next do
         myStaff.FirstBarSetting.Next.Delete()
      end
   end
end
```

4.3.7.2.2. Managing

BarSetting.CopyTo(...)

Description : Copy all the BarSetting setting in another BarSetting.
Parameters : The target BarSetting
Return value : None

4.3.7.2.3. Managing ornaments

BarSetting.AddOrnament(...)

Description : Adds a new ornament.
Parameters : (All parameters are optional)
Number : Ornament type (see constants in MSDefine (page 100).mys)
Number : X offset in 72 dpi pixel from its default position, origin is the left of the bar
Number : Y offset in 1/10 of staff interligne from its default position, origin is the staff baseline
Number : Color index for this ornament. Use Graph.RGBToIndex(...) (page 521) function to get an index from Re
Green, Blue color settings.
Boolean: true if this ornament is invisible, i.e. not displayed on score.
Next parameters are only useful for ORNAMENT_TEXT ornament type
String : Text value for this ornament
String : Font name for displaying this text
Number : Font size (72 dpi pixel) for displaying this text
String : Font face for displaying this text (see TextStyle object)
Return value : Ornament : ornament object that has just been inserted (or nil if error)
Example :

```
-- Add a picture to second bar setting of first staff
myScore=FrontScore()
if myScore==nil then
   print("Please load a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in this score")
   else
      --We preserve the current score state,
      --so the user will be able to undo the changes.
      myScore.Preserve()
      barSetting=myStaff.InsertBarSetting(2,false)
      barSetting.DeleteAllOrnaments()
      ornament=barSetting.AddOrnament(4,0,60)
      ornament.BeginRecord(64,64)
      --- Draw the picture
      Graph.FrameRect(0,0,64,64);
      Graph.PaintRect(8,8,64-8,64-8);
      Graph.EraseRect(16,16,64-16,64-16);
      --- Close the recording, add the picture and set the scale
      ornament.EndRecord(10)
   end
end
```

BarSetting.DeleteAllOrnaments(...)

Description : Deletes all the ornaments of a bar setting.
Parameters : None
Return value : None
Example :

```
      barSetting.DeleteAllOrnaments()
```

4.3.8. Clef Object Type

General description of this object type : A clef. It can be a clef change in a staff, got from BarSetting.Clef (page 332)

4.3.8.1. Clef values

4.3.8.1.1. Clef settings

Clef.Type

Type : Number
Description : Clef type, as defined in MSDefine (page 100).
Can be :
·CLEF_TYPE_NONE=0
·CLEF_TYPE_TREBLE=1
·CLEF_TYPE_BASS=2
·CLEF_TYPE_TENOR=3
Note : Setting these values changes clef, but does not change the way notes are played.
Notes can be moved at a different position to match the new clef settings, but their played pitch remain the same.

To change the clef and transpose the notes accordingly, you must use Clef.Transpose(...) (page 337) method

Clef.LineOffset

Type : Number
Description : Line offset for this clef

Clef.OctaveOffset

Type : Number
Description : Octave offset for this clef

Clef.BaseNote

Type : Number
Description : Base note for this clef.
Note : IMPORTANT: this field is a fixed value that depends on the clef that has been set.
It gives the note pitch that is played when a note is written on the staff base line.
Changing it can make the score difficult to read to somebody who is not aware about the change you have made.
Above this, setting any other value (Type, LineOffset, OctaveOffset) resets the base note to its regular value. If you need to set it, you have therefore to change it AFTER having set all other parameters.

4.3.8.1.2. Clef display values

Clef.Scale

Type : Number
Description : scale for displaying the clef symbol, in percent

4.3.8.1.3. Staff display

Clef.IsDisplayStaffLines

Type : Boolean
Description : Specify if the staff lines will be displayed from this position. This allow you to create ossia or blank bar before a coda.

4.3.8.2. Clef methods

4.3.8.2.1. Changing a clef and transposing

Clef.Transpose(...)

Description : Changes a clef by another in a score and transposes the following notes
If you need the notes to be adjusted to take the new clef into account, you must use this method instead of setting the clef vlues.
Parameters : Number: Clef type
Can be :
·CLEF_TYPE_NONE=0
·CLEF_TYPE_TREBLE=1
·CLEF_TYPE_BASS=2
·CLEF_TYPE_TENOR=3
Number: Line offset (from -4 to 4)
Number: Octave offset (from -2 to 2)
Number: Transposing type
Can be :
· TRANSPOSE_PITCH_UNCHANGED
· TRANSPOSE_UPWARD
· TRANSPOSE_DOWNWARD
· TRANSPOSE_DISPLAY_UNCHANGED
Return value : true if successfully transposed
Example :

```
Include "MSDefine"
-- Changes 1st clef to bass, and transposes notes
myScore=FrontScore()
if myScore==nil then
    print("Please load a score first")
else
    myStaff=myScore.FirstStaff
    if myStaff==nil then
        print("There is no staff in this score")
    else
        --We preserve the current score state,
        --so the user will be able to undo the changes.
        myScore.Preserve()
        barSetting=myStaff.FirstBarSetting
        if barSetting~=nil then
            barSetting.Clef.Transpose(CLEF_TYPE_BASS ,0,0,TRANSPOSE_DISPLAY_UNCHANGED)
        end
    end
end
```

4.3.9. KeySignature Object Type

General description of this object type : A key signature.
It can be a key change in a staff (set of sharps or flats that is written in a piece of music to indicate the key of the piece from this point), got from BarSetting.KeySignature (page 333)

4.3.9.1. KeySignature values

4.3.9.1.1. KeySignature settings

KeySignature.Name

Type : String
Description : Key name, in anglo-saxon format :
First character : base note name (C,D,E,F,G,A,B)
Optional: accidental (#,b), omitted if natural
Optional: minor mode (m), omitted if major
For example, C#m means a C sharp base note and a minor key.
Note : Setting these values changes key signature, but does not change the way notes are played.
Notes can be moved at a different position to match the new key, but their played pitch remains the same.
To change the key and transpose notes accordingly, you must use KeySignature.Transpose(...) (below) methodTo examine custom accidentals that could have been set to a key, use KeySignature.GetAccidental(...) (page 339) method.
To examine set custom accidentals to a key, use KeySignature.SetAccidental(...) (page 340) method.

KeySignature.RelativeDisplay

Type : Number
Description : Way of displaying the new key signature relatively to the previous one in score.
Can be (see constants in MSDefine (page 100).mys)
· ABSOLUTE_KEY New key is displayed as is
· PARTIAL_RELATIVE_KEY New key cancels previous accidentals when necessary
· FULL_RELATIVE_KEY New key cancels all previous accidentals

4.3.9.2. KeySignature methods

4.3.9.2.1. Changing key signature and transposing

KeySignature.Transpose(...)

Description : Changes a key signature by another in a score and transposes the following notes
If you need the notes to be adjusted to take the new key into account, you must use this method instead of setting the key values.
Parameters : String: Key name (for example Ebm)
Number: Transposing type
Can be :
· TRANSPOSE_PITCH_UNCHANGED
· TRANSPOSE_UPWARD
· TRANSPOSE_DOWNWARD
· TRANSPOSE_DISPLAY_UNCHANGED
Optional : 7 Numbers : Accidental for each scale degree if the key signature is custom
Each of these numbers can be :
· SHARP
· NATURAL
· FLAT
· DOUBLE_SHARP
· DOUBLE_FLAT
Return value : true if successfully transposed
Example :

```
Include "MSDefine"
-- Changes 1st key to C#m, and transposes notes so that they remain at the same graphical position
myScore=FrontScore()
if myScore==nil then
   print("Please load a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in this score")
   else
      --We preserve the current score state,
      --so the user will be able to undo the changes.
      myScore.Preserve()
      barSetting=myStaff.FirstBarSetting
      if barSetting~=nil then
         barSetting.KeySignature.Transpose("C#m",TRANSPOSE_DISPLAY_UNCHANGED)
      end
   end
end
```

Note : Key name must be provided in Anglo-saxon format only (C,D,E,...).
If you prefer using Latin format (Do, Re, Mi...) you must convert the string to Anglo-Saxon format through the global functions.

4.3.9.2.2. Custom key signature

KeySignature.GetAccidental(...)

Description : Gets the accidental for a given scale degree
Parameters : Number: Scale degree (from 1=C to 7=B)
Return value : Number: Accidental for this degree :
- SHARP
- NATURAL
- FLAT
- DOUBLE_SHARP
- DOUBLE_FLAT

or nil if error.
Example :

```
Include "MSDefine"
-- Displays accidentals for each degree in the 1st key signature
local degreeName={"C","D","E","F","G","A","B"} -- degree names
myScore=FrontScore()
if myScore==nil then
   print("Please load a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in this score")
   else
      barSetting=myStaff.FirstBarSetting
      if barSetting~=nil then
         for index=1,7 do
            accidental=barSetting.KeySignature.GetAccidental(index);
            if accidental==SHARP then str="Sharp"
               elseif accidental==FLAT then str="Flat"
               elseif accidental==DOUBLE_SHARP then str="Double-sharp"
               elseif accidental==DOUBLE_FLAT then str="Double-flat"
               else str="Natural"
            end
            print("Degree",index,"(",degreeName[index],") is ",str)
         end
      end
   end
end
```

KeySignature.SetAccidental(...)

Description : Sets the accidental for a given scale degree of the key signature
Parameters : Number: Scale degree (from 1=C to 7=B)
Number: Accidental for this degree :
- SHARP
- NATURAL
- FLAT
- DOUBLE_SHARP
- DOUBLE_FLAT

Return value : None
Example :

```
Include "MSDefine"
-- Sets a random custom key signature
myScore=FrontScore()
if myScore==nil then
    print("Please load a score first")
else
    myStaff=myScore.FirstStaff
    if myStaff==nil then
        print("There is no staff in this score")
    else
        --We preserve the current score state,
        --so the user will be able to undo the changes.
        myScore.Preserve()
        barSetting=myStaff.InsertBarSetting(3)
        if barSetting~=nil then
            for index=1,7 do
                r=random(-1,1)
                if r==-1 then accidental=FLAT
                elseif r==0 then accidental=NATURAL
                else accidental=SHARP
                end
                barSetting.KeySignature.SetAccidental(index,accidental);
            end
        end
    end
end
```

4.3.10. Ornament Object Type

General description of this object type : An indexed ornament that can be attached to a Symbol (page 298), attached BarSetting (page 332) or applied to a note through a Rule (page 416)
An "Ornament" object is an indexed element. It means it refers internally to the Nth ornament of a symbol. Therefore, when a variable has been set to a given ornament, deleting an ornament that is located at a smaller index can make the variable change the ornament it refers to.

Related values :

·Symbol.NumberOfOrnaments (page 301) Number of ornaments attached to this symbol. Ornament index can rang from 1 to this value (included).
·Symbol.Ornaments (page 301) "Root" of ornament array for this symbol. Cannot be used as is. Must be indexed.
·Symbol.FirstOrnament (page 302) First ornament attached to this symbol (equivalent to Symbol.Ornaments[1]) nil there is no ornament currently attached to the symbol
·Symbol.LastOrnament (page 302) Last ornament attached to this symbol, nil if there is no ornament currently attached

These values are also available for BarSetting (page 332) and Rule (page 416) objects.

You can define an ornament is a picture taken from a binary picture file (PNG,JPG,BMP,TIF) or build on the fly in vectorial using the Graph (page 516) object.
The image data are embedded in the document.
In any case, it is recommanded to use large image and use a scale factor to reduce it: this will increase the printing quality.

4.3.10.1. Ornament values

4.3.10.1.1. Browsing Ornaments

Ornament.Next

Type : Ornament
Description : Next ornament in the symbol
If there is no next ornament, value is nil.
Note : a) Ornaments are not sorted in symbol. They are linked in the order they have been inserted.
b) You can browse ornaments through Next and Previous fields, but you can also access an ornament at a given index through Symbol.Ornaments[index]

Ornament.Previous

Type : Ornament
Description : Previous ornament in the symbol
If there is no previous ornament, value is nil.
Note : a) ornament are not sorted in staff. They are linked in the order they have been inserted.
b) You can browse ornament through Next and Previous fields, but you can also access an ornament at a given index through Symbol.Ornaments[index]

Ornament.Index

Type : Number
Description : Index of this ornament in the symbol
Example :

```
-- Displays the index of the 2nd ornament, if exists
ornament=mySymbol.Ornaments[2]
if ornament~=nil then
    print(ornament.Index) -- Displays "2"
end
```

4.3.10.1.2. Ornament general attributes

Ornament.Type

Type : Number
Description : Ornament type (see constants in MSDefine (page 100).mys)

Ornament.Symbol

Type : Symbol
Description : Symbol the ornament is related to (can be nil if the ornament is not related to a symbol)

Ornament.Tag

Type : String
Description : Tag related to this ornament. Tags are saved in the file.
See also : Score.SearchOrnamentsByTag(...) (page 247) Score.ClearTagOrnaments(...) (page 247) Staff.SearchOrnamentsByTag(...) (page 296) Staff.ClearTagOrnaments(...) (page 296)

4.3.10.1.3. Ornament display settings

Ornament.IsInvisible

Type : Boolean
Description : true if the ornament is invisible (not displayed)

Ornament.Origin

Type : Number
Description : Display origin of this ornament. See constants in MSDefine (page 100).mys:
ORNAMENT_TOP_STAFF=128
ORNAMENT_BOTTOM_STAFF=128+64
ORNAMENT_TOP_SYMBOL=0
ORNAMENT_BOTTOM_SYMBOL=64
ORNAMENT_LEFT_SYMBOL=32
ORNAMENT_RIGHT_SYMBOL=32+64

Ornament.AlongStemDirection

Type : Boolean
Description : The ornamement position will follow the stem direction.

Ornament.XOffset

Type : Number
Description : 72 dpi pixel X offset from its regular position.

Ornament.YOffset

Type : Number
Description : 72 dpi pixel Y offset from its regular position.

Ornament.Color

Type : Number
Description : Ornament color index

Ornament.Text

Type : Styled string (page 177)
Description : Text to be displayed in Text ornament
Note : Only processed when Ornament.Type equals ORNAMENT_TEXT

Ornament.Rotation

Type : Number
Description : Rotation of text in degree
Note : Only processed when Ornament.Type equals ORNAMENT_TEXT

Ornament.Font

Type : String
Description : Font to be used for displaying a Text ornament
Note : Only processed when Ornament.Type equals ORNAMENT_TEXT

Ornament.Face

Type : String
Description : Face to be used for displaying a Text ornament (See <TextStyle> object for an explanation of Face string)
Note : Only processed when Ornament.Type equals ORNAMENT_TEXT

Ornament.Size

Type : StringNumber
Description : Font size (72 DPI pixel) for displaying a Text ornament
Note : Only processed when Ornament.Type equals ORNAMENT_TEXTOnly processed when Ornament.Type equals ORNAMENT_TEXT

4.3.10.1.4. Ornament play settings

Ornament.IsPlayed

Type : Boolean
Description : true if this ornament is played, false if only decorative.

4.3.10.1.5. Ornament miscellaneous values

Ornament.IsFramed

Type : Boolean
Description : If true, a green frame is draw around the ornament area. This value is not saved and for editing purpose only.

4.3.10.1.6. Picture

Ornament.GraphRecord

Type : Graph
Description : Current recorded graphical context (created with Ornament.BeginRecord).

Ornament.GraphicScale

Description : Related graphic scale.

4.3.10.1.7. Slide

Ornament.SlideThickness

Type : Number
Description : Slide thickness

Ornament.SlideTension

Type : Number
Description : Slide tension

4.3.10.2. Ornament methods

4.3.10.2.1. Deleting an ornament

Ornament.Delete(...)

Description : Deletes the ornament related to symbol, rule or bar setting.
Parameters : None
Return value : true if successfully deleted
Example :

```
-- Deletes all "BEND" ornaments in a symbol
ornament=mySymbol.LastOrnament
while ornament~=nil do
   -- Once ornament has been deleted, Previous member could not be retrieved
   previousOrnament=ornament.Previous
   if ornament.Type==ORNAMENT_BEND or ornament.Type==ORNAMENT_RELEASE or ornament.Type==ORNAMENT_BEND_AND_RELEASE then
      ornament.Delete()
   end
   ornament=previousOrnament
end
```

```
-- Deletes all ornaments related to bar setting in the first staff
myScore=FrontScore()
if myScore==nil then
   print("Please load a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in this score")
   else
      --We preserve the current score state,
      --so the user will be able to undo the changes.
      myScore.Preserve()
      -- 1st bar setting cannot be deleted
      barSetting=myStaff.FirstBarSetting
      while barSetting~=nil do
         while barSetting.FirstOrnament~=nil do
            barSetting.FirstOrnament.Delete()
         end
         barSetting=barSetting.Next
      end
   end
end
```

4.3.10.2.2. Managing Pictures

Ornament.AddPicture(...)

Description : Adds a Picture object to the ornament related to symbol or bar setting.
Parameters : String : Full path on the picture file. The file can be a BMP,PICT,GIF,JPEG,PNG,TIFF.
Number (Optional): The scale from 0.000001 to 2.0, by default 1.0.
Return value : Boolean
Example :

```
-- Adds a picture in the first symbol of the first staff of the frontmost score
Include "MSDefine"
myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in the document")
   else
      symbol=myStaff.FirstSymbol
      if symbol==nil then
         print("There is no symbol on first staff.")
      else
         myFileName=GetFileNameLoad("bmp,pct,jpg,png,gif")
         if(myFileName~=nil) then
            ornament=symbol.AddOrnament()
            ornament.AddPicture(myFileName,20)
         end
      end
   end
end
```

Ornament.BeginRecord(...)

Description : Beginning the recording of a picture. All following graph methods will be recorded in this picture. Only one recording can be active at the same time. All recording must be close using the Ornament.EndRecord(...) (page 346) method.
If you want to get the current graphic context where the graphic commands are recorded, you can use Ornament.GraphRecord.
Parameters : Optional :
- Number : horizontal size of the picture, in pixel (by default 100)
- Number : vertical size of the picture, in pixel (by default 100)

Return value : Boolean : true if the recording was inited, false if it failed (for example a record picture is pending)

Example :

```
theScale=100
Include "MSDefine"
-- Adds a picture in the first symbol of the first staff of the frontmost score
myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   myStaff=myScore.FirstStaff
if myStaff==nil then
   print("There is no staff in the document")
else
   symbol=myStaff.FirstSymbol
   if symbol==nil then
   print("There is no symbol on first staff.")
   else
      ornament=symbol.AddOrnament()
      ornament.BeginRecord(64*theScale,64*theScale)
      --- Draw the picture
      Graph.FrameRect(0,0,64*theScale,64*theScale);
      Graph.PaintRect(8*theScale,8*theScale,64*theScale-8*theScale,64*theScale-8*theScale);
      Graph.EraseRect(16*theScale,16*theScale,64*theScale-16*theScale,64*theScale-16*theScale);
      --- Close the recording and add the picture
      ornament.EndRecord(10/(theScale*64))
      end
   end
end
```

Ornament.EndRecord(...)

Description : End the recording of a picture and add the resulting picture to the symbol.
Parameters : None
Optional: Number : The scale from 0.000001 to 2, by default 1.0
Return value : Boolean : true if picture is added

4.3.10.2.3. Ornament play settings

Ornament.GetPlayParams(...)

Description : Gets the play parameters of an ornament related to a note.
Parameters : None
Return value : The returned values depends on the Ornament type (see constants in MSDefine (page 100).mys)
Ornament type can be retrieved through the <Ornament.Type> value.
Here is the complete list of play parameters for each ornament type :

ORNAMENT_SLIDE, ORNAMENT_SLIDE_UP and ORNAMENT_SLIDE_DOWN :
Number : Delay before slide
Number : Kind of sliding (see constants in MSDefine (page 100).mys).
SLIDE_NONE=0
SLIDE_SEMITONE=1
SLIDE_SMOOTH=2
SLIDE_PIANO=3

ORNAMENT_BEND :
Number : Delay before bend
Number : Duration of bending before reaching the new pitch
Number : Pitch offset (in 100th of semitone)
ORNAMENT_RELEASE:
Number : Duration of pitch-altered part
Number : Duration of release before reaching the regular pitch
Number : Pitch offset (in 100th of semitone)
ORNAMENT_BEND_AND_RELEASE
Number : Delay before bend
Number : Duration of bending before reaching the new pitch
Number : Duration of pitch-altered part
Number : Duration of release before returning to regular pitch
Number : Pitch offset (in 100th of semitone)
ORNAMENT_COMMA_1_SHARP to ORNAMENT_COMMA_8_FLAT
Number : Pitch alteration in 100th of semitone
ORNAMENT_VIBRATO_HIT and ORNAMENT_VIBRATO
Number : Delay before vibrato start
Number : Duration of vibrato depth increasing
Number : Vibrato period
Number : Vibrato maximum depth
ORNAMENT_TREMOLO
Number : Delay before tremolo start
Number : Duration of tremolo depth increasing
Number : Tremolo period
Number : Tremolo maximum depth
ORNAMENT_UP_GRUPPETTO, ORNAMENT_DOWN_GRUPPETTO and ORNAMENT_UP_MORDENT to ORNAMENT_DOWN_DOUBLE_MORDENT
Number : Speed
ORNAMENT_TRILL to ORNAMENT_NATURAL_TRILL
Number : Speed
Boolean : Trill inverted start (true= starts from the above note)
Number : length in percent of note duration
Number : graphical length in number of wave
ORNAMENT_VOLUME_UP
Number : Starting volume
Number : Duration of volume increase
Number : Maximum (ending) volume
ORNAMENT_VOLUME_DOWN
Number : Maximum (starting) volume
Number : Duration of volume decrease
Number : Ending volume
ORNAMENT_VOLUME_UP_AND_DOWN
Number : Starting volume
Number : Duration of volume increase
Number : Maximum (middle) volume
Number : Duration of volume decrease
Number : Ending volume
ORNAMENT_FERMATA_UP and ORNAMENT_FERMATA_DOWN
Number : Percentage of original tempo
ORNAMENT_ARPEGGIO
Number : Base delay for arpeggio (delay of first note in chord)
Number : Delay increment
Number : Arpeggio direction (low-pitched to high-pitched : 0 or high-pitched to low-pitched : 1)
Number : True if an arrow is draw to show the arpeggio orientation
Number : Arpeggio size in staff interline

ORNAMENT_BEAM_1 to ORNAMENT_BEAM_4
Number : Speed
ORNAMENT_GOLPE
Number : Volume
Number : Kind
All other ornaments have no play parameters.
Example :

```
if myOrnament.Type==ORNAMENT_BEND then
    myDelay,myDuration,myBendValue=myOrnament.GetPlayParam()
end
```

Ornament.SetPlayParams(...)

Description : Sets the play parameters of an ornament.
Parameters : Ornament parameters (see <Symbol.GetOrnamentPlayParams> for a complete list of parameters for each type of ornament)
Return value : None
Example :

```
if myOrnament.Type==ORNAMENT_BEND then
    myOrnament.SetPlayParams(10,10,200)
end
```

4.3.10.2.4. Position

Ornament.AutoPosition(...)

Description : Set the relative position of the ornement from the symbol to the default value. This avoid, for example, ornaments to overlap the staff lines or the note head.
Parameters : None
Return value : True if success

4.3.11. GraceNote Object Type

General description of this object type : An indexed grace note that can be attached to a Symbol (page 298), or applie to a note through a Rule (page 416)
An "GraceNote" object is an indexed element. It means it refers internally to the Nth grace note of a symbol. Therefore, when a variable has been set to a given grace note, deleting a grace note that is located at a smaller index can make the variable change the grace note it refers to.
Be careful, number of grace notes attached to a symbol is limited to 10

4.3.11.1. GraceNote values

4.3.11.1.1. Browsing grace notes

GraceNote.Next

Type : GraceNote
Description : Next grace note
If there is no next grace note, value is nil.
Note : You can browse grace notes through Next and Previous fields, but you can also access a grace note at a given index through Symbol.GraceNotes[index] or Rule.AppliedGraceNotes[index]

GraceNote.Previous

Type : GraceNote
Description : Previous grace note
If there is no previous grace note, value is nil.
Note : You can browse grace notes through Next and Previous fields, but you can also access a grace note at a given index through Symbol.GraceNotes[index] or Rule.AppliedGraceNotes[index]

GraceNote.Index

Type : Number
Description : Index of this grace note
Example :

```
-- Displays the index of the 2nd grace note of a symbol, if exists
grace=mySymbol.GraceNotes[2]
if grace~=nil then
   print(grace.Index) -- Displays "2"
end
```

4.3.11.1.2. Grace note attributes

GraceNote.IsAfter

Type : Boolean
Description : true if grace note is an acciacatura, i.e. located after the note, false if it is an appoggiatura, i.e. located before.

GraceNote.PitchOffset

Type : Number
Description : value of the semitone offset between the grace note and the note it is attached to.
Example :

```
-- Adding an octave grace before a note
grace=myNote.AddGraceNote()
if grace~=nil then
   grace.IsAfter=false
   grace.PitchOffset=12 -- +12 semitones = +1 octave
end
```

GraceNote.Velocity

Type : Number
Description : Value of velocity from 0 to 127. -1 Mean same velocity as the note.

GraceNote.Accidental

Type : Number
Description : preferred accidental to display the grace note. See constants in MSDefine (page 100).mys :
NATURAL=0
SHARP=1
AUTO=2
FLAT=3
DOUBLE_SHARP=4
DOUBLE_FLAT=5

GraceNote.Divider

Type : Number
Description : Divider of the regular grace note displayed and played duration
Note : grace notes have a regular way of being displayed and played. This value enables to let the grace note being played/displayed shorter than this default setting.
0 means regular duration
1 means halved duration
2 means duration divided by 4
3 means duration divided by 8
etc

GraceNote.IsInChord

Type : Boolean
Description : true if grace note is in chord with next grace

GraceNote.IsParenthesis

Type : Boolean
Description : true if the grace has parenthesis around the head

4.3.11.2. GraceNote methods

4.3.11.2.1. Deleting a grace note

GraceNote.Delete(...)

Description : Deletes the grace note
Parameters : None
Return value : true if successfully deleted

4.3.12. Dynamic Object Type

General description of this object type : Dynamics, Tempo changes, Octave shifting, Pedal marks in a staff
All these notions are named hereafter under the generic term "dynamic"
Dynamic type constants:
DYNAMIC_CRESCENDO=1
DYNAMIC_DECRESCENDO=2
DYNAMIC_PPPPPP=26
DYNAMIC_PPPPP=27
DYNAMIC_PPPP=28
DYNAMIC_PPP=3
DYNAMIC_PP=4
DYNAMIC_P=5
DYNAMIC_MP=6
DYNAMIC_MF=7
DYNAMIC_F=8
DYNAMIC_FF=9
DYNAMIC_FFF=10
DYNAMIC_FFFF=31
DYNAMIC_FFFFF=30
DYNAMIC_FFFFFF=29
DYNAMIC_SF=11
DYNAMIC_FZ=12
DYNAMIC_FP=13
DYNAMIC_FPP=14
DYNAMIC_SFZ=15
DYNAMIC_SFFZ=16
DYNAMIC_SFP=25
DYNAMIC_SFPP=17
DYNAMIC_RF=32
DYNAMIC_RFZ=33
DYNAMIC_OCTAVE_UP=18 -- 8va
DYNAMIC_OCTAVE_DOWN=19 -- 8vb
DYNAMIC_DOUBLE_OCTAVE_UP=20 --15ma
DYNAMIC_DOUBLE_OCTAVE_DOWN=21 --15mb
DYNAMIC_TEMPO=22
DYNAMIC_PEDAL_ON=23
DYNAMIC_PEDAL_OFF=24

4.3.12.1. Dynamic values

4.3.12.1.1. Browsing Dynamics

Dynamic.Next

Type : Dynamic
Description : Next dynamic
If there is no next dynamic, value is nil.
Note :
a) dynamics are not sorted. They are linked in the order they have been inserted.
b) You can browse dynamics through Next and Previous fields, but you can also access a dynamic at a given index through, for example, Staff.Dynamics (page 261)[index]

Dynamic.Previous

Type : Dynamic
Description : Previous dynamic
If there is no previous dynamic, value is nil.
Note :
a) dynamics are not sorted. They are linked in the order they have been inserted.
b) You can browse dynamics through Next and Previous fields, but you can also access a dynamic at a given index through, for example, Staff.Dynamics (page 261)[index]

Dynamic.Index

Type : Number
Description : Index of this dynamic
Example :

```
-- Displays the index of the 2nd dynamic of a staff, if exists
dyn=myStaff.Dynamics[2]
if dyn~=nil then
   print(dyn.Index) -- Displays "2"
end
```

4.3.12.1.2. Dynamics general parameters

Dynamic.Type

Type : Number
Description : Dynamic type (see constants in MSDefine (page 100).mys)
Note : When type is changed, play parameters for the dynamic are reset to default value.
So you must change the dynamic type before setting its play values.

Dynamic.IsSelected

Type : Boolean
Description : true if the dynamic is currently selected
Example :

```
-- Displays the type of all selected dynamics
dyn=myStaff.FirstDynamic
while dyn~=nil do
   if dyn.IsSelected then print("Selected : ",dyn.Type) end
   dyn=dyn.Next
end
```

Note : "Selected" means the dynamic is framed in red, after having been clicked with the selection tool.
A dynamic that belongs to the selection range is not considered as selected.

4.3.12.1.3. Dynamics display settings

Dynamic.AreaLeft

Type : Number
Description : Left bound of the dynamic, in time position
Note : must be smaller that AreaRight

Dynamic.AreaRight

Type : Number
Description : Right bound of the dynamic, in time position
Example :

```
-- Moves the first dynamic of a staff at the beginning of bar 1 (in 4/4 time signature)
dyn=myStaff.FirstDynamic
if dyn~=nil then
   dynWidth=dyn.AreaRight-dyn.AreaLeft
   dyn.AreaLeft=DURATION_WHOLE
   dyn.AreaRight=dyn.AreaLeft+dynWidth
end
```

Note : must be greater that AreaLeft

Dynamic.AreaTop

Type : Number
Description : Upper bound of the dynamic, in 72 DPI pixel relatively to the staff baseline (positive values to the bottom)
Note : must be smaller that AreaBottom

Dynamic.AreaBottom

Type : Number
Description : Lower bound of the dynamic, in 72 DPI pixel relatively to the staff baseline (positive values to the bottom)
Example :

```
-- Moves the first dynamic of a staff just under the base line
dyn=myStaff.FirstDynamic
if dyn~=nil then
    dynHeight=dyn.AreaBottom-dyn.AreaTop
    dyn.AreaTop=1
    dyn.AreaBottom=dyn.AreaTop+dynHeight
end
```

Note : must be smaller that AreaTop

Dynamic.AreaLeftAbsolutePixel

Type : Number
Description : Left bound of the dynamic, in absolute pixel position

Dynamic.AreaRightAbsolutePixel

Type : Number
Description : Righ bound of the dynamic, in absolute pixel position

Dynamic.LeftPixelOffset

Type : Number
Description : Pixel offset (72 DPI) to be added to the left bound of the dynamic frame to get its actual display location.
Because AreaLeft is provided in time position, some locations on staff (for example, the space over a clef or key change) could not be reached.
This pixel offset enables to make the object be displayed at such locations.

Dynamic.RightPixelOffset

Type : Number
Description : Pixel offset (72 DPI) to be added to the right bound of the dynamic frame to get its actual display location.
Because AreaRight is provided in time position, some locations on staff (for example, the space over a clef or key change) could not be reached.
This pixel offset enables to make the object be displayed at such locations.
Example :

```
-- Makes the first dynamic of a staff to be displayed over the clef
dyn=myStaff.FirstDynamic
if dyn~=nil then
    dynWidth=dyn.AreaRight-dyn.AreaLeft
    dyn.AreaLeft=0
    dyn.AreaRight=dyn.AreaLeft+dynWidth
    dyn.LeftPixelOffset=-50
    dyn.RightPixelOffset=-50
end
```

Dynamic.Color

Type : Number
Description : Color index for displaying this dynamic.

Dynamic.Title

Type : String
Description : Displayed string for this dynamic. You can use the following patterns: $p to display the "p" dynamics symbol, or $f, $m, $z. The $x pattern make the default dynamics symbol to be displayed
For example, "$x the second time" will display "ppp the second time" if the dynamic is a "ppp".

Dynamic.TitleFont

Type : String
Description : Change title font

Dynamic.TitleFace

Type : String
Description : Change title face (See <TextStyle> object for an explanation of Face string)

Dynamic.TitleSize

Type : Number
Description : Change title size (72 dpi pixel)

Dynamic.TitleColor

Type : string
Description : Get/set the color for title text. The color is coded
"rrggbb" where "rr" is the red value from 00 to FF, "gg" the green value, "bb" the blue value.

Dynamic.Rotation

Type : Number
Description : Dynamic rotation in degree (0-90 or 270-360). (Some dynamics don't manage rotation)

Dynamic.IsPrinted

Type : Boolean
Description : True if is dynamic is printed.

4.3.12.1.4. Dynamics play settings

Dynamic.IsPlayed

Type : Boolean
Description : true if the dynamic is played, false if it has no effect on music

Dynamic.AppliedTo

Type : Number
Description : how the dynamic is applied: 0 = to the staff only, 1 = to staff + merged staves, 2 = on all staves in the group (system), 3 = to all staves in the score

Dynamic.IsGlobal

Type : Boolean
Description : true if the dynamic has a global effect, i.e. applies to all staves. False if only applied to the embedding staff.
Obsolete, kept for backward compatibility, replaced by Dynamic.AppliedTo

Dynamic.IsAppliedToMergedStaves

Type : Boolean
Description : When several staves are merged together, a "true" value means that the dynamic or pedal applies to all the merged staves.
False means that only the staff on which the dynamic or pedal is written is affected.
Obsolete, kept for backward compatibility, replaced by Dynamic.AppliedTo

Dynamic.IsAppliedToWholeGroup

Type : Boolean
Description : When two staves are joined by a brace, a "true" value means that the dynamic or pedal applies to both.
False means that only the staff on which the dynamic or pedal is written is affected.
In regular music notation, a dynamic applies only to the first staff of a braced group when written above it, to both when
written between the two staves, and to the last staff when written below it.
Obsolete, kept for backward compatibility, replaced by Dynamic.AppliedTo

Dynamic.IsCumulative

Type : Boolean
Description : true if the dynamic is cumulative

Dynamic.Value

Type : Number
Description : value for power marks (DYNAMIC_PPP to DYNAMIC_SFPP) or Pedal (DYNAMIC_PEDAL)
Note : Only applies to such objects. Value is nil for other objects (and cannot be set).

Dynamic.StartValue

Type : Number
Description : Start value for crescendo/decrescendo (DYNAMIC_CRESCENDO or DYNAMIC_DECRESCENDO)
Note : Only applies to such objects. Value is nil for other objects (and cannot be set).
Dynamic type changes automatically between DYNAMIC_CRESCENDO and DYNAMIC_DECRESCENDO according to the respective values of StartValue and EndValue

Dynamic.EndValue

Type : Number
Description : End value for crescendo/decrescendo (DYNAMIC_CRESCENDO or DYNAMIC_DECRESCENDO)

Note : Only applies to such objects. Value is nil for other objects (and cannot be set).
Dynamic type changes automatically between DYNAMIC_CRESCENDO and DYNAMIC_DECRESCENDO according to the respective values of StartValue and EndValue

4.3.12.1.5. Tempo-specific settings

These fields are valid for tempo change objects only.
Reading them on any other kind of dynamics will get a nil value, setting them will have no effect

You can change the general tempo of the score and the values of all the tempo objects using the Score.SetGeneralTempo(...) (page 248) method.

Dynamic.TempoTitle

Type : String
Description : Displayed string for this tempo change

Dynamic.TempoFont

Type : String
Description : Tempo change title font

Dynamic.TempoFace

Type : String
Description : Tempo change title face (See <TextStyle> object for an explanation of Face string)

Dynamic.TempoSize

Type : Number
Description : Tempo change title size (72 dpi pixel)

Dynamic.TempoReferenceNote

Type : Number
Description : Tempo change reference note duration

Dynamic.TempoAlignment

Type : Number
Description : Tempo change alignment (see constants in MSDefine (page 100).mys)
Enables to hide note symbol and tempo value by setting TEMPO_HIDDEN. In this case only the content of Dynamic.TempoTitle (above) is displayed.

Dynamic.IsTempoDrawLine

Type : Boolean
Description : true if a line is drawn at the bottom of the tempo object

Dynamic.TempoValue

Type : Number (can be non-integer)
Description : Tempo value for this tempo change

Dynamic.TempoNoteScale

Type : Number
Description : Scale of the little note displayed in the tempo.

Dynamic.Scale

Type : Number
Description : Scale of the dynamic from 10% to 800%.

Dynamic.IsTempoSoftChange

Type : Boolean
Description : true if tempo changes smoothly from its previous value to the new one. You can define the curve using
$JDynamic.TempoCurvePoint1] and $JDynamic.TempoCurvePoint2]

Dynamic.TempoCurvePoint1

Type : Number
Description : First curve point for smooth tempo change. (Y coordinate in percent)

Dynamic.TempoCurvePoint2

Type : Number
Description : Seconf curve point for smooth tempo change. (Y coordinate in percent)

4.3.12.2. Dynamic methods

4.3.12.2.1. Conditionnal

Dynamic.GetActivationState(...)

Description : For a dynamic specify if the dynamic will be enabled at this time
Parameters : Number : Time to check from 1 to 31
Return value : true if activated

Dynamic.SetActivationState(...)

Description : For a dynamic specify if the dynamic will be enabled at this time
Parameters : Number : Time to set from 1 to 30, Boolean : value to set
Return value : None

4.3.12.2.2. Deleting a dynamic

Dynamic.Delete(...)

Description : Deletes the dynamic
Parameters : None
Return value : true if successfully deleted
Example :

```
-- Deletes all dynamics in a staff, starting from the end to avoid index problems
dyn=myStaff.LastDynamic
while dyn~=nil do
   -- Once dynamic has been deleted, Previous member could not be retrieved
   previousDyn=dyn.Previous
   dyn.Delete()
   dyn=previousDyn
end
```

4.3.12.2.3. Editing a dynamic

Dynamic.Edit(...)

Description : Opens a window that enables the user to apply changes to a dynamic.
Parameters : None
Return value : true if successfully edited
Example :

```
-- Creates a new Tempo object in a staff and edits it
tempo=myStaff.AddDynamic(DYNAMIC_TEMPO)
if tempo~=nil then
    tempo.Edit()
end
```

4.3.12.2.4. Managing

Dynamic.CopyTo(...)

Description : Copy the dynamic on another dynamic. You can copy dynamic even from a score to another.
Parameters : Dynamic : the destination dynamic.
Return value : Boolean : true if the dynamic was successfully copied.

4.3.13. Picture Object Type

General description of this object type : Graphical object that can be added on a Staff (page 252)
A "Picture" object is an indexed element. It means it refers internally to the Nth object of this type in a staff. Therefore, deleting such an object can make the "Picture" variables that refer to another picture further in the list change the element they refer.

4.3.13.1. Picture values

4.3.13.1.1. Browsing Pictures

Picture.Next

Type : Picture
Description : Next Picture
If there is no next Picture, value is nil.
Note :
a) Pictures are not sorted. They are linked in the order they have been inserted.
b) You can browse Pictures through Next and Previous fields, but you can also access a Picture at a given index through, for example, Staff.Pictures (page 263)[index]

Picture.Previous

Type : Picture
Description : Previous Picture
If there is no previous Picture, value is nil.
Note :
a) Pictures are not sorted. They are linked in the order they have been inserted.
b) You can browse Pictures through Next and Previous fields, but you can also access a Picture at a given index through, for example, Staff.Pictures (page 263)[index]

Picture.Index

Type : Number
Description : Index of this Picture
Example :

```
-- Displays the index of the 2nd Picture of a staff, if exists
pict=myStaff.Pictures[2]
if pict~=nil then
    print(pict.Index) -- Displays "2"
end
```

4.3.13.1.2. General parameters

Picture.Type

Type : Number
Description : Picture type. Can be :
PICTURE_TYPE_STANDARD=0
PICTURE_TYPE_EMBEDDED_PICT=1
PICTURE_TYPE_EMBEDDED_BMP=2

Picture.GraphicLineKind

Type : Number
Description : Kind of the graphic line : SOLID_LINE, DASHED_LINE, DOTTED_LINE, WAVY_LINE, SOLID_CURVE, DOTTED_CURVE.
If the picture is not a graphic line, the value nil is returned.

Picture.IsSelected

Type : Boolean
Description : true if the picture is currently selected
Note : "Selected" means the picture is framed in red, after having been clicked with the selection tool.
A picture that belongs to the selection range is not considered as selected.

Picture.Staff

Type : Staff
Description : Staff that includes the picture.

Picture.IsRelatedToOrnament

Type : Boolean
Description : true if the picture is related to an ornament

4.3.13.1.3. Position and size

Picture.AreaLeft

Type : Number
Description : Left bound of the picture, in time position
Note : must be smaller that AreaRight

Picture.AreaRight

Type : Number
Description : Right bound of the picture, in time position
Note : must be greater that AreaLeft

Picture.AreaLeftAbsolutePixel

Type : Number
Description : Left bound of the picture, in absolute pixel position

Picture.AreaRightAbsolutePixel

Type : Number
Description : Righ bound of the picture, in absolute pixel position

Picture.AreaTop

Type : Number
Description : Upper bound of the picture, in 72 DPI pixel relatively to the staff baseline (positive values to the bottom)
Note : must be smaller that AreaBottom

Picture.AreaBottom

Type : Number
Description : Lower bound of the picture, in 72 DPI pixel relatively to the staff baseline (positive values to the bottom)
Note : must be greater that AreaTop

Picture.HorizontalScale

Type : Number
Description : Horizontal scale of the picture. From 1 to 4000. 100 mean standard scale.

Picture.VerticalScale

Type : Number
Description : Verticale scale of the picture. From 1 to 4000. 100 mean standard scale.

Picture.HorizontalPixelSize

Type : Number
Description : Horizontal size in pixel of the picture..

Picture.VerticalPixelSize

Type : Number
Description : Horizontal size in pixel of the picture..

Picture.GraphicLineBeginTime

Type : Number
Description : Left bound of the line, in time position
Note : must be smaller that GraphicLineEndTime

Picture.GraphicLineBeginYPos

Type : Number
Description : Y position, in 72 dpi of the start of the graphical line, relative to the staff base line

Picture.GraphicLineEndTime

Type : Number
Description : Right bound of the line, in time position
Note : must be greater that AreaLeft

Picture.GraphicLineEndYPos

Type : Number
Description : Y position, in 72 dpi of the end of the graphical line, relative to the staff base line

4.3.13.1.4. Display

Picture.GraphicLineBeginKind

Type : Number
Number
Description : Index of the general color related to the picture.
Kind of beginning of line : END_NONE, END_JOG_UP, END_JOG_DOWN, END_JOG_BOTH, END_ARROW.
If the picture is not a graphic line, the value nil is returned.

Picture.GraphicLineEndKind

Type : Number
Description : Kind of end of line : END_NONE, END_JOG_UP, END_JOG_DOWN, END_JOG_BOTH, END_ARROW.
If the picture is not a graphic line, the value nil is returned.

Picture.GraphicLineWidth

Type : Number
Description : Width of the graphic line
If the picture is not a graphic line, the value nil is returned.

Picture.GraphicLineCurveHeight

Type : Number
Description : If the picture is a graphic line, curve type, provide the vertical size.
If the picture is not a graphic line, the value nil is returned.

Picture.GraphicLineCurveWidth

Type : Number
Description : If the picture is a graphic line, curve type, provide the horizontal size.
If the picture is not a graphic line, the value nil is returned.

Picture.GraphicLineBracketSize

Type : Number
Description : Width of the jog of the graphic line
If the picture is not a graphic line, the value nil is returned.

Picture.Color

Type : Number
Description : Color of the picture of graphic line

Picture.Rotation

Type : Number
Description : Rotation angle in degrees (integer).

Picture.IsFixedScale

Type : Boolean
Description : If true the horizontal picture scale will not be affected by the bar scale.

Picture.IsEraseBackground

Type : Boolean
Description : If true the picture background will be erased before drawing the picture.

4.3.13.1.5. Tags

Picture.Tag

Type : String
Description : Tag related to this pictures. Tags are saved in the file.

4.3.13.2. Picture methods

4.3.13.2.1. Deleting a Picture

Picture.Delete(...)

Description : Deletes the Picture. If the picture is related to an ornament, the ornament is also deleted.
Parameters : None
Return value : true if successfully deleted
Example :

```
-- Deletes all Pictures in a staff, starting from the end to avoid index problems
pict=myStaff.LastPicture
while pict~=nil do
   -- Once picture has been deleted, Previous member could not be retrieved
   previousPict=pict.Previous
   pict.Delete()
   pict=previousPict
end
```

4.3.13.2.2. Drawing a Picture

Picture.DrawPicture(...)

Description : Draw the Picture
Parameters : Numbers: horizontal pixel position, Vertical pixel position
Optional : Horizontal pixel size, Vertical pixel size
Return value : none

4.3.13.2.3. Selecting a Picture

Picture.Select(...)

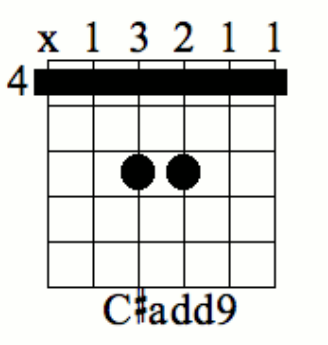
Description : Opens a window that enables the user to apply changes to a Picture.
Parameters : None
Return value : true if successfully edited
Example :

```
-- Select a picture
local myScore,myStaff,mypict

myScore=FrontScore()
if myScore~=nil then
   myStaff=myScore.FirstStaff
   if myStaff~=nil then
      mypict=myStaff.Pictures[1]
      if mypict~=nil then
         mypict.Select()
      end
   end
end
```

4.3.14. Diagram Object Type

General description of this object type : Chord diagram for fretted instrument that can be displayed on a staff.

Here is how a diagram look like :



This is a typical chord diagram C#add9 for guitar with three finger, one is a barre at cell #4.
The numerical notation of this diagram is "X45664".

The number at the bottom specify what finger press the strings.
String 1 of a chord diagram is the string at the right of the chord diagram, typicaly string with the hightest pitch.
Finger 1 press strings 1 to 6, cell #1, this is a barre.
Finger 2 and 3, press cell #3, strings #3 and #4.

There is different way to use chord diagrams, as object related to the staff, as object displayed on the chord line of a staff, as object displayed on the grid line of a score.
You can duplicate diagrams objects from staff to grid, grid to independant diagrams, etc.
The only restriction is a script is not allowed to change the global diagram database.

How Harmony Assistant choose what chord diagram to display in a chord grid or a chord line?
From the chord name (computed or user defined) a search is done first in the chord diagrams database related to the staff or the score then in the global chord diagrams database related to the application.
You can use the name you want for chord in chord diagram, the chord name must begin by letter 'A' to 'G'.

·**Object related to the staff**:
A "Diagram" object is an indexed element. It means it refers internally to the Nth object of this type in a staff.
Therefore, deleting such an object can make the "Diagram" variables that refer to another Diagram further in the list change the element they refer.
Related methods and values:
Staff.AddDiagram(...) (page 277), Staff.NumberOfDiagrams (page 257)

·**Object related to the chord line of a staff**:

A script can't modify the global chord database, but can duplicate a global chord diagram as a local copy, related to the staff.
In the staff database, if several diagrams have the same name, the first chord diagram is taken.
In the global database, the "prefered" chord diagram is taken.
Deleting such an object can make the "Diagram" variables that refer to another Diagram further in the list change the element they refer.
Related methods and values:
Staff.AddChordDiagram(...) (page 279), Staff.GetChordDiagramFromName(...) (page 279),
Staff.GetChordDiagramFromIndex(...) (page 279),
Staff.DeleteChordDiagrams(...) (page 279),Staff.DuplicateChordDiagrams(...) (page 279),Staff.NumberOfChordDiagrams

Some tips :
To extract the chord diagram from a chord line use Staff.GetChordDiagram(...) (page 278) providing the index
To test if the chord diagram is global or local use Diagram.IsLocal (page 364)
To duplicate a global or local chord diagram to local use Diagram.Duplicate(...) (page 366)
To duplicate the entire chord database from a staff to another staff use Staff.DuplicateChordDiagrams(...) (page 279)
To create from blanc a new local chord diagram use Staff.AddChordDiagram(...) (page 279)
To create fingering in this new chord diagram use Diagram.AddFinger(...) (page 365) or more quickly Diagram.StringValue 365)
(Never forget the fingering is relative to the first case Diagram.FirstCase (page 365))
To quickly convert fingering to or from text use Diagram.StringValue (page 365)
Finaly, you can delete all chords diagrams related to a staff using Staff.DeleteChordDiagrams(...) (page 279)

·**Object related to the chord grid of a score**:
These chords diagrams are displayed in the chord grid of the score. There are managed as the chord diagram related to a staff.
Related methods and values:
Score.AddChordDiagram(...) (page 225), Score.GetChordDiagramFromName(...) (page 225), Score.GetChordDiagramFromIndex(...) (page 225), Score.DeleteChordDiagrams(...) (page 225),Score.DuplicateChordDiagrams(...) (page 225),Score.NumberOfChordDiagr 201)

4.3.14.1. Diagram values

4.3.14.1.1. Browsing Diagrams

Diagram.Next

Type : Diagram
Description : Next Diagram
If there is no next Diagram, value is nil.
Note :
a) Diagrams are not sorted. They are linked in the order they have been inserted.
b) You can browse Diagrams through Next and Previous fields, but you can also access a Diagram at a given index through, for example, Staff.Diagrams (page 258)[index]

Diagram.Previous

Type : Diagram
Description : Previous Diagram
If there is no previous Diagram, value is nil.
Note :
a) Diagrams are not sorted. They are linked in the order they have been inserted.
b) You can browse Diagrams through Next and Previous fields, but you can also access a Diagram at a given index through, for example, Staff.Diagrams (page 258)[index]

Diagram.Index

Type : Number
Description : Index of this Diagram
Example :

```
-- Displays the index of the 2nd Diagram of a staff, if exists
diagram=myStaff.Diagrams[2]
if diagram~=nil then
    print(diagram.Index) -- Displays "2"
end
```

4.3.14.1.2. Diagrams general parameters

Diagram.ChordNumber

Type : Number
Description : Diagram type (see constants in MSDefine.mys)

Diagram.IsLocal

Type : Boolean
Description : True if the diagram is related to the staff, false if it is a default disgaram. See diagram.Duplicate()

Diagram.ChordName

Type : String
Description : Diagram chord name

Diagram.NumberOfStrings

Type : Number
Description : Number of strings

4.3.14.1.3. Diagrams fingering

Diagram.FirstCase

Type : Number
Description : First case? This is the case of the barre. 0 if no barre.

Diagram.StringValue

Type : String
Description : Fingers definitions as a string. This is the Numeric Chord Notation. For example guitar Em is 022000.
Each number is a cell from left (6th string) to right (1th string).
0 mean empty string, x mean mutted string (no sound)
Please note, this kind of notation don't allow to notate cell over 9th cell.
Getting and setting this value don't guarantee the result will be identical.

Diagram.NumberOfFingers

Type : Number
Description : Number of fingers from 0 to 5.

4.3.14.2. Diagram methods

4.3.14.2.1. Deleting a Diagram

Diagram.Delete(...)

Description : Deletes the Diagram
Parameters : None
Return value : true if successfully deleted
Example :

```
-- Deletes all Diagrams in a staff, starting from the end to avoid index problems
diagram=myStaff.LastDiagram
while diagram~=nil do
    -- Once Diagram has been deleted, Previous member could not be retrieved
    previousDiagram=diagram.Previous
    diagram.Delete()
    diagram=previousDiagram
end
```

4.3.14.2.2. Editing a Diagram

Diagram.Edit(...)

Description : Opens a window that enables the user to apply changes to a Diagram.
Parameters : None
Return value : true if successfully edited
Example :

```
-- Creates a new Diagram object in a staff and edits it
diagram=myStaff.AddDiagram()
if diagram~=nil then
    diagram.Edit()
end
```

4.3.14.2.3. Fingering

Diagram.AddFinger(...)

Description : Add a new finger to the chord diagram
Parameters :
Number : cell on the neck (1 based)
Number : first string
Number : last string, optional, if omited first string
Return value : true if added, false if error

Diagram.ClearFingers(...)

Description : Delete all finger to the chord diagram related to the chord line.
Parameters : None
Return value : None

Diagram.GetFinger(...)

Description : Get the cell of the finger and the strings related to the finger
For example if a finger press cell 4 on first string, the result will be 4,1,1
If the finger is a barre cell 1 the result will be 1,1,6
Parameters :
Number : finger index from 1 to 5
Return value :
Number : case relative to FirstCase (-1 if not defined)
Number : first string from 1 to number of strings
Number : last string
Example :

```
local staff
local diagram
staff=FrontScore().FirstStaff
diagram=staff.FirstDiagram
print("First case:".. diagram.FirstCase)
for i=1,diagram.NumberOfFingers do
   case,string1,string2=diagram.GetFinger(i)
   print("case "..case.." from string "..string1.." to "..string2)
end
```

Diagram.GetFingerNumberForString(...)

Description : Get the finger name for a string ("X" if string is not played)
If a there is no finger (open string) the result is "0".
If there is a finger the result will be "1" to "4")
Parameters :
Number : String number 1 to NumberOfString
Return value :
String : finger name

Diagram.GetCellFromString(...)

Description : Get the played cell from a string in this chord diagram
Parameters :
Number : String number (string 1 is the string at the right of the chord diagram)
Return value :
Number : Cell -2 if string not exist, -1 if string is not played, 0 if string played empty, other : played cell number on the neck
Number : Played pitch (-1 if not played)

4.3.14.2.4. Miscellaneous

Diagram.Duplicate(...)

Description : Copy the chord diagram in the staff chords diagram database
Parameters : (optional) Score (page 198) target score
 (optional) Staff (page 252) target staff
If staff is omitted the chord diagram is copied to the chord grid
If score is omitted the chord diagram is copied to the source staff chord diagram
Return value : Diagram
Example :

```
--Convert all chord diagrams in a chord line to local
local staff
local diagram

score=FrontScore()
staff=score.FirstStaff
staff.DeleteChordDiagrams()
for i=1,100 do
    diagram=staff.GetChordDiagram(i)
    if diagram~=nil then
        if staff.GetChordDiagramFromName(diagram.ChordName)==nil then
            diagram=diagram.Duplicate()
            print("Duplicate "..diagram.StringValue)
        end
    end
end
```

```
--Copy the first chord diagram of the chord grid to the first staff chord line

score=FrontScore()
chord=score.GetBarChords(1)
diagram=chord[1].Diagram
diagram.Duplicate(score,score.FirstStaff)
```

```
--Copy the first chord diagram of the global database to the score chord grid

score=FrontScore()
diagram=Application.GetChordDiagramFromIndex(1)
diagram=diagram.Duplicate(score)
print(diagram.ChordName.." "..diagram.StringValue)
```

4.3.15. Tablature Object Type

General description of this object type : A Tablature object represents all the setting needed to compute and display a tablature. Typically a Tablature object is related to a Staff object type.
There is three tablature kind :

- Tablature for fretted instruments like guitar, bass, banjo, luth...
- Tablature for harmonica (diatonic or chromatic)
- Tablature for diatonic accordion

To add a tablature to a staff, use Staff.AddTablature(...) (page 295)

4.3.15.1. Tablature values

4.3.15.1.1. Appearance

Tablature.ShowStandardStaff

Type : Boolean
Description : When priority is set to tablature, true if the standard staff is displayed, false otherwise. You can use the Tablature.Activate(...) (page 372) method to change this mode.
By setting this value to false or true, if the tablature is active, the show standard staff mode is changed.

Tablature.ShowNoteDuration

Type : Boolean
Description : For Fretted instruments tablature only. True if the notes durations are displayed.

Tablature.IsAlignStem

Type : Boolean
Description : When Tablature.ShowNoteDuration (above) is true, define if the stems are aligned on the top of the staff, or displayed /
on the note.

Tablature.IsDownStem

Type : Boolean
Description : When Tablature.ShowNoteDuration (above) is true, define if the stems are up or down.

Tablature.IsDisplayRest

Type : Boolean
Description : When Tablature.ShowNoteDuration (above) is true, define if the rests are displayed or not.

Tablature.IsBeamMode

Type : Boolean
Description : When Tablature.ShowNoteDuration (above) is true. The beam mode is a special mode where the stem don't reflect the note duration but the duration minus one.

Tablature.IsDisplayOrnaments

Type : Boolean
Description : Define if ornaments related to notes are displayed or not

Tablature.IsDisplaySlurs

Type : Boolean
Description : Define if slurs and ties related to notes are displayed or not

Tablature.IsEraseBackgroundNote

Type : Boolean
Description : Define if the background of the notes will be erased before drawing the note.

Tablature.SymbolVerticalOffset

Type : Number
Description : Vertical shift of the symbols.

Tablature.IsStemCentered

Type : Boolean
Description : True if stems are centered on the note character.

Tablature.StemVerticalOffset

Type : Number
Description : Vertical shift of the stem.

Tablature.BaseLine

Type : Number
Description : Tablature base line (in 72 DPI pixels). The tablature must be active. If the tablature is not active, return nil.
Example :

```
--- Check if the tablature base line is greater than the staff height and adjust the staff height
local myScore
local myStaff

--- Getting the score and the first staff of it
myScore=FrontScore()
if(myScore~=nil) then
   myStaff=myScore.FirstStaff
   if(myStaff ~=nil) then
      -- Preserve to undo the changes
      myScore.Preserve()
      -- Activate the tablature
      myStaff.Tablature.Activate()
      if(myStaff.Tablature.BaseLine>myStaff.Height) then
            myStaff.Height=myStaff.Tablature.BaseLine
         end
   end
else
   Message("Please load a score");
end
```

Tablature.VerticalSize (R)

Type : Number
Description : Vertical size of the tablature (in 72 DPI pixels). The tablature must be active. If the tablature is not active, return nil.

Tablature.Font

Type : String
Description : Font name for fingering display.

Tablature.FontSize

Type : Number
Description : Font size. The characters showing the fingering are displayed using this size.

Tablature.FontFace

Type : Number
Description : Font face. The characters showing the fingering are displayed using this fase.

Tablature.FontColor

Type : Number (color index)
Description : Color of the characters.

Tablature.GraceFont

Type : String
Description : Font name for grace fingering display.

Tablature.GraceFontSize

Type : Number
Description : Font size. The characters showing the grace fingering are displayed using this size.

Tablature.GraceFontFace

Type : Number
Description : Font face. The characters showing the grace fingering are displayed using this fase.

Tablature.GraceFontColor

Type : Number (color index)
Description : Color of the characters.

Tablature.AdditionalFont

Type : String
Description : Additional font name for miscellaneous display.

Tablature.AdditionalFontSize

Type : Number
Description : Font size. The characters showing the fingering are displayed using this size.

Tablature.AdditionalFontFace

Type : Number
Description : Font face. The characters showing the fingering are displayed using this fase.

Tablature.AdditionalFontColor

Type : Number (color index)
Description : Color of the characters.

Tablature.CharacterColor

Type : Number (color index)
Description : Color of the characters.

Tablature.LineColor

Type : Number (color index)
Description : Color of the lines.

4.3.15.1.2. Attribute

Tablature.IsActive

Type : Boolean
Description : Return true if the tablature related to the staff is active. If false, you can't access to the other tablature values.
You can activate the tablature related to the staff using Tablature.Activate(...) (page 372) method.
By setting this value to false, if the tablature is active, the tablature is released (all settings are lost). The tablature is no longer computed and displayed.
By setting this value to true, if the tablature is not active, a new default guitar tablature is created and displayed.

Tablature.Type

Type : Number
Description : Provide the tablature type related to the staff. Return nil if no tablature are related to the staff.
- GUITAR_TABLATURE=0
- HARMONICA_TABLATURE=1
- ACCORDION_TABLATURE=2

By setting this value to a tablature type, if the tablature is active, the tablature type is changed.
If the tablature is not active, do nothing.

Tablature.Priority

Type : Boolean
Description : True if priority is set to tablature, false otherwise. You can use the Tablature.Activate(...) (page 372 method to change this mode.
By setting this value to false or true, if the tablature is active, the tablature proprity is changed.

Tablature.ApplyToMergedStaves

Type : Boolean
Description : True if the computation is applied to all merged staves

Tablature.UseOttava

Type : Boolean
Description : Ottava will be taken into account when computing fingering.

Tablature.DiamondShapeIsHarmonic

Type : Boolean
Description : Diamond shaped notes will be processed as harmonic.

4.3.15.1.3. Computing

Tablature.IsComputed

Type : Boolean
Description : When priority is not set to tablature, it the tablature is computed. You can use the Tablature.Compute(...) (page 372) method to compute it. In any case the tablature will be automatically computed needed) when the next staff display event occurs.

Tablature.NeedComputing

Type : Boolean
Description : Return true if the tablature need to be computed. You can call Tablature.Compute(...) (page 372) to compute it. Please note the tablature will be automatically computed (if needed) when the staff is displayed.
By setting this value to true, you will force a tablature computation when the staff will update next time. You can't force this value to false.

4.3.15.1.4. Instruments

Tablature.Fretted

Type : FrettedTab
Description : Special tablature setting for fretted (page 374) instrument.

Tablature.Harmonica

Type : HarmonicaTab
Description : Special tablature setting for harmonica (page 376).

Tablature.Accordion

Type : AccordionTab
Description : Special tablature setting for accordion (page 379).

4.3.15.2. Tablature methods

4.3.15.2.1. Activating

Tablature.Activate(...)

Description : If a tablature is already related to the staff, you can use this method to change the tablature type, priority and standard staff display mode.
You can use the Tablature.IsActive (page 370) value to check if the tablature is already active before calling this method.
Only standard staff can have a tablature related to.
To add a tablature to a staff, use Staff.AddTablature(...) (page 295)
Parameters : Optional :
- Number : the tablature type
  - GUITAR_TABLATURE=0 (by default)
  - HARMONICA_TABLATURE=1
  - ACCORDION_TABLATURE=2
- Boolean : true if priority is set to tablature, false if not. (false by default)
- Boolean : if the tablature has priority, define if the standard staff will be displayed or not. (true by default). The standard staff is always displayed if priority is not set to tablature.

Return value : True if a new tablature is related to the staff. False otherwise.
Example :

```
--- Relate a guitar tablature to the first staff of the front score
local myScore
local myStaff

myScore=FrontScore()
if(myScore~=nil) then
   myStaff=myScore.FirstStaff
   if(myStaff ~=nil) then
      myStaff.AddTablature()
      myStaff.Tablature.Activate()
   end
else
   Message("Please load a score");
end
```

Tablature.Deactivate(...)

Description : If a tablature is related to the staff, dispose all data related to the tablature. The tablature is no longer computed and displayed.
This is identical to : Tablature.IsActive=false
Parameters : None
Return value : True if a tablature was related to the staff and deactivated. False otherwise.

4.3.15.2.2. Computing

Tablature.Compute(...)

Description : If a tablature is related to the staff and if priority is not set to the tablature, compute the tablature. Please note the tablature will be automatically computed (if needed) when the staff is displayed.
Parameters : None
Return value : None

4.3.15.2.3. Local metrics

Tablature.SetLocalBaseLine(...)

Description : The value Tablature.BaseLine (page 369) allow you to define the tablature base line inside a staff, relatively to the top of its area.
With Tablature.SetLocalBaseLine you can adjust localy the tablature base line in a system for example. This will only affect the tablature base line in this system.
Typically, this is used in conjunction with the methods which extract system informations from the score: Score.BarRangeInSystem(...) (page 245),Score.GetStaffInSystem(...) (page 245).
After apllying all changes you want to the staff, call Staff.AdjustLocalMetrics(...) (page 274) or Score.AdjustLocalMetrics(...) (page 235)
Parameters : Number : Local tablature base line Y position (in 72 DPI pixels) relatively to the top of the staff area.
Number: First bar to apply
Number: Last bar to apply
Return value : None

Tablature.GetLocalBaseLine(...)

Description : This will provide the local tablature base line in a bar range.
Typically, this is used in conjunction with the methods which extract system informations from the score: Score.BarRangeInSystem(...) (page 245),Score.GetStaffInSystem(...) (page 245).
Parameters : Number: First bar
Number: Last bar
Return value : Number : Tablature base line (in 72 DPI pixels), nil if the value can't be extracted

4.3.15.2.4. Miscellaneous

Tablature.SearchClosestSound(...)

Description : Compute the closest sound ID which match the tuning defined.
Parameters : None
Return value : Sound ID

Tablature.SetDefaultAccordion(...)

Description : Switch the tablature to accordion if not already done and set all parameters to the default accordion.
Parameters : String : default accordion name (for example "Irish Diatonic", "Cajun in C key")
Return value : True if done

4.3.16. FrettedTab Object Type

General description of this object type : A FrettedTab object is a precise description of all the specific settings of a fretted instrument tablature. Typically it is related to a Tablature (page 368) object type.

4.3.16.1. FrettedTab values

4.3.16.1.1. Computing

FrettedTab.MaximumGap

Type : Number
Description : Maximum gap between forefinger and little finger. From 1 to 8.

FrettedTab.FretChangePenalty

Type : Number
Description : Fret changing penalty. From -100 to 200.

FrettedTab.EmptyStringPenalty

Type : Number
Description : Empty string penalty. From -100 to 200.

FrettedTab.StringChangePenalty

Type : Number
Description : String changing penalty. From -100 to 200.

FrettedTab.VoidStringPenalty

Type : Number
Description : Void string in chord penalty. From 0 to 500.

FrettedTab.CalculationDepth

Type : Number
Description : Calculation depth. From 2 to 128.

FrettedTab.IsArpeggio

Type : Boolean
Description : True if arpeggio with right hand.

FrettedTab.CapodastreFret

Type : Number
Description : Fret where the capodastre is. From 0 to 128. 0 mean no capodastre.

4.3.16.1.2. Definition

FrettedTab.ImplicitTransposition

Type : Number
Description : Implicit transposition in semitone.

FrettedTab.SpecialMode

Type : Number
Description : Special mode. Values can be :
- 0 Standard tablature
- 1 Baroque tablature
- 2 Dulcimer tablature
- 3 Shamisen tablature

FrettedTab.NumberOfStrings

Type : Number
Description : Number of strings from 1 to 16

4.3.16.1.3. Display

FrettedTab.IsDisplayTuning

Type : Boolean
Description : True if the tunning is displayed.

FrettedTab.IsDisplayTAB

Type : Boolean
Description : True if "TAB" is displayed.

4.3.16.2. FrettedTab methods

4.3.16.2.1. Strings

FrettedTab.GetString(...)

Description : Get the string definition
Parameters : Number : the string number from 1 (first string) to FrettedTab.NumberOfStrings
Return value :
- Number. Empty string pitch
- Number. Number of cells for this string
- Number. First cell for this string

Example :

```
--- Relate a guitar tablature to the first staff of the front score
--- and display the standard tuning
local myScore
local myStaff

myScore=FrontScore()
if(myScore~=nil) then
   myStaff=myScore.FirstStaff
   if(myStaff ~=nil) then
      myStaff.Tablature.Activate()
      string=1
      while(string<=myStaff.Tablature.Fretted.NumberOfStrings) do
         print("The tuning for string # "..string.." is "..myStaff.Tablature.Fretted.GetString(string))
         string=string+1
      end

   end
else
   Message("Please load a score");
end
```

FrettedTab.SetString(...)

Description : Set the string definition
Parameters : Number : the string number from 1 (first string) to FrettedTab.NumberOfStrings
- Number. Empty string pitch or nil if don't change

Optional :
- Number. Number of cells for this string or nil if don't change
- Number. First cell for this string or nil if don't change

Return value : None

4.3.17. HarmonicaTab Object Type

General description of this object type : A HarmonicaTab object is a precise description of all the specific settings of an harmonica. Typically it is related to a Tablature (page 368) object type.

4.3.17.1. HarmonicaTab values

4.3.17.1.1. Computing

HarmonicaTab.IsOptimiseBreath

Type : Boolean
Description : True if the computation is optimised for breath.

HarmonicaTab.IsComputeBend

Type : Boolean
Description : True if bend are computed.

HarmonicaTab.IsComputeOverBend

Type : Boolean
Description : True if overbend are computed.

HarmonicaTab.IsDisplayUnresolvedNotes

Type : Boolean
Description : True unresolved note are displayed with a '?'

4.3.17.1.2. Definition

HarmonicaTab.Type

Type : Number
Description : Provide the current type of harmonica definition. Constants are :

- -1 user defined
- 0 build-in diatonic
- 1 build-in chromatic
- 2 build-in melody maker
- 3 build-in minor
- 4 build-in harmonic minor
- 5 build-in 16 holes chromatic

HarmonicaTab.Name

Type : String
Description : Name of the harmonica.

HarmonicaTab.BaseNote

Type : Number
Description : Base note

HarmonicaTab.NumberOfHoles

Type : Number
Description : Number of holes from 1 to 19

HarmonicaTab.IsSlideBar

Type : Boolean
Description : True if a slide bar is present.

4.3.17.1.3. Display

HarmonicaTab.SymbolMode

Type : Number
Description : Blow and draw symbols mode :
- 0 : + & -
- 1 : up and down thin arrow
- 2 : white and black triangle
- 3 : black and white triangle

HarmonicaTab.DisplayMode

Type : Number
Description : Display mode :
- 0 : Blaw and draw on the same line
- 1 : On two lines according to accidental
- 2 : On two lines according to breath

HarmonicaTab.IsDisplayThinLines

Type : Boolean
Description : True if thin lines are displayed between lines

HarmonicaTab.InvertNumberSymbol

Type : Boolean
Description : If true the number and symbol vertical position will be inverted

HarmonicaTab.SlideAsCircle

Type : Boolean
Description : Will display the slide on a chromatic harmonica as a circle around the hole number

4.3.17.2. HarmonicaTab methods

4.3.17.2.1. Holes

HarmonicaTab.GetHole(...)

Description : Get the hole definition
Parameters :
- Number. The hole from 1 to number of holes (maximum is 20)
- Boolean. The breath : true is blow, false is draw.

Optional :
- Boolean. The slide state : true is slide on, false is slide off. The harmonica must have a slide to use this parameter.

Return value :
- Number. The note pitch. Nil if the value is not defined for this harmonica.

Example :

```
--- Relate an harmonica tablature to the first staff of the front score
--- and display the standard tuning
local myScore
local myStaff
```

```
myScore=FrontScore()
if(myScore~=nil) then
   myStaff=myScore.FirstStaff
   if(myStaff ~=nil) then
      myStaff.Tablature.Activate(1)
      hole=1
      print("The harmonica is : "..myStaff.Tablature.Harmonica.Name)
      while(hole<=myStaff.Tablature.Harmonica.NumberOfHoles) do
         print("Blowing in hole  # "..hole.." provide note pitch "..myStaff.Tablature.Harmonica.GetHole(hole,true))
         print("Drawing in hole  # "..hole.." provide note pitch "..myStaff.Tablature.Harmonica.GetHole(hole,false))
         hole=hole+1
      end

   end
else
   Message("Please load a score !");
end
```

HarmonicaTab.SetHole(...)

Description : Set the hole definition
Parameters :
- Number. Pitch to set
- Number. The hole from 1 to number of holes (maximum is 20)
- Boolean. The breath : true is blow, false is draw.

Optional :
- Boolean. The slide state : true is slide on, false is slide off. The harmonica must have a slide to use this parameter.

Return value : True if the hole was changed, false otherwise (for example the hole number exceed the number of holes)

4.3.18. AccordionTab Object Type

General description of this object type : A AccordionTab object is a precise description of all the specific settings of an accordion. Typically it is related to a Tablature (page 368) object type.

4.3.18.1. AccordionTab values

4.3.18.1.1. Definition

AccordionTab.Name

Type : String
Description : Name of the accordion (English language)

AccordionTab.NameFr

Type : String
Description : Name of the accordion (French language)

AccordionTab.Comment

Type : String
Description : Comment.

AccordionTab.PushCaption

Type : String
Description : String to display for the push.

AccordionTab.PullCaption

Type : String
Description : String to display for the pull.

AccordionTab.BassLineCaption

Type : String
Description : String to display for the bass/chord line.

AccordionTab.FirstRowRightHand

Type : Number
Description : First row for right hand.

AccordionTab.NumberOfRowRightHand

Type : Number
Description : Number of row right hand. (from 1 to 3)

AccordionTab.NumberOfRowLeftHand

Type : Number
Description : Number of row left hand. (from 1 to 3)

4.3.18.1.2. Display

AccordionTab.IsDisplayModePushPull

Type : Boolean
Description : True if the display follow the push/pull rules, false it the display follow the collumn rules.

AccordionTab.IsDisplayLines

Type : Boolean
Description : True if thin lines are displayed.

AccordionTab.PushPullChange

Type : Number
Description : Quarter notes the push/pull change (from 0 to 32)

AccordionTab.ComputeRightHandAlongLeftHand

Type : Boolean
Description : The push/pull mode of the left hand will be used to compute the right hand.

AccordionTab.LeftHandChordDuration

Type : Number
Description : In note duration. Until this duration will not end, the push/pull mode of the left hand will be taken into account for the right hand computation. If the value is -1 : always taken into account.

4.3.18.2. AccordionTab methods

4.3.18.2.1. Left Hand

AccordionTab.GetPitchLeftHand(...)

Description : Get the pitch of the note for one line of the row of the left hand, along the push/pull mode.
Parameters :
- Number. Row from 1 to AccordionTab.NumberOfRowLeftHand
- Boolean. True if push, false if pull
- Number. Line from 1 to 8

Return value :
- String. Caption of the chord ,nil if row or line out of range
- String. Pitches of the note, nil if row or line out of range

Example :

```
--- Relate a accordion tablature to the first staff of the front score
--- and display informations about the left hand
local myScore
local myStaff

myScore=FrontScore()
if(myScore~=nil) then
   myStaff=myScore.FirstStaff
   if(myStaff ~=nil) then
      myStaff.Tablature.Activate(2)
      print("The accordion is : "..myStaff.Tablature.Accordion.Name)
      for row=1,myStaff.Tablature.Accordion.NumberOfRowLeftHand do
         for line=1,8 do
            print("The left hand chord of push/pull, line  # "..line.." is "..myStaff.Tablature.Accordion.
            GetPitchLeftHand(row,true,line).." , "..myStaff.Tablature.Accordion.GetPitchLeftHand(row,false,line))
         end
      end

   end
else
   Message("Please load a score !");
end
```

AccordionTab.SetPitchLeftHand(...)

Description : Set the pitch of the note for a row and a line of the left hand, along the push/pull mode.
Parameters :
- Number. Row from 1 to AccordionTab.NumberOfRowLeftHand
- Boolean. True if push, false if pull
- Number. Line from 1 to 8
- String. Caption of the chord

Optional :
- String. Pitches of the note (list of numbers from 1 to 255 ended with 0) Maximum 36 values.

Return value : true if the row and the line are in a valid range, false otherwise.

4.3.18.2.2. Right Hand

AccordionTab.GetCaptionRightHand(...)

Description : Get the caption for each row of the right hand.
Parameters :
· Number. Row from 1 to AccordionTab.NumberOfRowLeftHand
Return value :
· String. The caption for this row.
Example :

```
--- Relate a accordion tablature to the first staff of the front score
--- and display informations about the right hand
local myScore
local myStaff

myScore=FrontScore()
if(myScore~=nil) then
   myStaff=myScore.FirstStaff
   if(myStaff ~=nil) then
      myStaff.Tablature.Activate(2)
      print("The accordion is : "..myStaff.Tablature.Accordion.Name)
      for row=1,myStaff.Tablature.Accordion.NumberOfRowRightHand do
         print("The caption of right hand row   # "..row.." is "..myStaff.Tablature.Accordion.GetCaptionRightHand(
         row))
         for line=1,16 do
            print("The pitch of  push/pull, line  # "..line.." is "..myStaff.Tablature.Accordion.GetPitchRightHand(row,
            true,line).." , "..myStaff.Tablature.Accordion.GetPitchRightHand(row,false,line))
         end
      end

   end
else
   Message("Please load a score !");
end
```

AccordionTab.SetCaptionRightHand(...)

Description : Set the pitch of the note for a given row of the right hand, along with the push/pull mode.
Parameters :
· Number. Row from 1 to AccordionTab.NumberOfRowLeftHand
· String. Caption for this row.
Return value : true if the row is in a valid range, false otherwise.

AccordionTab.GetPitchRightHand(...)

Description : Get the pitch of the note for a given row and line of the right hand, along with the push/pull mode.
Parameters :
· Number. Row from 1 to AccordionTab.NumberOfRowLeftHand
· Boolean. True if push, false if pull
· Number. Line from 1 to 16
Return value : Number. Pitch of the note or -1 if row or line are out of range

AccordionTab.SetPitchRightHand(...)

Description : Set the pitch of the note for a row and a line of the right hand, along the push/pull mode.
Parameters :
· Number. Row from 1 to AccordionTab.NumberOfRowLeftHand
· Boolean. True if push, false if pull
· Number. Line from 1 to 16
· Number. Pitch of tne note
Return value : true if the row and the line are in a valid range, false otherwise.

4.3.19. StaffText Object Type

General description of this object type : An indexed text element that can be displayed on a Staff (page 252).
A "StaffText" object is an indexed element. It means it refers internally to the Nth text of a staff.
Therefore, deleting a text can make the "StaffText" variables that refer to another text further in the list change the text element they refer.
For example, if there are three texts : "One", "Two" and "Three" in the staff, and a script variable is set to the second one ("Two"), if you delete the first text, then your variable will still be set to the remaining second text, i.e. "Three".
To add a new StaffText object use the Staff.AddText(...) (page 293) method.

4.3.19.1. StaffText values

4.3.19.1.1. Browsing Staff Text elements

StaffText.Next

Type : StaffText
Description : Next staff text in the staff
If there is no next staff text, value is nil.
Note :
a) Text elements are not sorted in staff. They are linked in the order they have been inserted.
b) You can browse staff text elements through Next and Previous fields, but you can also access an element at a given index through Staff.Text[index]

StaffText.Previous

Type : StaffText
Description : Previous staff text in the staff
If there is no previous staff text, value is nil.
Note :
a) Text elements are not sorted in staff. They are linked in the order they have been inserted.
b) You can browse staff text elements through Next and Previous fields, but you can also access an element at a given index through Staff.Text[index]

StaffText.Index

Type : Number
Description : Index of this text in the staff
Example :

```
-- Displays the index of the 2nd text, if exists
text=myStaff.Text[2]
if text~=nil then
   print(text.Index) -- Displays "2"
end
```

4.3.19.1.2. Staff Text attributes

StaffText.Type

Type : Number
Description : Text type (see constants in MSDefine (page 100).mys)
COMMENT_STAFF_TEXT : text is a comment (text added through the miscellaneous tool palette)
LYRICS_STAFF_TEXT : text is a lyrics part. Use <Staff> object type members to process lyrics in a more convenient way
CHORD_STAFF_TEXT : text is a chord line part. Use <Staff> object type members to process chords in a more convenient way
Example :

```
-- Displays the content of all "comment" texts in the current staff
text=myStaff.FirstText
while text~=nil do
   if text.Type==COMMENT_STAFF_TEXT then print(text.Content) end
   text=text.Next
end
```

Note : It is recommended NOT to set this field to another value. This can lead to unexpected results in chords or lyrics processing.

StaffText.Content

Type : Styled string (page 177)
Description : Text content (displayed string)
Example :

```
-- Displays the content of all texts in the current staff
text=myStaff.FirstText
while text~=nil do
    print(text.Content)
    text=text.Next
end
```

StaffText.Rotation

Type : Number
Description : Text rotation in degree.

StaffText.IsSelected

Type : Boolean
Description : true if the text is currently selected
Example :

```
-- Displays the content of all selected texts
text=myStaff.FirstText
while text~=nil do
    if text.IsSelected then print("Selected : ",text.Content)
    end
    text=text.Next
end
```

Note : "Selected" means the text is framed in red, after having been clicked with the selection tool.
A text that belongs to the selection range is not considered as selected.

StaffText.AreaLeft

Type : Number
Description : Left bound of the text area, in time position
Note : must be smaller that AreaRight

StaffText.AreaRight

Type : Number
Description : Right bound of the text area, in time position
Example :

```
-- Moves the first text of a staff at the beginning of bar 1 (in 4/4 time signature)
text=myStaff.FirstText
if text~=nil then
    textWidth=text.AreaRight-text.AreaLeft
    text.AreaLeft=DURATION_WHOLE
    text.AreaRight=text.AreaLeft+textWidth
end
```

Note : must be greater that AreaLeft

StaffText.AreaLeftAbsolutePixel

Type : Number
Description : Left bound of the text object, in absolute pixel position

StaffText.AreaRightAbsolutePixel

Type : Number
Description : Righ bound of the text object, in absolute pixel position

StaffText.AreaTop

Type : Number
Description : Upper bound of the text area, in 72 DPI pixel relatively to the staff baseline (positive values to the bottom)
Note : must be smaller that AreaBottom

StaffText.AreaBottom

Type : Number
Description : Lower bound of the text area, in 72 DPI pixel relatively to the staff baseline (positive values to the bottom)
Example :

```
-- Moves the first text of a staff just under the base line
text=myStaff.FirstText
if text~=nil then
    textHeight=text.AreaBottom-text.AreaTop
    text.AreaTop=1
    text.AreaBottom=text.AreaTop+textHeight
end
```

Note : must be smaller that AreaTop

StaffText.LeftPixelOffset

Type : Number
Description : Pixel offset (72 DPI) to be added to the left bound of the text frame to get its actual display location.
Because AreaLeft is provided in time position, some locations on staff (for example, the space over a clef or key change) could not be reached.
This pixel offset enables to make the text be displayed at such locations.

StaffText.RightPixelOffset

Type : Number
Description : Pixel offset (72 DPI) to be added to the right bound of the text frame to get its actual display location.
Because AreaRight is provided in time position, some locations on staff (for example, the space over a clef or key change) could not be reached.
This pixel offset enables to make the text be displayed at such locations.
Example :

```
-- Moves the first text of a staff to be displayed over the clef
text=myStaff.FirstText
if text~=nil then
    textWidth=text.AreaRight-text.AreaLeft
    text.AreaLeft=0
    text.AreaRight=text.AreaLeft+textWidth
    text.LeftPixelOffset=-50
    text.RightPixelOffset=-50
end
```

StaffText.RGBColor

Type : String
Description : Default text RGBColor.

StaffText.Font

Type : String
Description : Text font

StaffText.Face

Type : String
Description : Text face (See <TextStyle> object for an explanation of Face string)

StaffText.Size

Type : Number
Description : Text size (72 dpi pixel)

StaffText.IsEraseBackground

Type : Boolean
Description : The area background of the object will be erased

StaffText.ViewIndex

Type : Number
Description : View(s) in which the text will be visible.
Can be either the view index as provided by View.Index (page 401) if the text is to be displayed on a single view the following constants:
·-3 : displayed on every view except the general view
·-2 : displayed on every view that contains the object's staff
·-1 : displayed only on general view
· 0 : displayed in each and every view
·(Default: -2)

4.3.19.2. StaffText methods

4.3.19.2.1. Appearance

StaffText.Draw(...)

Description : Draw the text.
Parameters : None
Return value : None

4.3.19.2.2. Deleting a text

StaffText.Delete(...)

Description : Deletes the text.
Parameters : None
Return value : true if successfully deleted
Example :

```
-- Deletes all texts in a staff, starting from the end to avoid index problems
text=myStaff.LastText
while text~=nil do
    -- Once text has been deleted, Previous member could not be retrieved
    previousText=text.Previous
    text.Delete()
    text=previousText
end
```

4.3.20. Chord Object Type

General description of this object type : The chord grid is related to the score.
To adjust the way the chord grid is displayed, you can use the score values :
·Score.IsChordGridDisplayed (page 201) to hide or show the chord grid,
·Score.IsChordGridDisplayDiagrams (page 201) to display or not the guitar chord diagrams,
·Score.IsChordGridLeftAligned (page 201) to setup the chord text position.

To adjust the way the chord grid is computed, you can use the score values :
· Score.IsChordGridComputedOutsideModalArray (page 200) : define if the computation will respect the modal array or not.

To get a chord array from a chord grid related to a score use Score.GetBarChords(...) (page 224) method.
To convert a written time position to chord use Score.TimeToChord(...) (page 225) method.
Each bar of the score can be splitted in 1,2,3 or 4 chords.
Each chord can be computed from the notes in the staves (only for staves whith Staff.InChordGrid (page 270) active) or u defined.
When you change a value in a chord, the chord is automaticaly switched to user mode.
Accompaniements can follow the chord grid while playing the score.

The Application object provide some useful methods to convert chord name to chord components and chord components to chord name.

·Application.ConvertChordNameToElements(...) (page 183)
·Application.ConvertElementsToChordName(...) (page 183)

Remark: All management operations can be applied even if the chord grid is not visible.

4.3.20.1. Chord values

4.3.20.1.1. General

Chord.Name

Type : String
Description : Chord name. Setting a chord name will recompute all the chord values, including the chord name itself. The chord is switched in user mode (it will not be recomputed if the score change). If the chord is not already defined in the bar it is added.
Inversion, octave are not affected.
Example :

```
Include "MSDefine"
local i
local chordBar
local chord
local numberOfChord
score=FrontScore()
score.IsChordGridDisplayed=true
for i=1,score.NumberOfBars do
   chordBar,numberOfChord=score.GetBarChords(i)
   for j=1,numberOfChord do
      print (chordBar[j].Name)
   end
end
```

Chord.Diagram

Type : Diagram (page 363)
Description : The chord diagram related to the chord.
How Harmony Assistant choose what chord diagram to display ?
From the chord name in the chord grid (computed or user defined) a search is done first in the chord diagrams database related to the score grid then in the global chord diagrams database related to the application.

A script can't modify the global chord database, but can duplicate a global chord diagram as a local copy, related to the score grid.
In the score grid database, if several diagrams have the same name, the first chord diagram is taken.
In the global database, the "prefered" chord diagram is taken.
Deleting such an object can make the "Diagram" variables that refer to another Diagram further in the list change the element they refer.

Related methods and values:
Score.AddChordDiagram(...) (page 225), Score.GetChordDiagramFromName(...) (page 225), Score.GetChordDiagramFromIndex(...) (page 225),Score.DeleteChordDiagrams(...) (page 225), Score.DuplicateChordDiagrams(...) (page 225),Score.NumberOfChordDiagrams (page 201)

Chord.RootNote

Type : Number
Description : Root note : C=0,D=1, etc

Chord.Accidental

Type : Number
Description : Accidental as definied in MSDefine.

Chord.Third

Type : Number
Description : As defined in MSDefine :
SEMITONE_OFFSET_NONE=2
SEMITONE_OFFSET_DIMINISHED=-1
SEMITONE_OFFSET_MAJOR=0
SEMITONE_OFFSET_AUGMENTED=1

Chord.Fifth

Type : Number
Description : As defined in MSDefine :
SEMITONE_OFFSET_NONE=2
SEMITONE_OFFSET_DIMINISHED=-1
SEMITONE_OFFSET_MAJOR=0
SEMITONE_OFFSET_AUGMENTED=1

Chord.Seventh

Type : Number
Description : As defined in MSDefine :
SEMITONE_OFFSET_DOUBLE_DIMINISHED=-2
SEMITONE_OFFSET_DIMINISHED=-1
SEMITONE_OFFSET_MAJOR=0
SEMITONE_OFFSET_NONE=2
Example :

```
Include "MSDefine"
-- This script will toggle the 7th from a chord
score=FrontScore()
-- Extraction the chord in bar1, position 1
chordBar=score.GetBarChords(1)
print("The chord name is : "..chordBar[1].Name)
--Creating a new chord toggling the 7th and setting it
-- at the same position
if chordBar[1].Seventh==SEMITONE_OFFSET_NONE then
    chordBar[1].Name=Application.ConvertElementsToChordName(-1,chordBar[1].RootNote,chordBar[1].
    Accidental,chordBar[1].Third,chordBar[1].Fifth,SEMITONE_OFFSET_DIMINISHED,chordBar[1].Ninth)
else
    chordBar[1].Name=Application.ConvertElementsToChordName(-1,chordBar[1].RootNote,chordBar[1].
    Accidental,chordBar[1].Third,chordBar[1].Fifth,SEMITONE_OFFSET_NONE,chordBar[1].Ninth)

end
```

Chord.Ninth

Type : Number
Description : As defined in MSDefine :
SEMITONE_OFFSET_NONE=2
SEMITONE_OFFSET_DIMINISHED=-1
SEMITONE_OFFSET_MAJOR=0
SEMITONE_OFFSET_AUGMENTED=1

Chord.User

Type : Boolean
Description : true if the chord is defined by the user : it will not be computed from the score. False otherwise.

Chord.Octave

Type : Number
Description : Chors base octave.

Chord.Inversion

Type : Number
Description : Inversion of the chord as defined in MSDefine :
INVERSION_FUNDAMENTAL=0
INVERSION_1ST=1
INVERSION_2ND=2
INVERSION_3RD=3
INVERSION_4TH=4
INVERSION_1ST_BOTTOM=5
INVERSION_2ND_BOTTOM=6
INVERSION_3RD_BOTTOM=7
INVERSION_4TH_BOTTOM=8

4.3.20.1.2. Position

Chord.BarNumber

Type : Number
Description : Provide the bar position of the chord.
Example :

```
---Displaying bar position, position in bar, time position of each chord of the chord grid
score=FrontScore()
score.IsChordGridDisplayed=true
for i=1,score.NumberOfBars do
   chordBar,numberOfChord=score.GetBarChords(i)
   for j=1,numberOfChord do
      print ("Chord : '"..chordBar[j].Name.."' Bar # ".. chordBar[j].BarNumber.." Position in bar : "..chordBar[j].
      PositionInBar.." Time : "..chordBar[j].TimePosition)
   end
end
```

Chord.PositionInBar

Type : Number
Description : Provide the position in the bar of the chord from 1 to 16.

Chord.TimePosition

Type : Number
Description : Time position of the beginning of the chord.

4.3.20.2. Chord methods

4.3.20.2.1. Editing

Chord.Edit(...)

Description : Open the standard dialog box and prompt the user to edit the chord.
Parameters : None
Return value : Boolean: true if the chord was successfuly edited.
Example :

```
-- This script will ask the user to edit the first chord
score=FrontScore()
chordBar,numberOfChord=score.GetBarChords(1)
chordBar[1].Edit()
```

4.3.20.2.2. Managing

Chord.Split(...)

Description : Split the chord in two. The created chord is a copy of the source chord.
Parameters : None
Return value : Boolean: true if the chord was successfuly splitted.
Example :

```
-- This script will add a chord in all bar
Include "MSDefine"
local i
local chordBar
local chord
local numberOfChord
score=FrontScore()
score.IsChordGridDisplayed=true
for i=1,score.NumberOfBars do
   chordBar,numberOfChord=score.GetBarChords(i)
   chordBar[1].Split()
end
```

Chord.Delete(...)

Description : Delete the chord
Parameters : None
Return value : Boolean : true if the chord was successfully deleted.
Example :

```
--- Delete the first chord of the bar #1
score=FrontScore()
score.Preserve()
score.IsChordGridDisplayed=true
chordBar=score.GetBarChords(1)
chordBar[1].Delete()
```

Chord.CopyTo(...)

Description : Copy the chord on another chord. You can copy chord even from a score to another. If the destination chord don't exist in the destination bar it is created. The destination chord is switched to user mode. You can also use this method to copy all chords present in a bar to another bar.
Parameters : Chord : the destination chord.
Return value : Boolean : true if the chord was successfully copied.
Example :

```
--- Copy the first  chord of the bar #3 on the first division
--- of the bar #1
score=FrontScore()
score.Preserve()
--- Showing the chord grid
score.IsChordGridDisplayed=true
chordBarDest=score.GetBarChords(1)
chordBarSource=score.GetBarChords(3)
chordBarSource[1].CopyTo(chordBarDest[1])
```

```
--- Copy all the chords of bar 3 to bar 1
score=FrontScore()
score.Preserve()
--- Showing the chord grid
score.IsChordGridDisplayed=true
chordBarDest=score.GetBarChords(1)
chordBarSource=score.GetBarChords(3)
chordBarSource.CopyTo(chordBarDest)
```

4.3.21. FreeObject Object Type

General description of this object type : A free object. A free object is a graphical object (frame, text, or even a score) related to the score and not to a staff.
So, when the page setting change, the free object keep in place.
Related methods :
To get the first free object of score, use Score.FirstFreeObject (page 202)
To get the last free object of score, use Score.LastFreeObject (page 202)
To get the number of free object in a score, use Score.NumberOfFreeObject (page 202)
To add a free object to a score use Score.NewFreeObject(...) (page 234)

4.3.21.1. FreeObject values

4.3.21.1.1. Browsing FreeObjects

FreeObject.Next

Type : FreeObject
Description : Next FreeObject
If there is no next FreeObject, value is nil.

FreeObject.Previous

Type : FreeObject
Description : Previous FreeObject
If there is no previous FreeObject, value is nil.

4.3.21.1.2. General values

These values can apply to any kind of FreeObject object.

FreeObject.Type

Type : Number
Description : FreeObject type. Can be any of the constants defined in MSDefine.mys, i.e. :
- FREE_OBJECT_TEXT_TYPE
- FREE_OBJECT_PICTURE_TYPE
- FREE_OBJECT_SCORE_TYPE
- FREE_OBJECT_FRAME_TYPE
- FREE_OBJECT_FILLED_FRAME_TYPE
- FREE_OBJECT_CIRCLE_TYPE
- FREE_OBJECT_FILLED_CIRCLE_TYPE
- FREE_OBJECT_ROUND_FRAME_TYPE
- FREE_OBJECT_FILLED_ROUND_FRAME_TYPE
- FREE_OBJECT_HORIZONTAL_LINE_TYPE
- FREE_OBJECT_VERTICAL_LINE_TYPE
- FREE_OBJECT_LINE_DOWN_TYPE
- FREE_OBJECT_LINE_UP_TYPE
- FREE_OBJECT_CURVED_LINE_UP_TYPE
- FREE_OBJECT_CURVED_LINE_DOWN_TYPE
- FREE_OBJECT_CURVED_LINE_LEFT_TYPE
- FREE_OBJECT_CURVED_LINE_RIGHT_TYPE
- FREE_OBJECT_BRACE_TYPE

Note : Be careful: setting this value will reset the data related to the object.
For example a text object will have an empty string, a picture or score object, no file related, etc.

FreeObject.DontPrint

Type : Boolean
Description : If true, the object is not printed.

FreeObject.Locked

Type : Boolean
Description : If true, the object can't be moved.

4.3.21.1.3. Related files

FreeObject.State

Type : Number
Description : When a file can be related to a FreeObject, describe the state of the related file.
- FREE_OBJECT_NO_FILE_RELATED
- FREE_OBJECT_FILE_UP_TO_DATE
- FREE_OBJECT_FILE_NOT_UP_TO_DATE
- FREE_OBJECT_FILE_LINK_BROKEN

4.3.21.1.4. Position

FreeObject.Origin

Type : Number
Description : The page origin. The page number field (FreeObject.PageNumber (below)) will be counted from this origin.
Don't confuse with FreeObject.OriginInPage (below) which provide the location of the object inside the page.
- FREE_OBJECT_FROM_START
- FREE_OBJECT_FROM_END
- FREE_OBJECT_ALL_PAGES
- FREE_OBJECT_EVEN_PAGES
- FREE_OBJECT_ODD_PAGES
- FREE_OBJECT_AND_NEXT
- FREE_OBJECT_AND_PREVIOUS
- FREE_OBJECT_AFTER_LAST

FreeObject.PageNumber

Type : Number
Description : Page number (From the origin described by origin value). From 1 to n.

FreeObject.OriginInPage

Type : Number
Description : Origin in the page.
- FREE_OBJECT_FROM_TOP_OF_PAGE
- FREE_OBJECT_FROM_BOTTOM_OF_PAGE
- FREE_OBJECT_FROM_RIGHT_OF_PAGE
- FREE_OBJECT_FROM_BOTTOM_RIGHT_OF_PAGE

FreeObject.Left

Type : Number
Description : Left position of the object area in the page (in 1/10th of mm) following the FreeObject.OriginInPage (above) value.

FreeObject.Right

Type : Number
Description : Right position of the object area in the page (in 1/10th of mm) following the FreeObject.OriginInPage (above) value.

FreeObject.Top

Type : Number
Description : Top position of the object area in the page (in 1/10th of mm) following the FreeObject.OriginInPage (above) value.

FreeObject.Bottom

Type : Number
Description : Bottom position of the object area in the page (in 1/10th of mm) following the FreeObject.OriginInPage (above) value.

FreeObject.IsForeground

Type : Boolean
Description : True if the object is on the foreground plane, false if the object is on the background plane.

4.3.21.1.5. Selecting

FreeObject.IsSelected

Type : Boolean
Description : True if the object is selected

4.3.21.1.6. Specific data

FreeObject.Text

Type : Styled string (page 177)
Description : Text related to the FreeObject. The object type must be FREE_OBJECT_TEXT_TYPE. Setting a text to an object with another type switch the type to FREE_OBJECT_TEXT_TYPE.
Trying to get the text of an object of another type will provide an empty string.

FreeObject.TextStyle

Type : TextStyle
Description : Text style for FreeObject of type FREE_OBJECT_TEXT_TYPE.

FreeObject.NoWrap

Type : Boolean
Description : For type FREE_OBJECT_TEXT_TYPE : the text will not be wrapped to the right of the area. For FREE_OBJECT_GRAPHIC_TYPE, the graphic fill the frame.

FreeObject.IsEraseBackground

Type : Boolean
Description : For type FREE_OBJECT_TEXT_TYPE : the area background will be erased.

FreeObject.Rotation

Type : Number
Description : For type FREE_OBJECT_TEXT_TYPE or FREE_OBJECT_GRAPHIC_TYPE rotation in degree

FreeObject.Justify

Type : Number
Description : Justification mode for FreeObject of type FREE_OBJECT_TEXT_TYPE.

FreeObject.EmbeddedScorePage

Type : Number
Description : Displayed page for FreeObject of type FREE_OBJECT_SCORE_TYPE. From 1 to n

FreeObject.EmbeddedScoreView

Type : Number
Description : Displayed view for FreeObject of type FREE_OBJECT_SCORE_TYPE. From 1 (general view) to n

FreeObject.Thickness

Type : Number
Description : Thickness of the line for FreeObject of type FREE_OBJECT_FRAME_TYPE to FREE_OBJECT_CURVED_LINE_RIGHT_TYPE. In 1/10th of mm. From 1 to 100.

FreeObject.Color

Type : Number
Description : Color of the line for FreeObject of type FREE_OBJECT_FRAME_TYPE to FREE_OBJECT_CURVED_LINE_RIGHT_TYPE.

FreeObject.RelatedFileName

Type : String
Description : Full path and name of the file related to the object. For FREE_OBJECT_SCORE_TYPE type and FREE_OBJECT_PICTURE_TYPE

FreeObject.ViewIndex

Type : Number
Description : View(s) in which the text will be visible.
Can be either the view index as provided by View.Index (page 401) if the text is to be displayed on a single view the following constants:
·-3 : displayed on every view except the general view
·-2 : displayed on every view that contains the object's staff
·-1 : displayed only on general view
· 0 : displayed in each and every view
·(Default: -2)

4.3.21.1.7. Display

FreeObject.GraphRecord

Type : Graph
Description : Current recorded graphical context (created with FreeObject.BeginRecord(...)).

4.3.21.1.8. Tags

FreeObject.Tag

Type : String
Description : Tag related to this free object. Tags are saved in the file.

4.3.21.2. FreeObject methods

4.3.21.2.1. Deleting

FreeObject.Delete(...)

Description : Deletes the FreeObject object
Parameters : None
Return value : None
Example :

```
-- Deletes all FreeObjects in score
myScore=FrontScore()
if myScore==nil then
   print("Please load a score first")
else
   --We preserve the current score state,
   --so the user will be able to undo the changes.
   myScore.Preserve()
   myFreeObject=myScore.LastFreeObject
   while myFreeObject~=nil do
      -- Once object has been deleted, Previous member could not be retrieved
      previousFreeObject=myFreeObject.Previous
      myFreeObject.Delete()
      myFreeObject=previousFreeObject
   end
end
```

4.3.21.2.2. Managing Pictures

FreeObject.BeginRecord(...)

Description : Begin the recording of a picture in a free object. All following graph methods will be recorded in this picture.
Only one recording can be active at the same time. All recording must be close using the FreeObject.EndRecord(...) (below) method.
If you want to get the current graphic context where the graphic commands are recorded, you can use FreeObject.GraphRecord.
This is particulary useful if you want to call a function in a floating window using dialog.functionName because this kind of call set the current graphic context to the dialog graphic context.
In this case you just have to call the function with FreeObject.GraphRecord as parameter and use the Graph object.
Parameters : Optional :

- Number : horizontal size of the picture, in pixel (by default 100)
- Number : vertical size of the picture, in pixel (by default 100)

Return value : Boolean : true if the recording was inited, false if it failed (for example a record picture is pending)

Example :

```
Include "MSDefine"
score=FrontScore()
object=score.NewFreeObject(FREE_OBJECT_PICTURE_TYPE)
object.BeginRecord(200,200)
Graph.PenSize(1,1)
Graph.ForeColor(15)
Graph.PaintOval(20,20,120,120)
object.EndRecord()
Application.UpdateScores()
```

FreeObject.EndRecord(...)

Description : End the recording of a picture and add the resulting picture to the free object. The free object type is automatically changed to FREE_OBJECT_PICTURE_TYPE.
Parameters : None
Return value : None

4.3.22. StorageArea Object Type

General description of this object type : A Storage area is a script-dedicated memory space in which a script can store and retrieve custom values related to a score or a staff.
A Storage area can contain an undefined number of items, each item in a storage area having a unique tag/ID combination.
A tag is a 4-character string that defines the type of data, and the ID is a unique number for this tag.
An item contains a character string. The script is responsible for parsing this string according to its needs.
Such a storage area is attached to the Score object: Score.StorageArea (page 202), as well as every staff in the score: Staff.StorageArea (page 267)
These areas are written on disk when document file is saved, and read from disk when the document is reloaded.
There is also a global storage area, Application.StorageArea (page 181), reset each time the program is restarted, which enables scripts to share custom data with each other. This area is not saved.
Warning: Changing the content of a storage area in a staff or in a score does not mark the document as modified.

4.3.22.1. StorageArea methods

4.3.22.1.1. Managing

StorageArea.CountItems(...)

Description : Counts the number of items into a storage area, or of a given tag in a storage area
If a tag is provided, counts the number of items in this tag.
If no parameter is provided, counts the total number of items in the storage area.
Parameters : <optional> Tag (4-characters string)
Return value : Number of items
Example :

```
score=FrontScore()
if(score~=nil) then
   nb=score.StorageArea.CountItems()
   print("There are "..nb.." items in the score storage area");
   nb=score.StorageArea.CountItems("ABCD")
   print("There are "..nb.." items of type ABCD in the score storage area");
end
```

StorageArea.GetIndexedItemInfo(...)

Description : Gets information about an item, from its index (NOT its ID), either in the complete Storage area, or in a specific tag of this storage area.
There are two ways for using this method:
If a tag and an index are provided, returns the ID of this item (the Nth item of the tag)
If only an index is provided, returns the tag and the ID of this item (the Nth item in the whole storage area)
Parameters : either :
(optional) Tag (4-characters string)
index (Number)
Return value : If tag is provided, returns an ID
If only an index is provided, returns a tag and an ID
Example :

```
score=FrontScore()
if(score~=nil) then
   nb=score.StorageArea.CountItems()
   print("Enumerating storage area")
   for i=1,nb do
      tag,ID=score.StorageArea.GetIndexedItemInfo(i)
      print("Tag: "..tag..", ID: "..id)
   end
end
```

StorageArea.StoreItem(...)

Description : Stores a string into a Storage area.
If no parameter is provided, deletes the item in the storage area.
Parameters : Tag (4-characters string)
ID (Integer)
Optional: Value (String)
Return value : None
Example :

```
score=FrontScore()
if(score~=nil) then
   score.StorageArea.StoreItem("ABCD",123456,"Hello, world")
   print("Element has been stored.")
   str=score.StorageArea.RetrieveItem("ABCD",123456)
   if str then print("Element content is: "..str)
   else print("No content") end
   score.StorageArea.StoreItem("ABCD",123456)
   print("Element has been deleted.")
   str=score.StorageArea.RetrieveItem("ABCD",123456)
   if str then print("Element content is: "..str)
   else print("No content") end
end
```

StorageArea.RetrieveItem(...)

Description : Retrieves a string from a Storage area.
Parameters : Tag (4-characters string)
ID (Integer)
Return value : item value (String)

4.3.23. Target Object Type

General description of this object type : Targets that can be included in a score.
A target is simply a text objet related to the first staff, including the $! tag. When a target is inserted the text content is automaticaly generated.
Related values :
Score.AutomaticTargetCaptions (page 220)
Score.TargetCaptionsMode (page 220)
Score.FrameTargetCaptions (page 220)
Score.TargetCaptionsHorizontalShift (page 220)
Score.TargetCaptionsVerticalShift (page 220)
Score.FirstTarget (page 220)
Score.LastTarget (page 220)
Score.TargetCaptionsTextStyle (page 220)
Related methods :
Score.InsertTarget(...) (page 247)
Score.RegenerateTargets(...) (page 248)

4.3.23.1. Target values

4.3.23.1.1. Browsing targets

Target.Next

Type : Target
Description : Next Target
If there is no next Target, value is nil.

Target.Previous

Type : Target
Description : Previous Target
If there is no previous Target, value is nil.

4.3.23.1.2. Target miscellaneous values

These values can apply to any kind of Target object.

Target.Bar

Type : Number
Description : Bar at which the Target object is located
Example :

```
Include "MSDefine"
-- Displays bar number for each Target
myScore=FrontScore()
if myScore==nil then
   print("Please load a score first")
else
   index=1
   myTarget=myScore.FirstTarget
   while myTarget~=nil do
      print("myTarget #",index,"is located at bar",myTarget.Bar);
      index=index+1
      myTarget=myTarget.Next
   end
end
```

Target.Name

Type : String
Description : Name of this target. This can be different of the displayed caption along the Score.AutomaticTargetCaptions (page 220) mode.

Target.DisplayCaption

Type : Boolean
Description : Specify if the target is displayed

4.3.23.2. Target methods

4.3.23.2.1. Deleting a Target

Target.Delete(...)

Description : Deletes the Target object
Parameters : None
Return value : None
Example :

```
-- Deletes all Targets in score
myScore=FrontScore()
if myScore==nil then
    print("Please load a score first")
else
    --We preserve the current score state,
    --so the user will be able to undo the changes.
    myScore.Preserve()
    myTarget=myScore.LastTarget
    while myTarget~=nil do
        -- Once object has been deleted, Previous member could not be retrieved
        previousTarget=myTarget.Previous
        myTarget.Delete()
        myTarget=previousTarget
    end
end
```

4.3.23.2.2. Managing

Target.CopyTo(...)

Description : Copy all the Target setting in another Target.
Parameters : The target Target
Return value : None

4.3.23.2.3. Play music

Target.PlayMusic(...)

Description : Play the score starting at the target.
Parameters : None
Return value : None

4.4. Display

This section describe all the settings about the page mode display and the score printing.

4.4.1. View Object Type

General description of this object type : Display view for a score. Each view is a partial reprentation of the score, in which staves can be omitted (invisible), printing settings modified, etc.

To get the first view in the score, use Score.FirstView (page 221)
Note: this first view is ALWAYS the general view
To get the last view in the score, use Score.LastView (page 221)
To get the number of views in a score, use Score.NumberOfViews (page 220)

To create a new view, use Score.NewView(...) (page 249)
To get the current view, use Score.GetCurrentView(...) (page 249)
To set the current view, use Score.SetCurrentView(...) (page 249)

4.4.1.1. View values

4.4.1.1.1. Browsing Views

View.Next

Type : View
Description : Next View
If there is no next view, value is nil.
Example :

```
-- Display view names in ascending order
myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   view=myScore.FirstView
   while view do
      print ("View :"..view.Name)
      view=view.Next
   end
end
```

View.Previous

Type : View
Description : Previous View
If there is no previous View, value is nil.
Note: the first View is the general one.
Example :

```
-- Display view names in descending order
myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   view=myScore.LastView
   while view do
      print ("View :"..view.Name)
      view=view.Previous
   end
end
```

View.Index (R)

Type : Number
Description : Index of the view in the score "Views" array

4.4.1.1.2. View settings

Changing these values will need the view to be recalculated

View.Name

Type : String
Description : Name of the view. Note that the name of the general view can't be modified.

View.DisplayTempo

Type : Boolean
Description : true if tempo marks are always displayed on the first staff of this view, false if they don't appear when belonging to a hidden staff for this view.

View.DisplayDynamics

Type : Boolean
Description : true if global dynamic marks are always displayed on the first staff of this view, false if they don't appear when belonging to a hidden staff for this view.

View.DisplayText

Type : Boolean
Description : true if global text objects are always displayed on the first staff of this view, false if they don't appear when belonging to a hidden staff for this view.

View.SpecificPageSetup

Type : Boolean
Description : true if page setup is specific to this view, false if it uses the general settings

View.SpecificEngraver

Type : Boolean
Description : true if engraver setup is specific to this view, false if it uses the general settings

View.SpecificScoreAppearance

Type : Boolean
Description : true if score appearance setup is specific to this view, false if it uses the general settings

View.SpecificFinePosition

Type : Boolean
Description : true if fine position setting is specific to this view, false if it uses the general settings

View.SpecificStaffAppearance

Type : Boolean
Description : true if staff appearance settings are specific to this view, false if the general settings are used.

View.SpecificConcertMode

Type : Boolean
Description : true if concert mode setting (display transposed/untransposed score) is specific to this view, false if the general setting is used.

View.SpecificPartGroup

Type : Boolean
Description : true if part group settings (bracket/braces in front of staves) are specific to this view, false if the general setting is used.

4.4.1.2. View methods

4.4.1.2.1. Deleting/Sorting views

View.Delete(...)

Description : Delete the view
Parameters : None
Return value : None
Example :

```
-- Deleting the last view
myScore=FrontScore()
if myScore==nil then
    print("You must open a score first")
elseif myScore.NumberOfViews<2 then
    print("There is only one view in this score")
else
    print("Deleting view: "..myScore.LastView.Name)
    myScore.LastView.Delete()
end
```

View.Move(...)

Description : Move a view within the view array of a score
Parameters : Index to move the view to
Return value : None
Example :

```
-- Moving view 2 to view 3
myScore=FrontScore()
if myScore==nil then
    print("You must open a score first")
elseif myScore.NumberOfViews<3 then
    print("You must have at least 3 views for this operation")
else
    score.Views[2].Move(3);
    print("View "..score.Views[3].Name.." moved at 3rd position)
end
```

4.4.1.2.2. View settings

View.CopyTo(...)

Description : Copy view parameters to another view (except Name)
Parameters : View to copy parameters to
Return value : None
Example :

```
-- Copying view 2 to view 3
myScore=FrontScore()
if myScore==nil then
    print("You must open a score first")
elseif myScore.NumberOfViews<3 then
    print("You must have at least 3 views for this operation")
else
    score.Views[2].CopyTo(score.Views[3]);
    print("View "..score.Views[2].Name.." copied to View "..score.Views[3].Name)
end
```

4.4.2. EngraverSetup Object Type

General description of this object type : A EngraverSetup describe the way the different symbols are spaced on score Typically a EngraverSetup object is related to a Score object.

4.4.2.1. EngraverSetup global functions

4.4.2.1.1. Managing

CopyEngraverSetup(...)

Description : Copy the entire content of a EngraverSetup object into another EngraverSetup object.
Parameters : The source EngraverSetup, the destination EngraverSetup.
Return value : None

DefaultEngraverSetup(...)

Description : Fill the EngraverSetup object with the default values.
Parameters : The EngraverSetup to fill.
Return value : None

NewEngraverSetup(...)

Description : Allocate a new EngraverSetup object in memory for special purpose. This EngraverSetup object is not related to a Score object and **must** be released by the script using the DisposeEngraverSetup function.
Parameters : None
Return value : A new EngraverSetup object nil if the function failled, for example if there is not enough memory.

DisposeEngraverSetup(...)

Description : Dispose a EngraverSetup object previously allocated by NewEngraverSetup. Never dispose a EngraverSetup object embedded in a Score object. Never dispose twice a EngraverSetup object.
Parameters : A EngraverSetup object.
Return value : None

4.4.2.1.2. Editing

EditEngraverSetup(...)

Description : Ask the user to edit a EngraverSetup object using the default application dialog box. The dialog box content is displayed according to the user's language.
Parameters : The EngraverSetup to edit.
Return value : true if the user validate his choice, false if the user canceled the dialog box. In this last case, values are not changed.
Example :

```
MyEngraverSetup=NewEngraverSetup()
if(MyEngraverSetup ~=nil) then
   DefaultEngraverSetup(MyEngraverSetup)
   EditEngraverSetup(MyEngraverSetup)
   DisposeEngraverSetup(MyEngraverSetup)
end
```

4.4.2.2. EngraverSetup values

4.4.2.2.1. Bars

EngraverSetup.IsComputeBarScale

Type : Boolean
Description : Define if the bar scale will be automatically computed.

EngraverSetup.SmallestBar

Type : Number
Description : Number of units for smallest bar. Only used if EngraverSetup.IsComputeBarScale (page 403) is tru

EngraverSetup.BiggestBar

Type : Number
Description : Number of units for biggest bar. Only used if EngraverSetup.IsComputeBarScale (page 403) is true

EngraverSetup.EmptyBar

Type : Number
Description : Number of units for empty bar. Only used if EngraverSetup.IsComputeBarScale (page 403) is true.

4.4.2.2.2. General

EngraverSetup.IsActive

Type : Boolean
Description : Define of the engraver mode is enabled or not.

4.4.2.2.3. Lyrics

EngraverSetup.Lyrics

Type : Number
Description : Number of units for lyrics.

4.4.2.2.4. Notes

EngraverSetup.Whole

Type : Number
Description : Number of units for a whole note or longer note.

EngraverSetup.Half

Type : Number
Description : Number of units for a half note.

EngraverSetup.Quarter

Type : Number
Description : Number of units for a quarter note.

EngraverSetup.Height

Type : Number
Description : Number of units for a height note.

EngraverSetup.Sixteenth

Type : Number
Description : Number of units for a sixteenth note

EngraverSetup.ThirtySecond

Type : Number
Description : Number of units for a 32nd note

EngraverSetup.SixtyFourth

Type : Number
Description : Number of units for a 64th note or shorter.

EngraverSetup.Unbeam

Type : Number
Description : Number of units for unbeam notes. When notes of short duration are not beamed together, an additional distance can be added to separate them.

EngraverSetup.Before

Type : Number
Description : Number of units before notes. When notes of short duration are not beamed together, an additional distance can be added to separate them.

EngraverSetup.Grace

Type : Number
Description : Number of units for grace notes. Grace notes (appoggiatura or acciacatura) also add an additional distance.

EngraverSetup.Accidental

Type : Number
Description : Number of units for accidental. When an accidental is displayed, an additional distance is also added.

4.4.3. PageSetup Object Type

General description of this object type : A PageSetup describes the document appearance. Typically a PageSetup object is related to a Score object. For example, a PageSetup object will define the margins, page headers, scale, the way the staffs are justified, how the multi-rests bars will be printed, etc. A Score object includes several independent PageSetup objects :

- PrintSetup to print the score. As the Page Mode display the score as it will be printed, the Printing Setup affect the page mode display.
- ExportSetup to export the score graphicaly
- StreetOrganSetup to print the street organs cards
- ChordGridSetup to print the chord grid

Related topic: Local metrics of the staff object which allow you to change the positions individually inside a system.

4.4.3.1. PageSetup global functions

4.4.3.1.1. Managing

CopyPageSetup(...)

Description : Copy the entire content of a PageSetup object into another PageSetup object.
Parameters : The source PageSetup, the destination PageSetup.
Return value : None

DefaultPageSetup(...)

Description : Fill the PageSetup object with the default values.
Parameters : The PageSetup to fill.
Return value : None

NewPageSetup(...)

Description : Allocate a new PageSetup object in memory for special purpose. This PageSetup object is not related to a Score object and **must** be released by the script using the DisposePageSetup function.
Parameters : None
Return value : A new PageSetup object nil if the function failled, for example if there is not enough memory.

DisposePageSetup(...)

Description : Dispose a PageSetup object previously allocated by NewPageSetup. Never dispose a PageSetup object embedded in a Score object. Never dispose twice a PageSetup object.
Parameters : A PageSetup object.
Return value : None

4.4.3.1.2. Editing

EditPageSetup(...)

Description : Ask the user to edit a PageSetup object using the default application dialog box. The dialog box content is displayed according to the user's language.
Parameters : The PageSetup to edit.
Return value : true if the user validated his choice, false if the user canceled the dialog box. In this last case, values are not changed.
Example :

```
MyPageSetup=NewPageSetup()
if(MyPageSetup ~=nil) then
   DefaultPageSetup(MyPageSetup)
   EditPageSetup(MyPageSetup)
   DisposePageSetup(MyPageSetup)
end
```

4.4.3.2. PageSetup values

4.4.3.2.1. General

PageSetup.Scale

Type : Number
Description : This is the display scale in percent. The scale must be greater than zero.

PageSetup.ReservedPages

Type : Short
Description : Number of reserved pages at the begin of the score.

PageSetup.PaperOrientation

Type : Number
Description : Paper orientation. 0 is default, 1 is portrait, 2 is landscape

PageSetup.BottomPageHeader

You can only write this value.
Type : PageHeader
Description : This is the PageHeader (page 410) object displated at the bottom of each page.

PageSetup.TopPageHeader

You can only write this value.
Type : PageHeader
Description : This is the PageHeader (page 410) object displated at the top of each page.

4.4.3.2.2. Margin

PageSetup.LeftMargin

Type : Number
Description : Left margin in 10th of millimeter.

PageSetup.RightMargin

Type : Number
Description : Right margin in 10th of millimeter.

PageSetup.TopMargin

Type : Number
Description : Top margin in 10th of millimeter.

PageSetup.BottomMargin

Type : Number
Description : Bottom margin in 10th of millimeter.

PageSetup.FirstGroupOffset

Type : Number
Description : Extra margin which will be used to shift the first staff group on the first page. In 10th of millimeter.

PageSetup.SwapLeftRightMargin

Type : Boolean
Description : If true, left and right margins are swapped on even pages.

4.4.3.2.3. Special

PageSetup.InfosOnFirstPage

Type : Boolean
Description : If true, Title, Composer, Remarks will be printed on the first page.

PageSetup.AddLyrics

Type : Boolean
Description : If true, lyrics will be printed at the end of the document on a separate page. A page will be added for each staff with lyrics.

PageSetup.AddChordDiagrams

Type : Boolean
Description : If true, chords diagrames used in the staff will be printed at the end of the document on a separate page.

PageSetup.AddStavesChordGrid

Type : Boolean
Description : If true, chord grid will be printed for each staff with a chord line at the end of the document on a separate page.

PageSetup.AddChordGrid

Type : Boolean
Description : If true, chord grid will be printed at the end of the document on a separate page.

4.4.3.2.4. Staves group

PageSetup.DistanceBetweenGroup

Type : Number
Description : Distance between two groups on the page in 10th of millimeter.

PageSetup.FirstBar

Type : Number
Description : First bar to be printed.

PageSetup.LastBar

Type : Number
Description : Last bar to be printed.

PageSetup.PrintRuler

Type : Boolean
Description : If true, the ruler will be printed before each staff group.

PageSetup.PrintChordGrid

Type : Boolean
Description : If true, the chord grid will be printed before each staff group.

PageSetup.PrintRythmsMatrix

Type : Boolean
Description : If true, the rythms matrix will be printed before each staff group.

PageSetup.PrintAccompanimentsMatrix

Type : Boolean
Description : If true, the accompaniments matrix will be printed before each staff group.

PageSetup.DontCutStaffGroup

Type : Boolean
Description : If true, when a staff group reach the end of a page, it will not be cut but printed on the next page instead.

PageSetup.JustifyBars

Type : Boolean
Description : If true, when right edge of the staves group will be aligned.

PageSetup.MultiRestBar

Type : Boolean
Description : If true, successive empty bars will be merged into one bar.

PageSetup.MultiRepeatBar

Type : Boolean
Description : If true, successive repeated bars (%) will be merged into one bar.

PageSetup.NumberOfMultiRestBar

Type : Number
Description : Number of succesive empty (or repeated) bars to merge. If the number of successive bars is lower than this value, bars will not be merged.

PageSetup.JumpAndTargetOnAllStaves

Type : Boolean
Description : If true, jump and target symbols will be displayed on all staves. If false, jump and target symbols will displayed only on the first staff of the group.

PageSetup.RepeatClef

Type : Boolean
Description : If true, Clefs will be displayed on all staves. If false, Clefs will displayed only on the first staff of the group.

PageSetup.ClefOnPreviousLine

Type : Boolean
Description : If true, a clef change at the beginning of a line will be repeated at the end of the previous line.

PageSetup.RemoveEmptyStaves

Type : Boolean
Description : If true, empty staves will not be printed.

PageSetup.EmptyStavesOnFirstGroup

Type : Boolean
Description : If true, empty staves will be printed on the first staff group.

PageSetup.JustifyGroup

Type : Boolean
Description : If true, groups will be justified vertically on the page : distance between group will be ajusted to fill the page.

PageSetup.JustifyGroupButNotLastPage

Type : Boolean
Description : If true, groups will not be justified on the last page.

PageSetup.RemoveEmptyGroups

Type : Boolean
Description : If true, empty groups will not be printed.

PageSetup.JustifyLastSystem

Type : Boolean
Description : If true, the last group of the document will be right justified.

4.4.4. PageHeader Object Type

General description of this object type : A page header is a set of values which will be printed at the top and at the bottom of each page of the document. The page header object is typically related to a PageSetup (page 406) object type There is two PageHeader by PageSetup object type : one on the top one on the bottom. (BottomPageHeader and TopPageHeader values of the PageSetup object type).
The strings displayed can include some tokens which will be substituted while printing. A token always start with the $ (dollar) character.
For example, $D token is replaced by the current date. You can use :

$F : file name
$T : document title
$D : current date
$H : current time
$P : page number
$N : number of pages
$$: $ character

4.4.4.1. PageHeader global functions

4.4.4.1.1. Managing

CopyPageHeader(...)

Description : Copy the entire content of a PageHeader object into another PageHeader object.
Parameters : The source PageHeader, the destination PageHeader.
Return value : None

DefaultPageHeader(...)

Description : Fill the PageHeader object with the default values. Because default values are different for page header and footer you can specify what kind of default value you want.
Parameters : The PageHeader to fill.
Optional : true if is the top object, false if it is the bottom object (true by default)
Return value : None

NewPageHeader(...)

Description : Allocate a new PageHeader object in memory for special purpose. This PageHeader object is not related to a PageStup object and **must** be released by the script using the DisposePageHeader function.
Parameters : None
Return value : A new PageHeader object nil if the function failled, for example if there is not enough memory.

DisposePageHeader(...)

Description : Dispose a PageHeader object previously allocated by NewPageHeader. Never dispose a PageHeader object embedded in a PageSetup object. Never dispose twice a PageHeader object.
Parameters : A PageHeader object.
Return value : None

4.4.4.1.2. Editing

EditPageHeader(...)

Description : As the user to define a top or a bottom PageHeader using the default application dialog box. The dialog box content is displayed according to the user's language.
Parameters : The PageHeader to edit.
Return value : true if the user validate its choice, false if the user cancel the dialog box. In this last case, values are not changed.
Example :

```
MyPageHeader=NewPageHeader()
if(MyPageHeader ~=nil) then
   DefaultPageHeader(MyPageHeader)
   EditPageHeader(MyPageHeader)
   DisposePageHeader(MyPageHeader)
end
```

4.4.4.2. PageHeader values

4.4.4.2.1. General

PageHeader.AlternateOddEven

Type : Boolean
Description : If on, left and right will be inverted on odd and even pages.

PageHeader.DontPrintOnFirstPage

Type : Boolean
Description : If on, nothing will be displayed on the first page of the document.

PageHeader.LineAbove

Type : Boolean
Description : If on a line will be draw above the page header.

PageHeader.LineBelow

Type : Boolean
Description : If on a line will be draw below the page header.

PageHeader.NotOnFirstPage

Type : Boolean
Description : The header/footer will not be displayed on the first page.

4.4.4.2.2. Left definition

PageHeader.LeftTextString

Type : String
Description : String displayed at the left of the page.

PageHeader.LeftTextStyle

Type : TextStyle
Description : Style of text displayed at the left of the page.

4.4.4.2.3. Margins

PageHeader.LeftMargin

Type : Number
Description : Left margin in 10th of millimeter.

PageHeader.RightMargin

Type : Number
Description : Right margin in 10th of millimeter.

PageHeader.VerticalMargin

Type : Number
Description : Vertical margin in 10th of millimeter.

4.4.4.2.4. Middle definition

PageHeader.MiddleTextString

Type : String
Description : String displayed at the middle of the page.

PageHeader.MiddleTextStyle

Type : TextStyle
Description : Style of text displayed at the middle of the page.

4.4.4.2.5. Right definition

PageHeader.RightTextString

Type : String
Description : String displayed at the right of the page.

PageHeader.RightTextStyle

Type : TextStyle
Description : Style of text displayed at the Right of the page.

4.4.5. Comment Object Type

General description of this object type : A Comment is a styled text related to a Score object. Title, Composer, Remark are stored in a Comment object. Comments are displayed in the Comment dialog box : Score. DisplayComments method or on the first printed page.

4.4.5.1. Comment values

4.4.5.1.1. Working with Comments

Comment.String

Type : String (styled)
Description : The string which will be displayed.

Comment.Style

Type : TextStyle (page 414) object
Description : The style (font, face, size, color) of the Comment

Comment.Justification

Type : Number
Description : Justification : 0 mean left justified, 1 centered, -1 right justified

Comment.HorizontalOffset

Type : Number
Description : Horizontal position on the first page, relatively to the margin, in 10th of millimeter.

Comment.VerticalOffset

Type : Number
Description : Vertical position on the first page, relatively to the margin, in 10th of millimeter.

Comment.Rotation

Type : Number
Description : Rotation in degree.

4.4.6. TextStyle Object Type

General description of this object type : This object type is used by many other objects, like for example, page footer. It define a text string related to
- a font (Times, Arial, Geneva...),
- a face (bold, italic, condensed, underlined),
- a color
- and a size (in pixel).

4.4.6.1. TextStyle global functions

4.4.6.1.1. Creating

NewTextStyle(...)

Description : Allocate a TextStyle object in memory. The TextStyle is not inited.
Use DefaultTextStyle to set the TextStyle to a default value.
You ought imperatively to dispose the TextStyle using the DisposeTextStyle function.
Parameters : Nothing
Return value : A TextStyle object

4.4.6.1.2. Asking the user

EditTextStyle(...)

Description : Ask the user to select a TextStyle using the standard dialog box
Parameters : The TextStyle to edit
Return value : True if the user validated his choice, False if the user canceled his choice

4.4.6.1.3. Working on TextStyle

CopyTextStyle(...)

Description : Copy a TextStyle to another TextStyle
Parameters : TextStyle source, TextTyle destination
Return value : Nothing

DefaultTextStyle(...)

Description : Fill the TextStyle object you provide with the default values : font Times, size 12, no face, color black.
Parameters : The TextStyle to fill
Return value : Nothing

4.4.6.1.4. Disposing TextStyle

DisposeTextStyle(...)

Description : Delete a TextStyle object previously allocated via NewTextStyle
Parameters : The TextStyle object to dispose.
Return value : Nothing
Note : Be aware to NEVER dispose a TextStyle you didn't allocated...

4.4.6.2. TextStyle values

4.4.6.2.1. Working on text style

TextStyle.Font

Type : string
Description : Get/set the font name recorded in a TextStyle object.

TextStyle.Face

Type : String
Description : Get/set the face (bold, italic, condensed...) recorded in a TextStyle object.
The string is constituted of characters (uppercase or lowercase) :
B is Bold
I is Italic
U is Underline
O is Outline
S is Shadow
C is Condense
E is Extend
N is No background

TextStyle.Size

Type : number
Description : Get/set the size in pixel recorded in a TextStyle object.

TextStyle.Color

Type : string
Description : Get/set the color recorded in a TextStyle object. The color is coded "rrggbb" where "rr" is the red value from 00 to FF, "gg" the green value, "bb" the blue value.

4.5. Music

This section describe all the document settings about to the music playing.
We recommend to take a look to some objects methods and values related to this topic :

- Application.CurrentPlayingScore (page 179) which provides the current playing score
- Score.PlayMusic(...) (page 237) which launches the music
- Score.StopMusic(...) (page 238) which stops the music
- Score.IsMetronomeActive (page 207) which activates or deactivates the metronome
- Score.IsPlayAccompanimentsActive (page 210) which activates or deactivates the accompaniments playing
- Score.IsPlayRythmsActive (page 210) which activates or deactivates the rythms playing
- Score.IsPlaySelectionActive (page 210) which activates or deactivates the playing of the selection
- Score.IsLoopPlayActive (page 210) which activates or deactivates the loop while music end

4.5.1. Rule Object Type

General description of this object type : Rules that can be applied to notes in a staff

4.5.1.1. Rule values

4.5.1.1.1. Browsing rules

Rule.Next

Type : Rule
Description : Next rule
If there is no next rule, value is nil.
Note :
a) Rules are not sorted. They are linked in the order they have been inserted.
b) You can browse rules through Next and Previous fields, but you can also access a rule at a given index through, for example, Staff.Rules (page 263)[index]

Rule.Previous

Type : Rule
Description : Previous rule
If there is no previous rule, value is nil.
Note :
a) Rules are not sorted. They are linked in the order they have been inserted.
b) You can browse rules through Next and Previous fields, but you can also access a rule at a given index through, for example, Staff.Rules (page 263)[index]

Rule.Index

Type : Number
Description : Index of this rule
Example :

```
-- Displays the index of the 2nd rule of a staff, if exists
rule=myStaff.Rules[2]
if rule~=nil then
   print(rule.Index) -- Displays "2"
end
```

4.5.1.1.2. Rule display settings

Rule.Title

Type : String
Description : Rule title (can be displayed on score)
Example :

```
-- Displays all rules title of a staff
print("Rule list for staff",myStaff.Title)
rule=myStaff.FirstRule
if rule==nil then print("No rule, sorry")
else
   while rule~=nil do
      print(rule.Title)
      rule=rule.next
   end
end
```

Rule.TitleDisplayMode

Type : Number
Description : Way of displaying rule title on score (see constants in MSDefine (page 100).mys)
RULE_DISPLAY_NEVER=0 -- never displayed
RULE_DISPLAY_ONCE=1 -- only once in the whole score
RULE_DISPLAY_EACH_SYSTEM=2 -- each system (line)
RULE_DISPLAY_ALWAYS=3 -- each time a note matches

4.5.1.1.3. Rule matching criteria

A rule applies to a not if this note matches a given set of criteria.
The following fields enable to get or set these matching criteria.

Rule.IsShapeCriterion

Type : Boolean
Description : true if note head shape is considered as a criterion, false if any shape matches.
Note : Note shape criterion value cannot be read if this value is false.

Rule.ShapeCriterion

Type : Number
Description : Head shape the note must have to get a chance to match the rule (see constants in MSDefine (pag
100).mys)
SHAPE_NORMAL=0
SHAPE_TRIANGLE=1
SHAPE_DIAMOND=2
SHAPE_CROSS=3
SHAPE_PLUS=4
SHAPE_SQUARE=5
SHAPE_BEAM=6
SHAPE_NONE=7
SHAPE_FLAG=8
SHAPE_HALF_CIRCLE=9
SHAPE_DROP=10
SHAPE_SAND_GLASS=11
SHAPE_ARC=12
SHAPE_EMPTY_DIAMOND=13
SHAPE_CONE=14
SHAPE_CRESCENT=15
SHAPE_LEFT_TRIANGLE=16
SHAPE_BELL=17
SHAPE_CONCAVE_TRIANGLE=18
SHAPE_CONVEX_TRIANGLE=19
SHAPE_JOHNSON_TRIANGLE=20
SHAPE_JOHNSON_SQUARE=21
SHAPE_CIRCLE_BOLD_CROSS=22
SHAPE_BOLD_CROSS=23
SHAPE_THIN_CROSS=24
SHAPE_HALF_CIRCLE_THIN_CROSS=25
Note : Can only be used if Rule.IsShapeCriterion (above) is true. Otherwise, this value is nil.
Setting this value automatically switches Rule.IsShapeCriterion (above) to true.

Rule.IsPitchCriterion

Type : Boolean
Description : true if note pitch is considered as a criterion, false if any note pitch matches.
Note : Note pich criterion value cannot be read if this value is false.

Rule.PitchCriterion

Type : String
Description : Pitch the note must have to get a chance to match the rule
Note : a) Can only be used if Rule.IsPitchCriterion (above) is true. Otherwise, this value is nil.
b) Setting this value automatically switches Rule.IsPitchCriterion (above) to true.
c) Note pitch comparison can be absolute or octave-independent, or even relative to the current key or the chord root note, depending on Rule.IsAnyOctave (below),Rule.IsKeyRelative (below) and Rule.IsRootRelative (pa 419)
d) This value is provided in the current notation system (Latin or Anglo-saxon), which depends on the user settings in Global setup.
e) The three members : PitchCriterion, AngloSaxonPitchCriterion and LatinPitchCriterion reflect the same internal value. Setting one of them alters the two others.

Rule.AngloSaxonPitchCriterion

Type : String
Description : Pitch the note must have to get a chance to match the rule
Note : a) Can only be used if Rule.IsPitchCriterion (above) is true. Otherwise, this value is nil.
b) Setting this value automatically switches Rule.IsPitchCriterion (above) to true.
c) Note pitch comparison can be absolute or octave-independent, or even relative to the current key or the chord root note, depending on Rule.IsAnyOctave (below),Rule.IsKeyRelative (below) and Rule.IsRootRelative (pa 419)
d) This value is provided in Anglo-saxon notation (A,B,C...).
e) The three members : PitchCriterion, AngloSaxonPitchCriterion and LatinPitchCriterion reflect the same internal value. Setting one of them alters the two others.

Rule.LatinPitchCriterion

Type : String
Description : Pitch the note must have to get a chance to match the rule
Note : a) Can only be used if Rule.IsPitchCriterion (above) is true. Otherwise, this value is nil.
b) Setting this value automatically switches Rule.IsPitchCriterion (above) to true.
c) Note pitch comparison can be absolute or octave-independent, or even relative to the current key or the chord root note, depending on Rule.IsAnyOctave (below),Rule.IsKeyRelative (below) and Rule.IsRootRelative (pa 419)
d) This value is provided in Latin notation (Do,Re,Mi...).
ed) The three members : PitchCriterion, AngloSaxonPitchCriterion and LatinPitchCriterion reflect the same internal value. Setting one of them alters the two others.

Rule.IsAnyOctave

Type : Boolean
Description : true if pitch comparison does not take the octave into account (if an "A3" pitch criterion is defined, all "A"s will match)
Note : Can only be used if Rule.IsPitchCriterion (above) is true. Otherwise, this value is nil, and it cannot be set

Rule.IsKeyRelative

Type : Boolean
Description : true if pitch comparison is relative to the current key. Pitch criterion is defined in the key of C major.

For example, if this value is true and you defined a pitch criterion of D, rule will match D notes in C major key, but also G notes in F major key.
Note : Can only be used if Rule.IsPitchCriterion (above) is true. Otherwise, this value is nil, and it cannot be set

Rule.IsRootRelative

Type : Boolean
Description : true if pitch comparison is relative to the root note of the current chord, if exists.
Root note is the one that matches the rule whose Rule.IsRootNote (page 420) field is true.
In such case, pitch offset between the note and the root note is compared to the pitch offset between the rule pitch criterion and C.
For example, if root note is an F, and rule pitch criterion is D, then a G note on the staff will match.
(one tone between C and D equals one between F and G).
Note : Can only be used if Rule.IsPitchCriterion (page 418) is true. Otherwise, this value is nil, and it cannot be s

Rule.IsDifferentiateEnharmonics

Type : Boolean
Description : true if only notes with the right accidental match, false if only the pitch value is compared.
For example, if true, a C# note won't match a Db rule.
Note : Can only be used if Rule.IsPitchCriterion (page 418) is true. Otherwise, this value is nil, and it cannot be s

Rule.IsColorCriterion

Type : Boolean
Description : true if note color is considered as a criterion, false if any color matches.
Note : Color criterion value cannot be read if this value is false.

Rule.ColorCriterion

Type : Number
Description : Color index the note must have to get a chance to match the rule
Note : Can only be read if Rule.IsColorCriterion (above) is true. Otherwise, this value is nil.
Setting this value automatically switches Rule.IsColorCriterion (above) to true.

Rule.IsVelocityCriterion

Type : Boolean
Description : true if note velocity is considered as a criterion, false if any velocity matches.
Note : velocity bounds cannot be read if this value is false.

Rule.MinVelocityCriterion

Type : Number
Description : Minimum velocity the note must have to get a chance to match the rule
Note : Can only be read if Rule.IsVelocityCriterion (above) is true. Otherwise, this value is nil.
Setting this value automatically switches Rule.IsVelocityCriterion (above) to true.

Rule.MaxVelocityCriterion

Type : Number
Description : Maximum velocity the note must have to get a chance to match the rule
Note : Can only be read if Rule.IsVelocityCriterion (above) is true. Otherwise, this value is nil.
Setting this value automatically switches Rule.IsVelocityCriterion (above) to true.

Rule.IsOrnamentCriterion

Type : Boolean
Description : true if a note ornament is considered as a criterion, false if ornament is not a criterion.
Note : Ornament criterion, cannot be read if this value is false.

Rule.OrnamentCriterion

Type : Number
Description : Ornament type the note must own to get a chance to match the rule (see constants in MSDefine (page 100).mys)
ORNAMENT_HAMMER_ON=1
ORNAMENT_PULLING_OFF=2
ORNAMENT_SLIDE=3
ORNAMENT_SLIDE_UP=4
ORNAMENT_SLIDE_DOWN=5
ORNAMENT_DADDI_PICKING=6
ORNAMENT_CHOKE=7
ORNAMENT_BEND=8
ORNAMENT_RELEASE=9
etc
Note : Can only be read if Rule.IsOrnamentCriterion (page 419) is true. Otherwise, this value is nil.
Setting this value automatically switches Rule.IsOrnamentCriterion (page 419) to true.

Rule.IsOrnamentCriterionNotPlayed

Type : Boolean
Description : true if the ornament that has been used as a criterion must not be played.
Note : This value can be read only if Rule.IsOrnamentCriterion (page 419) is true.

Rule.IsContinueProcessing

Type : Boolean
Description : true if next rules are processed even if the note matches, false if rule processing stops when note matches

4.5.1.1.4. Applied rule parameters

When a note matches a rule, these settings are applied to the note to change the way it is played.

Rule.IsDefaultInstrument

Type : Boolean
Description : true if the matching note plays using the default instrument of the staff, false if it plays using the Rule.AppliedInstrument (below) instrument.

Rule.AppliedInstrument

Type : Number
Description : instrument index in the instrument array (Score.Instruments (page 206)) the note will play.
Note : Only used if Rule.IsDefaultInstrument (above) is false.

Rule.IsRootNote

Type : Boolean
Description : true if this note is considered as the root note of the current chord.
In this case, pitch of the matching note will be used as the reference for pitch comparison in rules whose Rule.IsRootRelative (page 419) value is true.
Note : Can only be used if Rule.IsPitchCriterion (page 418) is true. Otherwise, this value is nil.

Rule.IsSeparateBuffer

Type : Boolean
Description : true if each rule generates its own pseudo-staff when playing, false if all digital data generated by all rules
when playing are merged together before the effect processors are applied

Rule.AppliedMinVelocity

Type : Number
Description : Velocity to set when the original note velocity is equal to Rule.MinVelocityCriterion (page 419)
Original velocities between Rule.MinVelocityCriterion (page 419) and Rule.MaxVelocityCriterion (page 419) will changed, using a linear transform, by a value between Rule.AppliedMinVelocity (above) and Rule.AppliedMaxVelocity (page 421)
Rule.AppliedMinVelocity can be greater than Rule.AppliedMaxVelocity.
Note : Only used if Rule.IsVelocityCriterion (page 419) is true.

Rule.AppliedMaxVelocity

Type : Number
Description : Velocity to set when the original note velocity is equal to Rule.MaxVelocityCriterion (page 419)
Original velocities between Rule.MinVelocityCriterion (page 419) and Rule.MaxVelocityCriterion (page 419) will changed, using a linear transform, by a value between Rule.AppliedMinVelocity (page 420) and Rule.AppliedMaxVelocity (above)
Rule.AppliedMinVelocity can be greater than Rule.AppliedMaxVelocity.
Note : Only used if Rule.IsVelocityCriterion (page 419) is true.

Rule.IsFixedVelocity

Type : Boolean
Description : true if a fixed velocity is applied to the note, false if note velocity remains unchanged
Note : Only available if Rule.IsVelocityCriterion (page 419) is false (nil otherwise)
Setting this value to true automatically switches Rule.IsVelocityCriterion (page 419) to false

Rule.AppliedFixedVelocity

Type : Number
Description : fixed velocity is applied to the note
Note : Only available if Rule.IsVelocityCriterion (page 419) is false AND Rule.IsFixedVelocity (above) is true (nil otherwise)
Setting this value automatically switches Rule.IsFixedVelocity (above) to true and Rule.IsVelocityCriterion (page to false

Rule.IsAccidentalApplied

Type : Boolean
Description : true if a preferred accidental has to be applied to the note, false if note preferred accidental remains unchanged

Rule.AppliedAccidental

Type : Number
Description : Preferred accidental to apply to the note (see constants in MSDefine (page 100).mys)
NATURAL=0
SHARP=1
AUTO=2
FLAT=3
DOUBLE_SHARP=4
DOUBLE_FLAT=5
Note : Can only be read if Rule.IsAccidentalApplied (above) is true
Setting this value automatically switches Rule.IsAccidentalApplied (above) to true

Rule.IsAccidentalShiftApplied

Type : Boolean
Description : true if an accidental graphical shift has to be applied to the note, false if it remains unchanged

Rule.AppliedAccidentalShift

Type : Number
Description : Shift in pixel to apply to the accidental.
Note : Only readable if Rule.IsAccidentalApplied (above) is true
Setting this value automatically switches Rule.IsAccidentalApplied (above) to true

Rule.IsPressureTimeApplied

Type : Boolean
Description : true if a pressure time has to be applied to the note, false if it remains unchanged

Rule.AppliedPressureTime

Type : Number
Description : Pressure time to apply to the note
Note : Only readable if Rule.IsPressureTimeApplied (above) is true
Setting this value automatically switches Rule.IsPressureTimeApplied (above) to true

Rule.IsDelayApplied

Type : Boolean
Description : true if a delay has to be applied to the note, false if it remains unchanged

Rule.AppliedDelay

Type : Number
Description : Delay to apply to the note (256th of quarter note)
Note : Only readable if Rule.IsDelayApplied (above) is true
Setting this value automatically switches Rule.IsDelayApplied (above) to true

Rule.IsColorApplied

Type : Boolean
Description : true if a color has to be applied to the note, false if it remains unchanged

Rule.AppliedColor

Type : Number
Description : Color index to apply to the note. Only have sense when splitting a staff according to its rules (otherwise the modified note is never seen)
Note : Only readable if Rule.IsColorApplied (above) is true
Setting this value automatically switches Rule.IsColorApplied (above) to true

Rule.AppliedShape

Type : Number
Description : Note head shape to apply to the note. (see constants in MSDefine (page 100).mys)
NO_SHAPE_NOTE=0
SHAPE_4_NOTE=1
SHAPE_7_NOTE_AIKEN=2
SHAPE_7_NOTE_SWAN=2
SHAPE_7_NOTE_WALKER=3
SHAPE_7_NOTE_FUNK=4
SHAPE_7_NOTE_JOHNSON=5
etc
Only have sense when splitting a staff according to its rules (otherwise the modified note is never seen)
Note : Shape of the note ALWAYS get this shape when the rule is matched. There is no boolean to avoid this.

Rule.IsHeadShiftApplied

Type : Boolean
Description : true if a head graphical shift has to be applied to the note, false if it remains unchanged

Rule.AppliedHeadShift

Type : Number
Description : head shift to apply to the note
Note : Only readable if Rule.IsHeadShiftApplied (above) is true
Setting this value automatically switches Rule.IsHeadShiftApplied (above) to true

Rule.IsSplitDurationApplied

Type : Boolean
Description : true if a splitting value has to be applied to the note, false if it remains unchanged

Rule.AppliedSplitDuration

Type : Number
Description : split duration to apply to the note
Note : Only available if Rule.IsSplitDurationApplied (above) is true
Setting this value automatically switches Rule.IsSplitDurationApplied (above) to true

Rule.IsStemHeightApplied

Type : Boolean
Description : true if a stem height has to be applied to the note, false if it remains unchanged

Rule.AppliedStemHeight

Type : Number
Description : stem height to apply to the note
Note : Only readable if Rule.IsStemHeightApplied (page 422) is true
Setting this value automatically switches Rule.IsStemHeightApplied (page 422) to true

Rule.IsStemTypeApplied

Type : Boolean
Description : true if a stem direction has to be applied to the note, false if it remains unchanged

Rule.AppliedStemType

Type : Number
Description : stem direction to apply to the note
Note : Only readable if Rule.IsStemTypeApplied (above) is true
Setting this value automatically switches Rule.IsStemTypeApplied (above) to true

4.5.1.1.5. Managing applied grace notes

When a rule is applied to a note, one or several grace notes can be added.
Management of this grace note list follows the same principles than a Symbol (page 298) grace note list.
It is recommended to learn first how grace notes can be inserted, modified or deleted when attached to a symbol before managing the rule-applied grace notes.
As soon as at least one applied grace note is defined for a rule, grace note play and display global settings (played duration, displayed duration, beams...) of the rule overrides the target note's.

Rule.NumberOfAppliedGraceNotes

Type : Number
Description : Number of applied grace notes in this rule. Grace note index can range from 1 to this value (included).

Rule.AppliedGraceNotes

Type : GraceNote array
Description : "Root" of applied grace note array for this rule.
Cannot be used as is. Must be indexed.
Example :

```
--- Displays "true" if the second applied grace note of a rule is an acciacatura (sounds after the note)
grace=myRule.AppliedGraceNotes[2]
if grace~=nil and grace.IsAfter==true then print("true") end
```

Rule.FirstAppliedGraceNote

Type : GraceNote
Description : First grace note in this rule (equivalent to Rule.AppliedGraceNotes[1])
nil if there is no grace note for being applied when this rule matches

Rule.LastAppliedGraceNote

Type : GraceNote
Description : Last grace note in this rule (equivalent to Rule.AppliedGraceNotes[Rule.NumberOfGraceNotes])
nil if there is no grace note for being applied when this rule matches

Rule.AppliedGraceNoteStdPlayDuration

Type : Number
Description : Standard play duration of all grace note.
Note : Only used if at least one grace note has to be applied when this rule matches.

Rule.AppliedGraceNoteStdDisplayDuration

Type : Number
Description : Standard display duration of all grace notes.
Note : Only used if at least one grace note has to be applied when this rule matches.

Rule.IsAppliedGraceNoteBeamHorizontal

Type : Boolean
Description : true if grace note beams are always horizontal.
Note : Only used if at least one grace note has to be applied when this rule matches.

Rule.IsAppliedGraceNoteSlur

Type : Boolean
Description : true if grace note is slured to its main note.
Note : Only used if at least one grace note has to be applied when this rule matches.

Rule.IsAppliedGraceNoteCrossAlone

Type : Boolean
Description : true if single grace notes are crossed.
Note : Only used if at least one grace note has to be applied when this rule matches.

4.5.1.1.6. Managing applied ornaments

When a rule is applied to a note, one or several ornaments can be added.
Management of this ornament list follows the same principles than a Symbol (page 298) ornament list.
It is recommended to learn first how ornaments can be inserted, modified or deleted when attached to a symbol before managing the rule applied ornaments.

Rule.NumberOfAppliedOrnaments

Type : Number
Description : Number of ornaments that will be applied. Ornament index can range from 1 to this value (included)..

Rule.AppliedOrnaments

Type : Ornament array
Description : "Root" of ornament array to be applied.
Cannot be used as is. Must be indexed.

Rule.FirstAppliedOrnament

Type : Ornament
Description : First ornament to be applied (equivalent to Rule.AppliedOrnaments[1])
nil if there is no ornament

Rule.LastAppliedOrnament

Type : Ornament
Description : Last ornament to be applied
nil if there is no ornament

4.5.1.1.7. Symbols

Rule.IsUseful

Type : Boolean
Description : true if a note use this rule (at least one note match this rule), false otherwise (the rule is not used)
Example :

```
-- Extract the first staff of the front score and check if rules are used
-- In this case, the rule name is displayed
score=FrontScore()
staff=score.FirstStaff
rule=staff.FirstRule
while rule~=nil do
    if rule.IsUseful==true then
        print(rule.Title.." is used")
    end
    rule=rule.Next
end
```

4.5.1.2. Rule methods

4.5.1.2.1. Deleting a rule

Rule.Delete(...)

Description : Deletes the rule
Parameters : None
Return value : true if successfully deleted
Example :

```
-- Deletes all rules in a staff, starting from the end to avoid index problems
rule=myStaff.LastRule
while rule~=nil do
    -- Once rule has been deleted, Previous member could not be retrieved
    previousRule=rule.Previous
    rule.Delete()
    rule=previousRule
end
```

4.5.1.2.2. Managing Applied grace notes

Rule.AddAppliedGraceNote(...)

Description : Adds a new grace note.
Parameters : (All parameters are optional)
GraceNote : Existing grace note BEFORE which the new one must be inserted.
nil or omitted if the grace note has to be inserted in last position
Boolean: true if the grace note has to be displayed after the note (acciacatura)
Number : Pitch offset from its refering note (semitone)
Number : Preferred accidental for displaying this grace note (see constants in MSDefine (page 100).mys)
NATURAL=0
SHARP=1
AUTO=2
FLAT=3
DOUBLE_SHARP=4
DOUBLE_FLAT=5
Number : Dividing value from the regular played and displayed length of grace note (see Symbol.GraceNoteStdPlayDuration (page 302) and Symbol.GraceNoteStdDisplayDuration (page 302) values)
0 means regular duration
1 means halved duration
2 means duration divided by 4
3 means duration divided by 8
etc
Number : Velocity from 0 to 127, or -1 if velocity is the note velocity (by default)
Boolean: true if grace note is in chord with the next one
Return value : GraceNote: newly inserted grace note, or nil if impossible to insert
Example :

```
myGraceNote=myRule.AddAppliedGraceNote(nil,false,-2)
```

Note : Up to 10 grace notes can be defined in a rule . Attempting to insert a 11th will fail, and return nil
When the rule is applied, grace notes are added after the existing in target note, if there is enough space in the grace note array (total number of grace notes cannot be over 10)

4.5.1.2.3. Managing applied grace notes

Rule.DeleteAllAppliedGraceNotes(...)

Description : Deletes all grace notes of a rule
Parameters : None
Return value : None
Example :

```
myRule.DeleteAllAppliedGraceNotes()
```

4.5.1.2.4. Managing applied ornaments

Rule.AddAppliedOrnament(...)

Description : Adds a new ornament to be applied to matching notes.
Parameters : (All parameters are optional)
Number : Ornament type (see constants in MSDefine (page 100).mys)
ORNAMENT_HAMMER_ON=1
ORNAMENT_PULLING_OFF=2
ORNAMENT_SLIDE=3
ORNAMENT_SLIDE_UP=4
ORNAMENT_SLIDE_DOWN=5
ORNAMENT_DADDI_PICKING=6
ORNAMENT_CHOKE=7
ORNAMENT_BEND=8
ORNAMENT_RELEASE=9
ORNAMENT_BEND_AND_RELEASE=10
etc
Number : X offset in 72 dpi pixel from its default position
Number : Y offset in 72 dpi pixel from its default position
Number : Display origin (see constants in MSDefine (page 100).mys)
ORNAMENT_TOP_STAFF=128
ORNAMENT_BOTTOM_STAFF=128+64
ORNAMENT_TOP_SYMBOL=0
ORNAMENT_BOTTOM_SYMBOL=64
ORNAMENT_LEFT_SYMBOL=32
ORNAMENT_RIGHT_SYMBOL=32+64
Boolean: true if this ornament will be played, false if it has no effect on sound output
Number : Color index for this ornament. Use <RGBToIndex> to get an index from Red, Green, Blue color settings.
Boolean: true if this ornament is invisible, i.e. not displayed on score.

Next parameters are only useful for ORNAMENT_TEXT ornament type
String : Text value for this ornament
String : Font name for displaying this text
Number : Font size (72 dpi pixel) for displaying this text
Number : Font face for displaying this text
Return value : Ornament : ornament object that has just been inserted (or nil if error)

Rule.DeleteAllAppliedOrnaments(...)

Description : Deletes all the applied ornaments in a rule.
Parameters : None
Return value : None

4.5.2. Instrument Object Type

General description of this object type : Instruments are related to a score. The same instrument can be used in staves, rules, accompaniments...
Instrument is an indexed object of Score (page 198). It means it refers internally to the Nth instrument of a score. Therefore, when a variable has been set to a given instrument, deleting an instrument that is located at a smaller index can make the variable change the instrument it refers to.

4.5.2.1. Instrument global functions

4.5.2.1.1. Managing Sound IDs

Management of sound IDs according to the current sound base

IsSoundIDAvailable(...)

Description : Tells whether a Sound ID is available in the current sound base
Parameters : Number: Sound ID
Return value : Boolean : true if the sound ID is available in the current sound base.
Example :

```
Include "MSSounds"
-- Lists all instruments in the score and tells whether their sound
-- is available in the current sound base
local myScore,myInstrument
myScore=FrontScore()
if myScore then
   myInstrument=myScore.FirstInstrument
   while myInstrument do
      if IsSoundIDAvailable(myInstrument.SoundID)==false then
         print("==> Instrument",myInstrument.Name,"uses sound",SoundArray[myInstrument.SoundID],"which is
         NOT present is current sound base")
      else
         print("    Instrument",myInstrument.Name,"uses sound",SoundArray[myInstrument.SoundID],"of current
         sound base")
      end
      myInstrument=myInstrument.Next
   end
end
```

ClosestAvailableSoundID(...)

Description : From a sound ID, provides the closest sound ID available in the current sound base.
Parameters : Number: Sound ID
Optional : Boolean : for drum sounds true if search has to be performed only within the same drum set (default= false)
Return value : Number : New Sound ID
Example :

```
Include "MSSounds"
-- Changes all sound ids in the front score to their closest available value in the current sound base
local myScore,myInstrument,newID
myScore=FrontScore()
if myScore then
   myInstrument=myScore.FirstInstrument
   while myInstrument do
      if IsSoundIDAvailable(myInstrument.SoundID)==false then
         newID=ClosestAvailableSoundID(myInstrument.SoundID,false);
         print("Instrument",myInstrument.Name,": used sound",SoundArray[myInstrument.SoundID])
         print("  It has been changed to ",SoundArray[newID])
         myInstrument.SoundID=newID
      end
      myInstrument=myInstrument.Next
   end
end
```

Note : When the sound is a drum sound, closest sound search can be performed only in the same drum set (if Boolean is true);
or in other drum sets (Boolean false, default)

GetSoundRange(...)

Description : From a sound ID provide the natural pitch range for this instrument
Parameters : Number: Sound ID
Return value : Numbers : minimal pitch, maximal pitch

4.5.2.2. Instrument values

4.5.2.2.1. Browsing Instruments

Instrument.Next

Type : Instrument
Description : Next valid instrument in the score
If there is no next instrument, value is nil.
Note :
Instruments are not sorted in score. They are inserted at the first free place in the instrument array.
Due to this special way of working, an "Instrument" variable will stick to its related instrument even if other instruments are added to or removed from the Instrument array.
But this implies new instruments won't be automatically inserted at the end of the instrument list. They can " appear" between two already existing instruments.

Instrument.Previous

Type : Instrument
Description : Previous valid instrument in the score
If there is no previous instrument, value is nil.
Note :
Instruments are not sorted in score. They are inserted at the first free place in the instrument array.
Due to this special way of working, an "Instrument" variable will stick to its related instrument even if other instruments are added to or removed from the Instrument array.
But this implies new instruments won't be automatically inserted at the end of the instrument list. They can " appear" between two already existing instruments.

Instrument.Index

Type : Number
Description : Index of this instrument in the score
Example :

```
-- Displays the name of the 2nd instrument, if exists
local myScore
-- Extracting score
myScore=FrontScore()
if(myScore~=nil) then
   -- Displays the name of the 2th instrument in this score
   if myScore.NumberOfInstruments>=2 then
      --- Extracting instrument #2
      instrument=myScore.Instruments[2];
      if instrument.IsUsed==true then
         Message(instrument.Name)
      else
         Message("This instrument #2 is unused")
      end
   else
      Message("There is no 2th instrument in this score")
   end
else
   Message("Please load a score")
end
```

4.5.2.2.2. Instrument general properties

Instrument.IsUsed

Type : Boolean
Description : true if this instrument is valid, false otherwise
Example :

```
--- Browse all the instruments of the front score and tell if the instrument is used or not.
local myScore
local index
local instrument
-- Extracting score
myScore=FrontScore()
if(myScore~=nil) then
   for index=1,myScore.NumberOfInstruments do
      --- Extracting instrument from index
      instrument=myScore.Instruments[index];
      --- Displaying value
      if instrument.IsUsed==true then
         print("The instrument #"..index.." name is "..instrument.Name.." and it is used.")
      else
      --- If the instrument is not used, we can't display is name.
         print("The instrument : #"..index.." is unused.")
      end
   end
else
   Message("Please load a score")
end
```

Note : Can only be false when accessing instruments in an indexed way, or when the instrument has been deleted.
When false, no other field can be read, and no field can be written.

Instrument.Score

Type : Score
Description : <Score> object this instrument belongs to.

Instrument.Name

Type : String
Description : Instrument name

Instrument.IsDrum

Type : Boolean
Description : true if the instrument is a drum, false if chromatic instrument
Note : Drum instruments are always played at their "standard" pitch.

Instrument.IsCustomInstrument

Type : Boolean
Description : true if the instrument is custom, false if standard

Instrument.SoundID

Type : Number
Description : Standard index of the sound in the sound base. Only used if Instrument.IsCustomInstrument (abov
is false.
Example :

```
Include "MSSounds"
-- Displays ID and name of all active instruments of the front score
local myScore
-- Extracting score
myScore=FrontScore()
if(myScore~=nil) then
   instrument=myScore.FirstInstrument
   while instrument~=nil do
      print("Instrument Name : '"..instrument.Name.."' is related to sound ID  "..instrument.SoundID.." ("..
      SoundArray[instrument.SoundID]..")")
      instrument=instrument.Next
   end
else
   Message("Please load a score")

end
```

Note : If Instrument.IsCustomInstrument is false, then from this index, the closest sound in the sound database is selected for playing this instrument, or, if in MIDI output, the closest MIDI program change.

Instrument.OutputDevice

Type : Number
Description : Output device currently selected for this instrument.
Note : Use only constants from MSDefine (page 100).mys:

```
NO_OUTPUT_DEVICE=-1
MIDI1_OUTPUT_DEVICE=9
MIDI2_OUTPUT_DEVICE=10
MIDI3_OUTPUT_DEVICE=11
MIDI4_OUTPUT_DEVICE=12
MIDI5_OUTPUT_DEVICE=13
MIDI6_OUTPUT_DEVICE=14
MIDI7_OUTPUT_DEVICE=15
MIDI8_OUTPUT_DEVICE=16
MIDI9_OUTPUT_DEVICE=17
MIDI10_OUTPUT_DEVICE=18
MIDI11_OUTPUT_DEVICE=19
MIDI12_OUTPUT_DEVICE=20
MYRSYNTH_OUTPUT_DEVICE=2
EXTERN_OUTPUT_DEVICE=3
DIGITAL_OUTPUT_DEVICE=6
SOFT_SYNTH_OUTPUT_DEVICE=7 -- Mac Only (previously QUICKTIME_OUTPUT_DEVICE)
```

Instrument.KeyShift

Type : Number
Description : Key shift (in semitone) for this instrument.
When playing a note, this value will be added to the note pitch to be played.
Warning: when setting this value while the score is not in Transposed mode (Score.TransposedMode (page 20
false), if the instrument
is used in a regular staff, the instrument key shift will be reset to the nearest octave, and the remainder
transferred to
the Staff.HiddenTransposition (page 268) member of the staves that use this instrument. When entering the
transposed mode again, the
instrument key shift will be restored to its correct value, and the staves transposed accordingly.

Instrument.VelocityOffset

Type : Number
Description : Velocity offset for this instrument.
Note : Actual output velocity ranges from 0 to 127. Values above or below these bounds will be clipped.

Instrument.VelocityDepth

Type : Number
Description : Velocity depth for this instrument.
Note : Actual output velocity ranges from 0 to 127. Values above or below these bounds will be clipped.

Instrument.IsVolume

Type : Boolean
Description : true if instrument volume is managed.

Instrument.Volume

Type : Number
Description : Volume (0-127) for this instrument.
Note : Only managed if Instrument.IsVolume (above) is true.

Instrument.IsPanning

Type : Boolean
Description : true if instrument panning is managed.

Instrument.Panning

Type : Number
Description : Panning (0=left to 127=right) for this instrument.
Note : Only managed if Instrument.IsPanning (above) is true.

Instrument.IsFading

Type : Boolean
Description : true if instrument fading (front-back) is managed.

Instrument.Fading

Type : Number
Description : Fading (0=front to 127=back) for this instrument.
Note : Only managed if Instrument.IsPanning (above) is true.

4.5.2.2.3. Instrument Layer Control

Instrument.LayerType

Type : Number
Description : Kind of layering applied to the instrument.
Can be one of the following values (from MSDefine):
. NO_LAYERING = The instrument is not layered. A Velocity filtering can be applied.
. VELOCITY_LAYERING = Layer is selected according to note velocity. Velocity filtering is then disabled.

Instrument.Layer1MaxInputValue

Type : Number
Description : Layer 1 will be the only to be selected if the input parameter (velocity) value falls between 0 and this value.
If the input value falls between this value and Layer2MinInputValue then layer 1 and 2 are played at the same time (mixed)

Instrument.Layer1MinOutputPower

Type : Number
Description : When layer 1 is selected, notes are sent with a power ranging from this value to Layer1MaxOutputPower

Instrument.Layer1MaxOutputPower

Type : Number
Description : When layer 1 is selected, notes are sent with a power ranging from Layer1MinOutputPower up to this value

Instrument.Layer2MinInputValue

Type : Number
Description : Layer 2 will be the only selected if the input parameter (velocity) value falls between this value and Layer2MaxInputValue
If the input value falls between Layer1MaxInputValue and this value then layer 1 and 2 are played at the same time (mixed)

Instrument.Layer2MaxInputValue

Type : Number
Description : Layer 2 will be the only to be selected if the input parameter (velocity) value falls between Layer2MinInputValue and this value.
If the input value falls between this value and Layer3MinInputValue then layer 2 and 3 are played at the same time (mixed)

Instrument.Layer2MinOutputPower

Type : Number
Description : When layer 2 is selected, notes are sent with a power ranging from this value to Layer2MaxOutputPower

Instrument.Layer2MaxOutputPower

Type : Number
Description : When layer 2 is selected, notes are sent with a power ranging from Layer2MinOutputPower up to this value

Instrument.Layer3MinInputValue

Type : Number
Description : Layer 3 will be the only selected if the input parameter (velocity) value falls between this value and Layer3MaxInputValue
If the input value falls between Layer2MaxInputValue and this value then layer 2 and 3 are played at the same time (mixed)

Instrument.Layer3MaxInputValue

Type : Number
Description : Layer 3 will be the only to be selected if the input parameter (velocity) value falls between Layer3MinInputValue and this value.
If the input value falls between this value and Layer4MinInputValue then layer 3 and 4 are played at the same time (mixed)

Instrument.Layer3MinOutputPower

Type : Number
Description : When layer 3 is selected, notes are sent with a power ranging from this value to Layer3MaxOutputPower

Instrument.Layer3MaxOutputPower

Type : Number
Description : When layer 3 is selected, notes are sent with a power ranging from Layer3MinOutputPower up to this value

Instrument.Layer4MinInputValue

Type : Number
Description : Layer 4 will be the only selected if the input parameter (velocity) value falls between this value and 127
If the input value falls between Layer3MaxInputValue and this value then layer 3 and 4 are played at the same time (mixed)

Instrument.Layer4MinOutputPower

Type : Number
Description : When layer 4 is selected, notes are sent with a power ranging from this value to Layer4MaxOutputPower

Instrument.Layer4MaxOutputPower

Type : Number
Description : When layer 4 is selected, notes are sent with a power ranging from Layer4MinOutputPower up to this value

4.5.2.2.4. Instrument MIDI settings

Instrument.MidiChannel

Type : Number
Description : Midi channel (1-16) for this instrument.
Note : Only managed in MIDI output.

Instrument.IsReverberation

Type : Boolean
Description : true if instrument MIDI reverberation is managed.
Note : Only processed in MIDI output

Instrument.Reverberation

Type : Number
Description : Panning (0=left to 127=right) for this instrument.
Note : Only managed if Instrument.IsReverberation (above) is true.

Instrument.IsChorus

Type : Boolean
Description : true if instrument MIDI chorus is managed.
Note : Only processed in MIDI output

Instrument.Chorus

Type : Number
Description : Chorus level (0-127) for this instrument.
Note : Only managed if Instrument.IsChorus (above) is true.

Instrument.MidiProgramChange

Type : Number
Description : MIDI program change for this instrument.
Note : Only available if Instrument.IsCustomInstrument (page 431) is true, otherwise it is calculated from Instrument.SoundID (page 431).

Instrument.IsMidiBankChange

Type : Boolean
Description : true if a bank change command is sent for this instrument.
Note : Only available if Instrument.IsCustomInstrument (page 431) is true, otherwise it is calculated from Instrument.SoundID (page 431).

Instrument.MidiBankChange

Type : Number
Description : MIDI bank change for this instrument.
Note : Only available if Instrument.IsCustomInstrument (page 431) is true, otherwise it is calculated from Instrument.SoundID (page 431).

Instrument.MidiDrumKey

Type : Number
Description : MIDI drum key to be played for this instrument.
Note : Only available if Instrument.IsCustomInstrument (page 431) is true, otherwise it is calculated from Instrument.SoundID (page 431).
Instrument must also be a drum (Instrument.IsDrum (page 431) must be true) otherwise this value is not used.

4.5.2.2.5. Instrument Velocity filtering

Instrument.IsVelocityFiltering

Type : Boolean
Description : true if velocity filtering applies, false if not processed

Instrument.InputVelocityFiltering

Type : Number
Description : For velocity filtering : velocity value at which sound played unfiltered

Instrument.MinVelocityFiltering

Type : Number
Description : For velocity filtering : velocity value below which low-velocity filter applies with the greatest possible power
Only applies if Instrument.IsVelocityFiltering (above) is true

Instrument.MaxVelocityFiltering

Type : Number
Description : For velocity filtering : velocity value above which high-velocity filter applies with the greatest possible power
Only applies if Instrument.IsVelocityFiltering (above) is true

Instrument.LowVelocityFirstFilter

Type : VelocityFilter
Description : First filter for low velocity filtering (VelocityFilter (page 440) object type)
Only applies if Instrument.IsVelocityFiltering (above) is true

Instrument.LowVelocitySecondFilter

Type : VelocityFilter
Description : Second filter for low velocity filtering (VelocityFilter (page 440) object type)
Only applies if Instrument.IsVelocityFiltering (above) is true

Instrument.HighVelocityFirstFilter

Type : VelocityFilter (page 440)
Description : First filter for high velocity filtering
Only applies if Instrument.IsVelocityFiltering (above) is true

Instrument.HighVelocitySecondFilter (R)

Type : VelocityFilter (page 440)
Description : Second filter for high velocity filtering
Only applies if Instrument.IsVelocityFiltering (page 435) is true

4.5.2.2.6. Instrument's Digital data

These fields are available only if Instrument.IsCustomInstrument (page 431) is true, otherwise digital sounds are calculated from Instrument.SoundID (page 431).

Instrument.IsCustomDigitalSounds

Type : Boolean
Description : true if instrument owns custom digital sounds.

Instrument.DigitalDrumPitch

Type : Number
Description : When instrument is a drum, pitch to play sound at.

Instrument.DigitalGroup

Type : Number
Description : Instrument exclusion group number. All sounds in the same groups, when played, stop the others.
Example : open and closed triangle or open, closed and pedal hihat...

Instrument.DigitalVolume

Type : Number
Description : Instrument volume from 1 to 1000 (0 sets the same volume as 1000 for compatibility reasons)

Instrument.NumberOfDigitalSounds (R)

Type : Number
Description : Number of digital sounds in this instrument
Only available if Instrument.IsCustomInstrument (page 431) AND Instrument.IsCustomDigitalSounds (above) are

Instrument.DigitalSounds (R)

Type : DigitalSound
Description : Digital sounds array for this instrument
Only available if Instrument.IsCustomInstrument (page 431) AND Instrument.IsCustomDigitalSounds (above) are

Instrument.FirstDigitalSound (R)

Type : DigitalSound
Description : First digital sound for this instrument
Only available if Instrument.IsCustomInstrument (page 431) AND Instrument.IsCustomDigitalSounds (above) are

Instrument.LastDigitalSound (R)

Type : DigitalSound
Description : Last digital sound for this instrument
Only available if Instrument.IsCustomInstrument (page 431) AND Instrument.IsCustomDigitalSounds (above) are

4.5.2.2.7. Instrument's Digital output settings

Instrument.IsCustomVibrato

Type : Boolean
Description : true if instrument has a custom digital vibrato. Otherwise, vibrato settings are taken from the current sound database.
Note : Only processed in digital output

Instrument.VibratoDelay

Type : Number
Description : Delay before custom vibrato starts (100th of second)
Note : Only processed in digital output

Instrument.VibratoRaiseDuration

Type : Number
Description : Duration of custom vibrato raising (100th of second)
Note : Only processed in digital output

Instrument.VibratoPeriod

Type : Number
Description : Period of custom vibrato oscillation (100th of second)
Note : Only processed in digital output

Instrument.VibratoDepth

Type : Number
Description : Custom vibrato power, or depth (1000th of semitone)
Note : Only processed in digital output

Instrument.IsCustomTremolo

Type : Boolean
Description : true if instrument has a custom digital tremolo. Otherwise, tremolo settings are taken from the current sound database.
Note : Only processed in digital output

Instrument.TremoloDelay

Type : Number
Description : Delay before custom tremolo starts (100th of second)
Note : Only processed in digital output

Instrument.TremoloRaiseDuration

Type : Number
Description : Duration of custom tremolo raising (100th of second)
Note : Only processed in digital output

Instrument.TremoloPeriod

Type : Number
Description : Period of custom tremolo oscillation (100th of second)
Note : Only processed in digital output

Instrument.TremoloDepth

Type : Number
Description : Custom tremolo power, or depth (percent of overall sound volume)
Note : Only processed in digital output

Instrument.IsCustomFXProcessor

Type : Boolean
Description : true if instrument has a custom FX processor. Otherwise, FX processor settings are taken from the current sound database.
Note : Only processed in digital output

Instrument.FXProcessor

Type : FXProcessor
Description : default FX processor for this instrument (these settings can be overriden by FXProcessor objects included in the staff.
Note : Only processed in digital output

4.5.2.2.8. Virtual Instrument

Instrument.IsVirtualInstrument

Type : Boolean
Description : true if instrument is virtual, i.e. uses physical modeling instead of sound base. Setting this value creates or delete the digital instrument
Note : Only processed in digital output

Instrument.VirtualInstrument

Type : VirtualInstrument
Description : Virtual Instrument details. If Instrument.IsVirtualInstrument is false, this member is nil.

4.5.2.3. Instrument methods

4.5.2.3.1. Managing

Instrument.Duplicate(...)

Description : Create a copy of the instument in the same score
Parameters : None
Return value : Instrument: the new instrument that has been created.

4.5.2.3.2. Managing digital sounds

Instrument.InsertDigitalSound(...)

Description : Inserts a DigitalSound (page 441) object in an instrument
Parameters : DigitalSound: Digital sound after which the new one must be inserted
nil if you want the new digital sound to be inserted at the beginning of the list
Number: value for DigitalSound.RangeMaxPitch (page 442)
Optional: Number: Layer number (default=1)
Return value : DigitalSound: Inserted digital sound
or nil if there is already too much sounds in this instrument
Example :

```
-- Inserts a digital sound in an instrument
while myInstrument.NumberOfDigitalSounds>0 do myInstrument.FirstDigitalSound.Delete() end
```

Note : The inserted digital sound is empty.
It is made of 10 zeroed samples (below this number, Harmony can crash when playing it) at 44 Khz, 16 bits
DigitalSound.RecordedPitch (page 442) is set to 60 (C4), loops are empty, attack, decay, release are 0.
You are responsible for sorting the digital sounds in increasing order of DigitalSound.Layer (page 442), then in increasing number of DigitalSound.RangeMaxPitch (page 442).
If it is not the case, then you must call Instrument.SortDigitalSounds(...) (below) before using the instrument.
This operation can change the order of the sounds, so you can't rely on any reference to a digital sound that has been got before it.

Instrument.SortDigitalSounds(...)

Description : Sorts the digital sounds in increasing order of layer number, then in increasing number of pitch range
Parameters : none
Return value : none
Note : This operation can change the order of the sounds, so you can't rely on any reference to a digital sound that
has been got before it.

4.5.2.3.3. Managing instruments

Instrument.Delete(...)

Description : Delete the instrument
Parameters : None
Return value : None
Note : The digital sounds of this instrument must not be used after the instrument is deleted.

4.5.2.3.4. Managing Instruments

Instrument.CopyTo(...)

Description : Copy the instrument to another instrument. You can copy instrument even from a score to another.
Parameters : instrument : the destination instrument.
Return value : Boolean : true if the instrument was successfully copied.

4.5.3. VelocityFilter Object Type

General description of this object type : Digital velocity filter settings for a digital Instrument (page 428)
Velocity filtering is applied according to a range of output velocities, and enables to muffle low-powered sounds, and to brighten hi-powered sounds of the same instrument,in order to make the result more realistic.
Two of these filters are applied progressively to low velocity sounds (Instrument.LowVelocityFirstFilter (page 435) and Instrument.LowVelocitySecondFilter (page 435)) while two others are applied progressively to high velocity sounds (Instrument.HighVelocityFirstFilter (page 435) and Instrument.HighVelocitySecondFilter (page 436)).

4.5.3.1. VelocityFilter values

4.5.3.1.1. Setup

VelocityFilter.IsHighPass

Type : Boolean
Description : true if this filter is High-Pass (low frequencies are softened), false if Low-Pass (high frequencies are softened)

VelocityFilter.Power

Type : Number (floating-point)
Description : Filtering maximum power, from 0 to 100%

VelocityFilter.CutOffHarmonic

Type : Number (floating-point)
Description : Sound harmonic number at which the cut-off applies, from 0.1 to 29.9
Note : an harmonic is a multiple of the sound frequency.
For example, if this value is 2 and the sound is played at 440 Hz, cutoff frequency will be 880 Hz
If the filter is high-pass, frequencies below 880 Hz will be softened, if it is low-pass, frequencies above 880 Hz will be softened.

VelocityFilter.Resonance

Type : Number (floating-point)
Description : Filter resonance, from 0 to 100%
Note : Resonance emphazises frequencies around the cutoff value

4.5.4. DigitalSound Object Type

General description of this object type : Digital sound for an Instrument (page 428).
A single digital instrument can be made of several digital sounds, that are used by Harmony Assistant according to the played note pitch.

4.5.4.1. DigitalSound values

4.5.4.1.1. Browsing digital sounds

DigitalSound.Next

Type : DigitalSound
Description : Next digital sound in the instrument
If there is no next digital sound, value is nil.
Note : You can browse digital sounds through Next and Previous fields, but you can also access a digital sound at a given index through Instrument.DigitalSounds[index]

DigitalSound.Previous

Type : DigitalSound
Description : Previous digital sound in the instrument
If there is no previous digital sound, value is nil.
Note : You can browse digital sounds through Next and Previous fields, but you can also access a digital sound at a given index through Instrument.DigitalSounds[index]

4.5.4.1.2. Digital sound envelope

DigitalSound.AttackSpeed

Type : Number
Description : Attack speed of this sound from 1 (slow) to 10000 (fast)
Attack is the speed at which sound volume increases when a note starts
Note : Real time for the attack is calculated from this value through the formula :
time in ms = 40000/value
i.e. for a value of 1, time is :40000 ms = 40 seconds
for a value of 10000, time is : 4 ms
For compatibility reasons,a value of 0 sets an immediate attack

DigitalSound.DecaySpeed

Type : Number
Description : Decay speed of this sound from 1 (slow) to 10000 (fast)
Decay is the speed at which sound volume decreases between the time attack is terminated and the time note is released
Note : Real time for the decay is calculated from this value through the formula :
time in ms = 160000/value
i.e. for a value of 1, time is :160000 ms = 160 seconds
for a value of 10000, time is : 16 ms

DigitalSound.ReleaseSpeed

Type : Number
Description : Release speed of this sound from 1 (slow) to 10000 (fast)
Release is the speed at which sound volume decreases after the note is released
Note : Real time for the relase is calculated from this value through the formula :
time in ms = 160000/value
i.e. for a value of 1, time is :160000 ms = 160 seconds
for a value of 10000, time is : 16 ms

4.5.4.1.3. Digital sound format

DigitalSound.SampleSize

Type : Number
Description : Sample size, in bits. A sample can be either 8 or 16 bits
8-bits samples are unsigned values from 0 to 255
16-bits samples are signed values from -32768 to 32767

DigitalSound.SampleRate

Type : Number
Description : Sample rate, in Hz. Possible values are 11025, 22050, 44100, 48000 or 96000
Note : Changing this value changes resamples the sound to fit the new sample rate.
Due to this, DigitalSound.NumberOfSamples (page 443), DigitalSound.AttackSize (below), DigitalSound.LoopS below) and DigitalSound.LoopEnd (page 443) are adjusted accordingly.

4.5.4.1.4. Digital sound miscellaneous settings

DigitalSound.Layer

Type : Number
Description : Sound layer number. Layers are controlled through fields in the parent Instrument object

DigitalSound.RecordedPitch

Type : Number
Description : Played pitch in the recorder sound. It enables Harmony Assistant to know how to stretch the sound to make it play other notes.
Note : This value is provided in midi pitch (semitones, from 0 to 127, 60 = middle C)

DigitalSound.RangeMaxPitch

Type : Number
Description : Maximum pitch for which this sound will be used.
Note : This value is provided in midi pitch (semitones, from 0 to 127, 60 = middle C).
You are responsible to keep this value increasing from sound to sound in order to keep a coherent sound array.

The last sound must have a maximum pitch of 127 to cover the whole range of possible values.

DigitalSound.AttackSize

Type : Number
Description : Sound shrinkable attack size, in number of samples. When a note is shorter in time than this attack size, the first samples are skipped in order to make the sound start faster.
Note : You are responsible for keeping this field in the range of available digital samples (DigitalSound.NumberOfSamples (page 443)).

DigitalSound.IsInfiniteLoop

Type : Boolean
Description : true if loop is infinite, false if not
Note : When loop is infinite, loop point remains active on this sound even after the note is released.
If not infinite, loop stops as soon as the note is released, and part of the sound located after the loop is played.

DigitalSound.IsDecayWaitForLoop

Type : Boolean
Description : true if decay only starts once the loop point has been reached

DigitalSound.LogDecayRatio

Type : Number
Description : Logarithmic decay/release ratio, in percent
Note : 0 means decay and releases are linear, 100 means decay and release are purely logarithmic.

DigitalSound.LoopStart

Type : Number
Description : Position of loop start, in samples
Note : You are responsible for keeping this field in the range of available digital samples (DigitalSound.NumberOfSamples (page 443)).

DigitalSound.LoopEnd

Type : Number
Description : Position of loop end, in samples
If there is no loop in this sample, both LoopStart and LoopEnd are 0
Note : You are responsible for keeping this field in the range of available digital samples (DigitalSound.NumberOfSamples (below)).
This field is not an integer number, enabling to manage precise loops that don't cover an integer number of samples

4.5.4.1.5. Digital sound data

DigitalSound.NumberOfSamples

Type : Number
Description : Number of digital samples in this sound

DigitalSound.NativeDigitalData

Type : DigitalData
Description : Digital data (samples) array for this sound, in native format (values depend on the sample size)

DigitalSound.NormalizedDigitalData

Type : DigitalData
Description : Digital data (samples) array for this sound, in normalized format (values are expressed from -1 to 1)

4.5.4.2. DigitalSound methods

4.5.4.2.1. Managing digital sounds

DigitalSound.Delete(...)

Description : Deletes a digital sound from an Instrument (page 428) object
Parameters : None
Return value : None
Example :

```
-- Erases all digital sounds of an instrument
while myInstrument.NumberOfDigitalSounds>0 do myInstrument.FirstDigitalSound.Delete() end
```

DigitalSound.Export(...)

Description : Exports a digital sound from an Instrument (page 428) object to a digital audio file
Parameters : · String : The complete file name.
· Number (Optional): the file format as defined in MSDefine (page 100)
FORMAT_WAV=15
Return value : True if file saved.
Example :

```
-- Saves all digital sounds of an instrument
for i=1,myInstrument.NumberOfDigitalSounds do
    myInstrument.DigitalSounds.Export(GetUserExportPathName("Music Export").."/Export "..myInstrument.
    Name.."-"..i..".wav")
end
```

Note : Currently, only WAV format is available. In WAV format, loop start and loop end positions are stored in the file

DigitalSound.Import(...)

Description : Imports a digital sound from digital audio file to an Instrument (page 428) object
Parameters : · String : The complete file name
· Number (Optional): the file format as defined in MSDefine (page 100)
FORMAT_WAV=15
Return value : True if the file has been loaded.
Note : Currently, only WAV format is available. In WAV format, loop start and loop end positions are stored in the file
This method only replaces digital data (and loop points) of an existing digital sound. All other parameters remain unchanged

4.5.5. DigitalData Object Type

General description of this object type : Digital data for a DigitalSound (page 441) or a digital track
A DigitalData object is managed as a special array.
Indexing it through brackets provides the sample value at this index, either in native format (0-255 for 8-bits samples, -32768-32767 for 16-bits samples)or in normalized format (-1 to 1 whatever the sample size is)
Setting a sample value changes this value. If the index is outside the available range of sample index, data are appended automatically.
However, it is recommended, for performance reason, before working on samples indexes that exceeds the original number of samples, to append first the data area, either by using DigitalData.Insert(...) (page 446), or simply by setting fir the value of the last sample.
For example: myDigitalData[futureMaxIndex]=0 (or myDigitalData[futureMaxIndex]=128 if data are 8-bits in size)

Getting value of a named field on this kind of object provides read-only information about the object (sample size, sample rate)
Calling methods on this kind of object performs various actions on a whole range of samples.
Example :

```
------ Creating a 440 Hz sine curve of 1/2 second --------
size=myDigitalData.SampleRate/2 -- #samples in 1/2 sec
-- Delete previous data
myDigitalData.Delete(1,myDigitalData.NumberOfSamples)
-- Insert new data
myDigitalData.Insert(1,size)
-- Calculate sine curve
for sampleIndex=1,size do
   -- 8-bits samples
   if myDigitalData.SampleSize==8 then
      data[sampleIndex]=128+127*sin(sampleIndex*440*360/myDigitalData.SampleRate)
   -- 16-bits samples
   elseif myDigitalData.SampleSize==16 then
      data[sampleIndex]=32767*sin(sampleIndex*440*360/myDigitalData.SampleRate)
   -- Normalized samples
   else
      data[sampleIndex]=sin(sampleIndex*440*360/myDigitalData.SampleRate)
   end
end
```

4.5.5.1. DigitalData values

4.5.5.1.1. Managing digital data

DigitalData.SampleSize

Type : Number
Description : Sample size, in bits. A sample can be either 8 or 16 bits, or 0 if the current DigitalData object is accessed through "normalized" data, i.e. values from -1 to 1 (in floating-point)

8-bits samples are unsigned values from 0 to 255
16-bits samples are signed values from -32768 to 32767
normalized samples are floating-point signed values from -1 to 1. Their value is converted to and from the native 8-bits or 16-bits values.
It means you may not read exactly the same value you set.

DigitalData.SampleRate

Type : Number
Description : Sample rate, in Hz. Possible values are 11025, 22050, 44100, 48000 or 96000. Setting this value changes the data to match the new rate.
This process can be destructive and has to be performed with caution.

DigitalData.NumberOfSamples

Type : Number
Description : Number of samples

DigitalData.EmbeddingDigitalSound

Type : DigitalSound
Description : If this DigitalData object represents the data for a DigitalSound object, returns the embedding DigitalSound object.
If this DigitalData object represents the data for a digital track, returns nil.

4.5.5.2. DigitalData methods

4.5.5.2.1. Managing digital data

DigitalData.Insert(...)

Description : Inserts a range of samples in a digital data array.
Parameters : Number: Start position, in sample
Number: Number of samples to insert
If the DigitalDataObject is related to a DigitalTrack (page 449) object, an additional parameter can be specified :
Boolean: true if insertion must stop at the next marker, false if not (false is the default value)
Return value : None
Example :

```
-- Inserts 200 samples at the end of a sound
myDigitalData.Insert(myDigitalSound.NumberOfSamples,200)
```

Note : If working on a digital instrument sound, you are responsible for moving DigitalSound.NumberOfSamples (page 443), DigitalSound.AttackSize (page 442), DigitalSound.LoopStart (page 442) and DigitalSound.LoopEnd (p accordingly, if necessary.

DigitalData.Delete(...)

Description : Deletes a range of samples in a digital data array.
Parameters : Number: Start position, in sample
Number: Number of samples to delete
If the DigitalDataObject is related to a DigitalTrack (page 449) object, an additional parameter can be specified :
Boolean: true if deletion must stop at the next marker, false if not (false is the default value)
Return value : None
Example :

```
-- Keeps only the 200 first samples in a sound
myDigitalData.Delete(200,myDigitalSound.NumberOfSamples-200)
myDigitalSound=myDigitalData.EmbeddingDigitalSound
if myDigitalSound then
   if myDigitalSound.LoopStart>200 then myDigitalSound.LoopStart=200 end
   if myDigitalSound.LoopEnd>200 then myDigitalSound.LoopEnd=200 end
   if myDigitalSound.AttackSize>200 then myDigitalSound.AttackSize=200 end
end
```

Note : If working on a digital instrument sound, you are responsible for moving DigitalSound.NumberOfSamples (page 443), DigitalSound.AttackSize (page 442), DigitalSound.LoopStart (page 442) and DigitalSound.LoopEnd (p accordingly, if necessary.

DigitalData.Crop(...)

Description : keeps only a range of samples in a digital data array.
Parameters : Number: Start position, in sample
Number: Number of samples to keep
Return value : None
Example :

```
-- Keeps only 200 samples at position 1000 in a sound
myDigitalData.Crop(1000,200)
myDigitalSound=myDigitalData.EmbeddingDigitalSound
if myDigitalSound then
   if myDigitalSound.LoopStart<1000 then myDigitalSound.LoopStart=0
   elseif myDigitalSound.LoopStart<1200 then myDigitalSound.LoopStart=myDigitalSound.LoopStart-1000
   else myDigitalSound.LoopStart=200
   if myDigitalSound.LoopEnd<1000 then myDigitalSound.LoopEnd=0
   elseif myDigitalSound.LoopEnd<1200 then myDigitalSound.LoopEnd=myDigitalSound.LoopEnd-1000
   else myDigitalSound.LoopEnd=200
   if myDigitalSound.AttackSize<1000 then myDigitalSound.AttackSize=0
   elseif myDigitalSound.AttackSize<1200 then myDigitalSound.AttackSize=myDigitalSound.AttackSize-
   1000
   else myDigitalSound.AttackSize=200
   end
end
```

Note : If working on a digital instrument sound, you are responsible for moving DigitalSound.NumberOfSamples (page 443), DigitalSound.AttackSize (page 442), DigitalSound.LoopStart (page 442) and DigitalSound.LoopEnd (p accordingly, if necessary.
If working on a digital track, markers are NOT taken into account.

DigitalData.Erase(...)

Description : Sets a range of digital data to 0 (or 128 if sample size is 8 bits)
Parameters : Number: Start position, in sample
Number: Number of samples to erase
Return value : None
Example :

```
-- Erases a whole digital data area
myDigitalData.Erase(1,myDigitalData.NumberOfSamples)
```

Note : If working on a digital track, markers are NOT taken into account.

DigitalData.GetBlock(...)

Description : Get a range of digital data values into a RawTable (page 451) object
Parameters : Number: Position of the first sample in the digital data object to be retrieved
Number: Number of samples to get
RawTable: Table where to store data (first data stored at index 0)
-- Optional --
Number: First index in RawTable where to store the result - default : 0 -
Return value : None
Example :

```
-- Allocate RawTable object
myRawTable=NewRawTable(44100)
-- Get the first second of data into a table
myDigitalData.GetBlock(1,44100,myRawTable)
-- processing
-- to be written
-- Free RawTable
myRawTable.Dispose()
```

Note : [.] When reading values outside the range of available digital data, the RawTable is padded with 0db values (i.e. 0 for normalized and 16bits values, 128 for 8-bits values).
[.]This method is functionnally equivalent to (but much faster than) something like:

```
for i=0,number-1 do
if myDigitalData[start+i] then
table[i+1]=myDigitalData[start+i]
else
table[i+1]=0 -- should be 128 if data are 8-bits
end
end
```

DigitalData.SetBlock(...)

Description : Sets a range of digital data from RawTable (page 451)
Parameters : RawTable: table containing the data to set
Number: Starting position in the DigitalData object where to write the block
Optional :
Number: number of samples to copy (default : full table)
Return value : None
Example :

```
-- Fade in the first second of data at 44kHz sample rate
table=NewRawTable(44100)
myDigitalData.GetBlock(1,44100,table)
for i=1,44100 do
    table=table*i/44100 -- ascending ramp
end
myDigitalData.SetBlock(table,1)
table.Dispose()
```

4.5.6. DigitalTrack Object Type

General description of this object type : A DigitalTrack object is related to a staff. It enables to manage digital track settings and digital data.
Markers are not sorted.

4.5.6.1. DigitalTrack values

4.5.6.1.1. Digital Data

DigitalTrack.NativeDigitalData

Type : DigitalData
Description : Digital data for this track in native format (44 KHz, 16 bits).

DigitalTrack.NormalizedDigitalData

Type : DigitalData
Description : Digital data for this track in normalized format (44 KHz, values from -1 to 1).

4.5.6.1.2. Digital track attributes

DigitalTrack.Panning

Type : Number
Description : Default panning position for this Digital track.

DigitalTrack.Volume

Type : Number
Description : Default volume for this Digital track.

DigitalTrack.IsFramed

Type : Boolean
Description : true if frame is displayed around digital track

DigitalTrack.FrameColor

Type : Number
Description : Color index for frame

DigitalTrack.BackgroundColor

Type : Number
Description : Color index for background

DigitalTrack.IsDataDisplayed

Type : Boolean
Description : true if frame is displayed around digital track

DigitalTrack.DataColor

Type : Number
Description : Color index for displaying data

DigitalTrack.IsDisplayTime

Type : Boolean
Description : True if time position is displayed

4.5.6.1.3. Markers

DigitalTrack.NumberOfMarkers

Type : Number
Description : Number of markers in this digital track. Markers can be got or set through DigitalTrack.GetMarker(...) (below) and DigitalTrack.SetMarker(...) (below) methods

4.5.6.1.4. Selection

DigitalTrack.BeginSelection

Type : Digital track position
Description : Provide the position in a digital track of the beginning of the selection, expressed in 100th of millisecond.
Can be used for getting selection range on a digital track.
If the digital track is not currently selected, return -1
Setting this value put the selection range to the digital track. To put it back to something else, use either Score.TimeBeginSelection (page 219) or Score.TimeEndSelection (page 219).

DigitalTrack.EndSelection

Type : Digital track position
Description : Provide the position in a digital track of the end of the selection expressed in 100th of millisecond.
Can be used for getting selection range on a digital track.
If the selection is not set to a digital track, returns the position of the last repeat of the end of the selection range.
Setting this value put the selection range to the digital track. To put it back to something else, use either Score.TimeBeginSelection (page 219) or Score.TimeEndSelection (page 219).

4.5.6.2. DigitalTrack methods

4.5.6.2.1. Managing Markers

DigitalTrack.GetMarker(...)

Description : Gets the indexed marker position, in sample.
Parameters : Number: marker index from 1 to DigitalTrack.NumberOfMarkers (above)
Return value : Marker position in samples from the beginning of the track

DigitalTrack.SetMarker(...)

Description : Changes position of an existing marker.
Parameters : Number: marker index from 1 to DigitalTrack.NumberOfMarkers (above)
Number: New marker position in samples from the beginning of the track
Return value : None

DigitalTrack.AddMarker(...)

Description : Adds a marker.
Parameters : Number: marker position in samples from the beginning of the track
Return value : Marker index or nil if impossible to add.
Note : the new marker is added at the end of the marker list.

DigitalTrack.DeleteMarker(...)

Description : Deletes a existing marker.
Parameters : Number: marker index
Return value : none.

4.5.7. RawTable Object Type

General description of this object type : RawTable object type consists in a pseudo-lua table that contains only numerical values, whose indexes are consecutive, and which cannot be ressized.
It provides very fast access to some tables when time-critical operations are performed.
For example, spectrum operations on DigitalData (page 445) objects use this kind of raw tables

A raw table has to be explicitely allocated and dimensioned before being used, through the NewRawTable(...) (below) global function, and disposed when not needed anymore by the program, through RawTable.Dispose(...) (page 452) met

When allocated, its size is fixed and cannot be changed. Indexes in such a raw table range from 0 to the maximum index provided when allocating it.
Each element of the table can only contain a int32_t integer or floating-point value (specified when allocated), and is accessed through bracketing.

4.5.7.1. RawTable global functions

4.5.7.1.1. Creating

NewRawTable(...)

Description : Creates and allocates a new raw table. It is filled with 0 values.
Parameters : Number: Number of elements in this table
-- Optional --
Number: kind of value that can be stored in each table element
Can be either :
RAW_TABLE_FLOAT (32 bits floating point value) - default -
or RAW_TABLE_LONG (32 bits int32_t integer)
Return value : The newly allocated table or nil if the function failed.
Example :

```
-- Allocates a raw table
myRTable=NewRawTable(1000)
if myRTable then
   -- stores a sinus curve into table
   for i=0,1000 do
      myRTable=sin(i*360/1000)
   end
   print(myRTable[250]) -- sin(90 degrees) equals 1
   -- Free table.
   myRTable.Dispose()
end
```

Note : - RawTable indexes are zero-based. It means, when you allocate a new table with a given number of elements, indexes will range from 0 to number of elements-1, included.
- RawTable indexes are integer only. It means when using a non-integer value to index such a table, the smaller integer will be used.
- You must free memory occupied by a RawTable through RawTable.Dispose(...) (page 452) method, once you d not need this table anymore.

4.5.7.2. RawTable values

4.5.7.2.1. Getting information about a raw table

RawTable.Size

Type : Number
Description : Number of elements in the raw table

RawTable.ContentType

Type : Number
Description : Raw Table content type

RawTable.MaximumValue (R)

Type : Number
Description : Maximum value stored in the raw table

RawTable.MinimumValue (R)

Type : Number
Description : Minimum value stored in the raw table

RawTable.AverageValue (R)

Type : Number
Description : Average value in the raw table

4.5.7.3. RawTable methods

4.5.7.3.1. Disposing

RawTable.Dispose(...)

Description : Disposes a previously created raw table

4.5.7.3.2. Fast operation on RawTable

RawTable.Fill(...)

Description : Fills a whole RawTable or a part of a RawTable with a fixed value
The source rawtable
Parameters : Number: Value to set
-- Optional --
Number: first index to process (default : 0)
Number: number of elements to fill (default: to the last available index)
Return value : None

RawTable.ToBase64(...)

Description : Create a Base64 string from the content of the whole rawtable.
Each element of the rawtable will be represented by one byte (0-255). Therefore, the base64 result might not reflect the exact content of the rawtable
Parameters : Number: value in the rawtable that will be output as 0
Number: value in the rawtable that will be output as 255
-- Optional --
Number: number of characters per line, a return char being inserted at the end of each line (default : 0 = none)
Return value : String: the resulting Base64 string
Note : For instance, if the rawtable is float and you call rawtable.ToBase64(-1,1), you will get a base64 string that contains 0 for all values smaller than -1,
255 for all values greater that +1, and a value from 0 to 255 for each float value in between.

RawTable.FromBase64(...)

Description : Fills a rawtable with the binary data decoded from a Base64 string
Each byte extracted from the input string will be one element in the rawtable. The rawtable can (will) therefore be resized to match the result size
Parameters : Number: value in the rawtable that will match the 0 value
Number: value in the rawtable that will match the 255 value
-- Optional --
Return value : None
Note : For instance, if the rawtable is float and you call rawtable.FromBase64(b64string,-1,1), you will get a rawtable that contains -1 for byte value of 0, +1 for byte value of 255, and from -1 to +1 for each byte value in between.

RawTable.ApplyWindow(...)

Description : Applies a windowing function to a RawTable
The source rawtable contains the data to process (and will receive the result)
Parameters : umber: Windowing function :
WINDOWING_NONE = no windowing,
WINDOWING_TRIANGLE = triangle window
WINDOWING_WELCH = Welch window
WINDOWING_HANNING = Hanning window
WINDOWING_HAMMING = Hamming window
Return value : None. Windowed data are stored in place into the original RawTable.

RawTable.Bound(...)

Description : Bound all elements of a RawTable to a given range.
Parameters : Number: minimum value (elements below this number will be set to this value) or nil if no low boundary check.
Number: maximum value (elements above this number will be set to this value) or nil if no high boundary check.
Return value : None. Cropped data are stored in place into the original RawTable.

RawTable.Operation(...)

Description : Performs operation between two RawTables
The source rawtable will receive the processed data
Parameters : Number: first index to process
Number: number of elements to process
RawTable: Operation RawTable (if a Number, uses it as a constant. The next parameter is then ignored)
Number: First index in operation table
Number: Operation type
0 (OP_COPY) Copies operation table elements into target
1 (OP_ADD) Adds operation table elements to target
2 (OP_MULT) Multiplies target by operation table elements, element to element (windowing)
3 (OP_CONVOLUTION) Sliding convolution window.
In a convolution, the operation table is shorter than the target
For each element of the target, the convolution window is applied (centered) to the neighbours of this element.
Return value : None
Note : The two rawtables (if not using the operation rawtrable parameter as a constant) have to be of the same type (float or int32_t)
No test is performed on value overflow.

4.5.7.3.3. Spectrum operation on RawTable

These functions process digital data stored in RawTables

RawTable.FFT(...)

Description : Performs a Fast Fourier Transform on a block of digital data stored into a RawTable
A Fast Fourier Transform enables to extract frequency-related information from a block of digital data.
From the provided digital data block :
- First, a "windowing" operation is applied to the data.
This windowing enables the begin/end edges of the digital data block to be smoothed, in order not to generate unwanted spectrum peaks.
- Then the FFT is applied. It transforms the windowed digital data block into a spectrum (two arrays of twice less values called "power" and "phase").
Instead of being, like the original block of data, an amplitude/time diagram, the result of the FFT is a power/frequency and phase/frequency diagram.
Here is how to understand the FFT results :
Each array index i (power or phase array) matches the frequency of a sine curve that would have i periods in the time of the digital data block.
It means, for an FFT processed on a 1024 sample block at a 44.1 kHz sampling rate :
array element #0 is the "bias" value for the block
array element #1 is the value at (1*44100)/1024 = 43.066 Hz
array element #2 is the value at (2*44100)/1024 = 86,133 Hz
...
array element #256 is the value at (256*44100)/1024 = 11025 Hz
...
array element #511 is the value at (511*44100)/1024 = 22006,934 Hz
array element #512 is the value at (512*44100)/1024 = 22050 Hz
There is no value above this index, because no frequency information can be relevant above half of the sampling frequency (Nyquist theorem).

"Power" array reflects the energy of each frequency in the spectrum
"Phase" array reflects information about phase (in radian) of each frequency in the spectrum

From a "Power" and "Phase" array, it is possible to reconstruct a digital data block through an Inverse Fast Fourier Transform (IFFT) operation.

Related entries : DigitalData.GetBlock(...) (page 447), DigitalData.SetBlock(...) (page 448), RawTable.IFFT(...) (b
PitchToFrequency(...) (page 299), FrequencyToPitch(...) (page 299)

The source rawtable contains digital data
Parameters : Number: number of bits in each sample
0= normalized samples (range : -1..1)
8= 8-bits samples (range : 0..255)
16= 16-bits samples (range : -32768..32767)
Number: Windowing function :
WINDOWING_NONE = no windowing,
WINDOWING_TRIANGLE = triangle window
WINDOWING_WELCH = Welch window
WINDOWING_HANNING = Hanning window
WINDOWING_HAMMING = Hamming window
RawTable: Table that will receive the spectrum power values (must be half the size of the original RawTable +1)
-- Optional --
RawTable: Table that will receive the spectrum phase values (must be half the size of the original RawTable +1)
Return value : None
Note : Number of samples in the source RawTable must be a power of 2 (ex : 1024,2048,4096...)

RawTable.IFFT(...)

Description : Performs an Inverse Fast Fourier Transform on a spectrum and the RawTable with digital data
A spectrum is stored into two RawTables (float), the first one being the spectrum 'power' array, and the second one the spectrum 'phase' array.
Number of elements in the two tables must be equal, and match a power of 2.
The resulting table of digital data has twice more elements than the source tables.
Related entries : DigitalData.GetBlock(...) (page 447), DigitalData.SetBlock(...) (page 448), RawTable.FFT(...) (ab
The source rawtable will receive the computed data
Parameters : RawTable: spectrum power values
RawTable : spectrum phase values
Number: number of bits in each sample
0= normalized samples (range : -1..1)
8= 8-bits samples (range : 0..255)
16= 16-bits samples (range : -32768..32767)
Return value : None

RawTable.Cepstrum(...)

Description : Computes the cepstrum of a spectrum
Encycl : (Math) Cepstrum is the Fourier transform of the logarithm of the autospectrum. Useful for determining periodicities in the autospectrum.
Additions in the cepstrum domain correspond to multiplication in the frequency domain and convolution in the time domain.

Related entries : DigitalData.GetBlock(...) (page 447), DigitalData.SetBlock(...) (page 448), RawTable.FFT(...) (pa
RawTable.IFFT(...) (page 454),RawTable.ICepstrum(...) (below)
The source rawtable contains the power spectrum
Parameters : RawTable: Table to receive the cepstrum value (same size as the input spectrum power table)
Example :

```
-- Allocate RawTable for digital data
rawdata=NewRawTable(1024)
-- Read Digital data on a digital track
myDigitalData.GetBlock(0,1024,rawdata)
-- Allocate & compute FFT (spectrum)
power=NewRawTable(1024/2+1)
phase=NewRawTable(1024/2+1)
rawdata.FFT(0,WINDOWING_TRIANGLE,power,phase)
-- Calculates cepstrum of power spectrum
cepstrum=NewRawTable(1024/2+1)
power.Cepstrum(cepstrum) -- stored in "cepstrum" RawTable
```

RawTable.ICepstrum(...)

Description : Computes an inverse cepstrum
From a cepstrum table, recreates back a spectrum power table.
Related entries : DigitalData.GetBlock(...) (page 447), DigitalData.SetBlock(...) (page 448), RawTable.FFT(...) (pa
RawTable.Cepstrum(...) (above)
The source rawtable contains the cepstrum
Parameters : RawTable: RawTable to receive the reconstructed power spectrum
Return value : None

4.5.7.3.4. Spectrum operation on RawTables

RawTable.SineSynth(...)

Description : Spectrum SineSynth generator is an Sine-Add type .
It enables a complex spectrum-oriented sound to be generated through its partials (harmonics).
A signal is considered as a sum of pure sine partials, or harmonics.
Each partial evolves smoothly, and phase coherence is kept from one frame to the next, so that there is no need to implement overlap-add technique.
The source rawtable will be filled with the computed data
Parameters : Number: Sampling frequency (Hz)
RawTable: partial phase table (used internally. Needs only to be sent to keep phase coherence from one frame to another)
RawTable: partial power table (elements : 0-1)
RawTable: partial frequency table (elements : Hz)
RawTable: partial ending power table (elements : 0-1)
RawTable: partial ending frequency table (elements : Hz)
Example :

```
--- Flat pitch vocoder
-- From a digital track (a capella singer voice),
-- flattens the pitch to a fixed value
Include "MSDefine"

basefrq=220     -- flat pitch (Hz)
blksize=1024    -- analysis block size
nbpartials=44100/(2*basefrq) -- number of partials
nbsecs=10 -- number of seconds to compute

-- Get first staff, should be the digital track
score=FrontScore()
score.Preserve("Flat Pitch Vocoder")
staff=score.FirstStaff
dt=staff.DigitalTrack
dd=dt.NormalizedDigitalData

-- Init sinsynth
partial=NewRawTable(nbpartials)
ppower=NewRawTable(nbpartials)
pfreq=NewRawTable(nbpartials)
pepower=NewRawTable(nbpartials)
pefreq=NewRawTable(nbpartials)
for i=0,nbpartials-1 do
   ppower=0
   pepower=0
   pfreq=basefrq*(i+1)
   pefreq=basefrq*(i+1)
end

-- prepare data to compute
rawdata=NewRawTable(blksize)   -- input data
halfrawdata=NewRawTable(blksize/2) -- output data
spec=NewRawTable(blksize/2+1) -- spectrum

-- process all blocks
for i=1,nbsecs*44100,blksize/2 do
   dd.GetBlock(i,blksize,rawdata) -- Get data
   rawdata.FFT(0,WINDOWING_TRIANGLE,spec)  -- compute spectrum
   for j=0,nbpartials-1  do
      index=basefrq*(j+1) *blksize/44100 -- Get index of partial in spectrum
      -- compute partial power based upon spectrum power
      if index>1 and index<blksize/2-1  then
         pepower[j]=(spec[index]+spec[index-1]+spec[index+1])/(400+20*j)
      else
         pepower[j]=0
      end
   end
   halfrawdata.SineSynth(44100,partial,ppower,pfreq,pepower,pefreq) --generate block
   dd.SetBlock(halfrawdata,i) -- stores it
   for j=0,nbpartials-1 do ppower[j]=pepower[j] end -- copy for next step
end
```

Note : - All tables (except generated samples) must be of the same size. Each table element matches a partial (harmonics).
- During the process, partials evolves smoothly from their original (power, frequency) values to the "ending" values.

- power and frequency (and phase) tables are modified accordingly.

4.5.8. FXProcessor Object Type

General description of this object type : Digital FX processor settings
This object type can be encountered in several cases :
1- Default FX processor for an instrument in Instrument.FXProcessor (page 437)
These settings are overriden by the FX procesor objects that can be inserted in the staff.
2- FX Processor objects on a staff, browsed through Staff.FXProcessors (page 261) or Staff.FirstFXProcessor (page 261
Staff.LastFXProcessor (page 262)
In this case, FXProcessor is considered as an indexed object type, so fields like FXProcessor.Index (page 459), FXProcessor.Previous (page 459) and FXProcessor.Next (below) can be used.

Activity/Kind of resonator. See constants in MSDefine (page 100).mys:
---------- FX PROCESSORS ----------
--- Resonator
FX_INACTIVE_RESONATOR=0
FX_STOPPED_RESONATOR=1
FX_FIXED_RESONATOR=2
FX_LFO_RESONATOR=3
FX_WAH_RESONATOR=4
--- Distortion
FX_INACTIVE_DISTORTION=0
FX_STOPPED_DISTORTION=1
FX_REGULAR_DISTORTION=2
FX_OVERDRIVEN_DISTORTION=3
--- Flanger/Chorus
FX_INACTIVE_FLANGER=0
FX_STOPPED_FLANGER=1
FX_REGULAR_FLANGER=2
FX_CHORUSED_FLANGER=3
--- Equalizer
FX_INACTIVE_EQUALIZER=0
FX_STOPPED_EQUALIZER=1
FX_ACTIVE_EQUALIZER=2
--- Panning
FX_INACTIVE_PANNING=0
FX_STOPPED_PANNING=1
FX_FIXED_PANNING=2
FX_LFO_PANNING=3
--- Reverb
FX_INACTIVE_REVERB=0
FX_STOPPED_REVERB=1
FX_DELAY_REVERB=2
FX_MULTIPLE_DELAY_REVERB=3
FX_NATURAL_REVERB=4

4.5.8.1. FXProcessor values

4.5.8.1.1. Browsing FX Processors

IMPORTANT : These fields can only be used if this FXProcessor is an indexed object (like the FX processors that are inserted in a staff).
Trying to use these fields on a unique object (like the default FX processor of an instruments) stops the script with an error message.

FXProcessor.Next

Type : FXProcessor
Description : Next FX processor
If there is no next FX processor, value is nil.
Note :
a) FX processor are not sorted. They are linked in the order they have been inserted.
b) You can browse FX processors through Next and Previous fields, but you can also access an FX processor at a given index through, for example, Staff.FXProcessors (page 261)[index]

FXProcessor.Previous

Type : FXProcessor
Description : Previous FX processor
If there is no previous FX processor, value is nil.
Note :
a) FX processor are not sorted. They are linked in the order they have been inserted.
b) You can browse FX processors through Next and Previous fields, but you can also access an FX processor at a given index through, for example, Staff.FXProcessors (page 261)[index]

FXProcessor.Index

Type : Number
Description : Index of this FX processor
Example :

```
-- Displays the index of the 2nd FX processor of a staff, if exists
fx=myStaff.FXProcessors[2]
if fx~=nil then
   print(fx.Index) -- Displays "2"
end
```

4.5.8.1.2. FX Processor display settings

IMPORTANT : These fields can only be used if this FXProcessor is a displayed object (like the FX processors that are inserted in a staff).
Trying to use these fields on a non-displayed object (like the default FX processor of an instrument) returns nil

FXProcessor.Title

Type : String
Description : Title for this FX processor

FXProcessor.Comments

Type : String
Description : Comments for this FX processor

FXProcessor.IsSelected

Type : Boolean
Description : true if the FX processor is currently selected
Example :

```
-- Displays the title of all selected FX processor
fx=myStaff.FirstFXProcessor
while fx=nil do
   if fx.IsSelected then print("Selected : ",fx.Title)
   end
   fx=fx.Next
end
```

Note : "Selected" means the FX processor is framed in red, after having been clicked with the selection tool.
A FX processor that belongs to the selection range is not considered as selected.

FXProcessor.AreaLeft

Type : Number
Description : Left bound of the FX processor, in time position
Note : must be smaller that AreaRight

FXProcessor.AreaRight

Type : Number
Description : Right bound of the FX processor, in time position
Example :

```
-- Moves the first text of a staff at the beginning of bar 1 (in 4/4 time signature)
fx=myStaff.FirstFXProcessor
if fx=nil then
    fxWidth=fx.AreaRight-fx.AreaLeft
    fx.AreaLeft=DURATION_WHOLE
    fx.AreaRight=fx.AreaLeft+fxWidth
end
```

Note : must be greater that AreaLeft

FXProcessor.AreaLeftAbsolutePixel

Type : Number
Description : Left bound of the FX processor, in absolute pixel position

FXProcessor.AreaRightAbsolutePixel

Type : Number
Description : Righ bound of the FX processor, in absolute pixel position

FXProcessor.AreaTop

Type : Number
Description : Upper bound of the FX processor, in 72 DPI pixel relatively to the staff baseline (positive values to the bottom)
Note : must be smaller that AreaBottom

FXProcessor.AreaBottom

Type : Number
Description : Lower bound of the FX processor, in 72 DPI pixel relatively to the staff baseline (positive values to the bottom)
Example :

```
-- Moves the first FX processor of a staff just under the base line
fx=myStaff.FirstFXProcessor
if fx~=nil then
    fxtHeight=fx.AreaBottom-fx.AreaTop
    fx.AreaTop=1
    fx.AreaBottom=fx.AreaTop+fxHeight
end
```

Note : must be smaller that AreaTop

FXProcessor.LeftPixelOffset

Type : Number
Description : Pixel offset (72 DPI) to be added to the left bound of the FX processor frame to get its actual display location.
Because AreaLeft is provided in time position, some locations on staff (for example, the space over a clef or key change) could not be reached.
This pixel offset enables to make the FX processor be displayed at such locations.

FXProcessor.RightPixelOffset

Type : Number
Description : Pixel offset (72 DPI) to be added to the right bound of the FX processor frame to get its actual display location.
Because AreaRight is provided in time position, some locations on staff (for example, the space over a clef or key change) could not be reached.
This pixel offset enables to make the FX processor be displayed at such locations.
Example :

```
-- Makes the first FX processor of a staff to be displayed over the clef
fx=myStaff.FirstFXProcessor
if fx~=nil then
   fxWidth=fx.AreaRight-fx.AreaLeft
   fx.AreaLeft=0
   fx.AreaRight=fx.AreaLeft+fxWidth
   fx.LeftPixelOffset=-50
   fx.RightPixelOffset=-50
end
```

FXProcessor.RGBColor

Type : String
Description : FX processor title RGB Color

FXProcessor.Font

Type : String
Description : FX processor title font

FXProcessor.Face

Type : String
Description : FX Processor title face (See <TextStyle> object for an explanation of Face string)

FXProcessor.Size

Type : Number
Description : FX processor title size (72 dpi pixel)

FXProcessor.IsPrinted

Type : Boolean
Description : true if the FX processor is printed, false if hidden on printed page

4.5.8.1.3. Resonator settings

FXProcessor.ResonatorType

Type : Number
Description : Activity/Kind of resonator. See constants in MSDefine (page 100).mys

FXProcessor.FixedResonatorFrequency

Type : Number
Description : Fixed resonator frequency (Hz)
Range : 50 Hz to 4000 Hz
Note : Used only if FXProcessor.ResonatorType (above)==FX_FIXED_RESONATOR

FXProcessor.IsSmoothFixedResonatorFrequency

Type : Boolean
Description : true if fixed resonator frequency moves smoothly to its next value
Note : Used only if FXProcessor.ResonatorType (above)==FX_FIXED_RESONATOR

FXProcessor.FixedResonatorResonance

Type : Number
Description : Fixed resonator resonance (%)
Range : 0% to 100%
Note : Used only if FXProcessor.ResonatorType (above)==FX_FIXED_RESONATOR

FXProcessor.IsSmoothFixedResonatorResonance

Type : Boolean
Description : true if fixed resonator resonance moves smoothly to its next value
Note : Used only if FXProcessor.ResonatorType (page 461)==FX_FIXED_RESONATOR

FXProcessor.FixedResonatorGain

Type : Number
Description : Fixed resonator output gain (%)
Range : 0% to 200%
Note : Used only if FXProcessor.ResonatorType (page 461)==FX_FIXED_RESONATOR

FXProcessor.LFOResonatorStartFrequency

Type : Number
Description : LFO resonator start frequency (Hz)
Range : 50 Hz to 4000 Hz
Note : Used only if FXProcessor.ResonatorType (page 461)==FX_LFO_RESONATOR

FXProcessor.LFOResonatorMinFrequency

Type : Number
Description : LFO resonator minimum frequency (Hz)
Range : 50 Hz to 4000 Hz
Note : Used only if FXProcessor.ResonatorType (page 461)==FX_LFO_RESONATOR

FXProcessor.LFOResonatorMaxFrequency

Type : Number
Description : LFO resonator maximum frequency (Hz)
Range : 50 Hz to 4000 Hz
Note : Used only if FXProcessor.ResonatorType (page 461)==FX_LFO_RESONATOR

FXProcessor.LFOResonatorPeriod

Type : Number
Description : LFO resonator frequency oscillation Period (ms)
Range : 1 ms to 5000 ms
Note : Used only if FXProcessor.ResonatorType (page 461)==FX_LFO_RESONATOR

FXProcessor.LFOResonatorDirection

Type : Boolean
Description : true if LFO resonator frequency starts in direction of high frequencies, false if in direction of low frequencies
Note : Used only if FXProcessor.ResonatorType (page 461)==FX_LFO_RESONATOR

FXProcessor.LFOResonatorResonance

Type : Number
Description : LFO resonator resonance (%)
Range : 0% to 100%
Note : Used only if FXProcessor.ResonatorType (page 461)==FX_LFO_RESONATOR

FXProcessor.IsSmoothLFOResonatorResonance

Type : Boolean
Description : true if LFO resonator resonance moves smoothly to its next value
Note : Used only if FXProcessor.ResonatorType (page 461)==FX_LFO_RESONATOR

FXProcessor.LFOResonatorGain

Type : Number
Description : LFO resonator output gain (%)
Range : 0% to 200%
Note : Used only if FXProcessor.ResonatorType (page 461)==FX_FIXED_RESONATOR

FXProcessor.WahResonatorMinFrequency

Type : Number
Description : Wah resonator minimum frequency (Hz)
Range : 50 Hz to 4000 Hz
Note : Used only if FXProcessor.ResonatorType (page 461)==FX_WAH_RESONATOR

FXProcessor.WahResonatorMaxFrequency

Type : Number
Description : Wah resonator maximum frequency (Hz)
Range : 50 Hz to 4000 Hz
Note : Used only if FXProcessor.ResonatorType (page 461)==FX_WAH_RESONATOR

FXProcessor.WahResonatorMinVolume

Type : Number
Description : Wah resonator minimum Volume (%)
Range : 0% to 100%
Note : Used only if FXProcessor.ResonatorType (page 461)==FX_WAH_RESONATOR

FXProcessor.WahResonatorMaxVolume

Type : Number
Description : Wah resonator maximum Volume (%)
Range : 0% to 100%
Note : Used only if FXProcessor.ResonatorType (page 461)==FX_WAH_RESONATOR

FXProcessor.WahResonatorResonance

Type : Number
Description : Wah resonator resonance (%)
Range : 0% to 100%
Note : Used only if FXProcessor.ResonatorType (page 461)==FX_WAH_RESONATOR

FXProcessor.IsSmoothWahResonatorResonance

Type : Boolean
Description : true if Wah resonator resonance moves smoothly to its next value
Note : Used only if FXProcessor.ResonatorType (page 461)==FX_WAH_RESONATOR

FXProcessor.WahResonatorGain

Type : Number
Description : Wah resonator output gain (%)
Range : 0% to 200%
Note : Used only if FXProcessor.ResonatorType (page 461)==FX_WAH_RESONATOR

4.5.8.1.4. Distortion/Overdrive settings

FXProcessor.DistortionType

Type : Number
Description : Activity/Kind of distortion/overdrive. See constants in MSDefine (page 100).mys

FXProcessor.DistortionPower

Type : Number
Description : Regular distorsion Power (%)
Range : 0% to 100%
Note : Used only if FXProcessor.DistortionType (above)==FX_REGULAR_DISTORTION

FXProcessor.DistortionTone

Type : Number
Description : Regular distorsion Tone (bass/treble) (%)
0 means loud tone, 100 means treble tone
Range : 0% to 100%
Note : Used only if FXProcessor.DistortionType (above)==FX_REGULAR_DISTORTION

FXProcessor.DistortionGain

Type : Number
Description : Regular distorsion output gain (%)
Range : 0% to 100%
Note : Used only if FXProcessor.DistortionType (page 463)==FX_REGULAR_DISTORTION

FXProcessor.OverdrivePower

Type : Number
Description : Overdrive Power (%)
Range : 0% to 100%
Note : Used only if FXProcessor.DistortionType (page 463)==FX_OVERDRIVEN_DISTORTION

FXProcessor.OverdriveTone

Type : Number
Description : Overdrive Tone (bass/treble) (%)
0 means loud tone, 100 means treble tone
Range : 0% to 100%
Note : Used only if FXProcessor.DistortionType (page 463)==FX_OVERDRIVEN_DISTORTION

FXProcessor.OverdriveGain

Type : Number
Description : Overdrive output gain (%)
Range : 0% to 100%
Note : Used only if FXProcessor.DistortionType (page 463)==FX_OVERDRIVEN_DISTORTION

4.5.8.1.5. Flanger/Chorus settings

FXProcessor.FlangerType

Type : Number
Description : Activity/Kind of flanger/chorus. See constants in MSDefine (page 100).mys
Note : Flanger and chorus are processed exactly the same way.
The only change is the range of oscillation period.
Setting either FX_REGULAR_FLANGER or FX_CHORUSED_FLANGER type won't change the resulting sound, only the range of available values when this FX Processor will be edited by user

FXProcessor.FlangerMinDelay

Type : Number
Description : Flanger/Chorus minimum delay (from 0 to 11000 microseconds)
Range : 0 microsecond to 11000 microseconds

FXProcessor.FlangerMaxDelay

Type : Number
Description : Flanger/Chorus maximum delay (from 0 to 11000 microseconds)
Range : 0 microsecond to 11000 microseconds

FXProcessor.FlangerPeriod

Type : Number
Description : Flanger/Chorus oscillation period (ms)
Range : 0 ms to 5000 ms
Note :
For regular flanger (FXProcessor.FlangerType (above)=FX_REGULAR_FLANGER, ranges from 0 to 5000
For chorus (FXProcessor.FlangerType (above)=FX_CHORUSED_FLANGER, ranges from 0 to 200

FXProcessor.FlangerPower

Type : Number
Description : Flanger Power (%)
Range : 0% to 100%

4.5.8.1.6. Equalizer settings

FXProcessor.EqualizerType

Type : Number
Description : Activity/Kind of equalizer. See constants in MSDefine (page 100).mys

FXProcessor.EqualizerBass

Type : Number
Description : For simple bass/treble equalizer: Bass enhancement (%)
Range : 0% to 100%

FXProcessor.EqualizerTreble

Type : Number
Description : For simple bass/treble equalizer: Treble enhancement (%)
Range : 0% to 100%

FXProcessor.IsSmoothEqualizerBassTreble

Type : Boolean
Description : true if simple bass/treble equalizer : move smoothly to its next value

FXProcessor.EqualizerGain

Type : Number
Description : For simple bass/treble equalizer: Equalizer output gain (%)
Range : 0% to 200%

FXProcessor.EqualizerLimiter

Type : Number
Description : For Simple bass/treble equalizer: Equalizer limiter gain (%)
Sound data over this limit are clipped. Provides as well a way to avoid too loud sound as a basic clipped-distortion if this setting is too low
Range : 0% to 200%

FXProcessor.IsSmoothEqualizerGain

Type : Boolean
Description : true if output gain moves smoothly to its next value. For complete Multi-band equalizer, all the levels (including gain) move smoothly

FXProcessor.MultibandEQGain

Type : Number
Description : Master gain (in dB) for multi-band equalizer. From -48 to 48

FXProcessor.MultibandEQNumberOfBands

Type : Number
Description : Number of bands in the multi-band equalizer. This value is read-only.

FXProcessor.MultibandEQFrequencies

Type : Array
Description : Center frequency of each band in the multi-band equalizer (in Hz). This array is read-only.

FXProcessor.MultibandEQLevels

Type : Array
Description : band level (in dB) for each band of the multi-band equalizer. From -24 to 24
Example :

```
Include "MSDefine"
-- Add an Fx (quick code, would need more tests)
staff=FrontScore().FirstStaff
 fx=staff.AddFXProcessor(0)
-- Sets it as a multiband equalizer
fx.EqualizerType=FX_ACTIVE_MULTIBAND_EQ
-- Set master gain (in dB)
fx.MultibandEQGain=-4
-- Set all the band levels in dB.
-- Only lets frequencies from 350 to 3500 Hz pass through to simulate phone sound
fx.MultibandEQLevels={-24,-24,-24,-24,13,20,24,-24,-24,-24,-24}
```

Note : This array can only be set by setting it in one stroke. For instance, fx.MultibandEQLevels=levelArray is OK, but fx.MultibandEQLevels[1]=2 isn't

4.5.8.1.7. Panning settings

FXProcessor.PanningType

Type : Number
Description : Activity/Kind of panning. See constants in MSDefine (page 100).mys

FXProcessor.FixedPanningValue

Type : Number
Description : Fixed panning value from -100 (left) to 100 (right).
This value is added to the regular panning (stereo) position of the instrument.
Range : -100% (left) to 100% (right)

FXProcessor.FixedFadingValue

Type : Number
Description : Fixed panning value from -100 (left) to 100 (right).
This value is added to the regular fading (front/rear) position of the instrument.

FXProcessor.IsSmoothFixedPanning

Type : Boolean
Description : true if fixed panning position moves smoothly to its next value

FXProcessor.LFOPanningStartValue

Type : Number
Description : LFO panning starting value, from -100 (left) to 100 (right). In 2D, -100=left, 0=front, 100=right, -200=200=rear
Range : -100% (left) to 100% (right)

FXProcessor.LFOPanningMinValue

Type : Number
Description : FO stereo minimum value, from -100 (left) to 100 (right)
Range : -100% (left) to 100% (right)

FXProcessor.LFOPanningMaxValue

Type : Number
Description : FO stereo maximum value, from -100 (left) to 100 (right)
Range : -100% (left) to 100% (right)

FXProcessor.LFOFaderMinValue

Type : Number
Description : FO fader minimum value, from -100 (front) to 100 (rear)
Range : -100% (left) to 100% (right)

FXProcessor.LFOFaderMaxValue

Type : Number
Description : FO fader maximum value, from -100 (front) to 100 (rear)
Range : -100% (left) to 100% (right)

FXProcessor.LFOPanningPeriod

Type : Number
Description : LFO panning value oscillation Period (ms)
Range : 1 ms to 5000 ms

FXProcessor.LFOPanningDirection

Type : Boolean
Description : true if LFO panning value starts in direction of max value (right), false if in direction of min value (left)

4.5.8.1.8. Delay/Reverb settings

FXProcessor.ReverbType

Type : Number
Description : Activity/Kind of Delay/Reverb. See constants in MSDefine (page 100).mys

FXProcessor.DelayTime

Type : Number
Description : Simple delay : time before the unique echo (ms)
Range : 10 ms to 500 ms
Note : only used if FXProcessor.ReverbType (above)==FX_DELAY_REVERB

FXProcessor.DelayPower

Type : Number
Description : Simple delay power (%).
Range : 0% to 100%

FXProcessor.IsBouncedDelay

Type : Boolean
Description : true if simple delay sounds at the opposite panning of original sound.

FXProcessor.MultipleDelayTime

Type : Number
Description : Multiple delay : time between each echo (ms)
Range : 10 ms to 500 ms
Note : only used if FXProcessor.ReverbType (above)==FX_MULTIPLE_DELAY_REVERB

FXProcessor.MultipleDelayPower

Type : Number
Description : Multiple delay power (%).
Range : 0% to 100%

FXProcessor.IsBouncedMultipleDelay

Type : Boolean
Description : true if multiple delay bounces from left to right.

FXProcessor.ReverbTime

Type : Number
Description : Realistic reverb : total time before end of reverberation (ms)
Range : 200 ms to 5000 ms
Note : only used if FXProcessor.ReverbType (above)==FX_NATURAL_REVERB

FXProcessor.ReverbPower

Type : Number
Description : Realistic reverb power (%).
Range : 0% to 100%

FXProcessor.IsBouncedReverb

Type : Boolean
Description : true if Realistic reverb bounces from left to right.

4.5.8.2. FXProcessor methods

4.5.8.2.1. Deleting an indexed FX processor

FXProcessor.Delete(...)

Description : Deletes the FX processor.
Parameters : None
Return value : true if successfully deleted
Example :

```
-- Deletes all FX processors in a staff, starting from the end to avoid index problems
fx=myStaff.LastFXProcessor
while fx~=nil do
    -- Once FX processor has been deleted, Previous member could not be retrieved
    previousFx=fx.Previous
    fx.Delete()
    fx=previousFx
end
```

Note : IMPORTANT : This method can only be used if this FXProcessor is an indexed object (like the FX processors that are inserted in a staff).
Trying to use this method on a unique embedded object (like the default FX processor of an instrument) stops the script with an error message.

4.5.8.2.2. Editing an FX processor

FXProcessor.Edit(...)

Description : Opens a window that enables the user to apply changes to an FX processor.
Parameters : None
Return value : true if successfully edited
Example :

```
-- Creates a new FX processor in a staff and edits it
fx=myStaff.AddFXProcessor()
if fx~=nil then
    fx.Edit()
end
```

Note : If the edited FXProcessor is a non-displayable object (like the default FX processor of an instrument), icon, name and comment won't appear in the edit window.

4.5.8.2.3. Managing FX processor icon

a- IMPORTANT : These methods can only be used if this FXProcessor is a displayed object (like the FX processors that are inserted in a staff).
Trying to use these methods on a non-displayed object (like the default FX processor of an instrument) stops the script with an error message
b- FX processor icons are managed as indexed color icons.
There can be up to 256 color indexes in such an icon, each color could have its own RGB components. Pure white ("FFFFFF") is considered as transparent.
Icon itself is a 32x32 array of color indexes.

FXProcessor.SelectIcon(...)

Description : Enables the user to select the FX processor icon in the list of predefined icons
Parameters : None
Return value : None

FXProcessor.EditIcon(...)

Description : Enables the user to edit the current icon of the FX processor
Parameters : None
Return value : None

FXProcessor.GetIconRGBColor(...)

Description : Gets the RGB components of the color at the given index in the FX processor icon
Parameters : Number: icon color index (1-256)
Return value : String: RGB component for this color index
If the RGB components for this icon have never been set (unused color index), returns nil.
Note : Returns nil if the icon is not editable (FX processor belongs to an instrument)

FXProcessor.SetIconRGBColor(...)

Description : Sets the RGB components of the color at the given index in the FX processor icon
Parameters : Number: icon color index (1-256)
String: RGB color to set or nil (or omitted) if the color index is not used anymore
Be careful not to set a color index as unused if still used in the icon. It could result in unexpected color display.
Return value : Boolean: true if OK

FXProcessor.GetIconPixel(...)

Description : Gets the icon color index at the give x,y position
Parameters : Number: X coordinate (1-32) to get the color index from
Number: Y coordinate (1-32) to get the color index from
Return value : Number: Color index at this location (1-256) or nil if error
Note : Returns nil if the icon is not editable (FX processor belongs to an instrument)

FXProcessor.SetIconPixel(...)

Description : Sets the icon color index at the give x,y position
Parameters : Number: X coordinate (1-32) to set the color index to
Number: Y coordinate (1-32) to get the color index to
Number: Color index (1-256) to set. You are responsible for ensuring this color index is marked as used in the icon color table.
Return value : Boolean: true if successful, false if error

FXProcessor.EraseIcon(...)

Description : Erases the whole icon
After being erased, only the first color remains used in the icon color table, and its RGB components are set to pure white (transparent)
All pixels in the icon are set to the first color index (1), so that the whole icon becomes transparent.
Parameters : None
Return value : Boolean: true if successful, false if error
Example :

```
-- Sets the first FX processor icon as a blue square crossed in red
fx=myStaff.FXProcessors[1]
if fx~=nil then
   fx.EraseIcon()
   fx.SetIconRGBColor(1,"FFFFFF") -- index 1 : transparent color
   fx.SetIconRGBColor(2,"0000FF") -- index 2 : blue color
   fx.SetIconRGBColor(3,"FF0000") -- index 3 : red color

   for i=1,32 do
      fx.SetIconPixel(i,i,3) -- first diagonal: red
      fx.SetIconPixel(33-i,i,3) -- 2nd diagonal: red
      fx.SetIconPixel(i,1,2) -- up bound : blue
      fx.SetIconPixel(1,i,2) -- left bound : blue
      fx.SetIconPixel(32,i,2) -- right bound : blue
      fx.SetIconPixel(i,32,2) -- bottom bound : blue
   end
end
```

4.5.9. Curve Object Type

General description of this object type : Parameter curve attached to a staff.

4.5.9.1. Curve values

4.5.9.1.1. Browsing Curves

Curve.Next

Type : Curve
Description : Next Curve
If there is no next Curve, value is nil.

Curve.Previous

Type : Curve
Description : Previous Curve
If there is no previous Curve, value is nil.

4.5.9.1.2. Curve settings

Curve.Type

Type : Number
Description : Curve type
Here is the list of available curve types, followed by their internal value.
It is recommended to use one of these constants. Using directly a non-listed 4-character identifier enables to manage custom curves, that are neither interpreted, nor available for the user to edit.

- VOLUME_CURVE "Volu"
- PANNING_CURVE "Ster"
- FREQUENCY_CURVE "Freq"
- RESONATOR_RESONANCE_CURVE "Resr"
- RESONATOR_FREQUENCY_CURVE "Resf"
- OVERDRIVE_POWER_CURVE "Ovep"
- OVERDRIVE_TONE_CURVE "Ovet"
- FLANGER_POWER_CURVE "Flap"
- EQUALIZER_TREBLE_CURVE "Equa"
- EQUALIZER_BASS_CURVE "Equg"
- REVERBERATION_POWER_CURVE "Revp"
- MIDI1_CURVE "Mid1"
- MIDI2_CURVE "Mid2"
- MIDI3_CURVE "Mid3"
- MIDI4_CURVE "Mid4"
- MIDI5_CURVE "Mid5"

Example :

```
-- Displays types of all Curves in the first staff
myScore=FrontScore()
if myScore==nil then
   print("Please load a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in this score")
   else
      curve=myStaff.FirstCurve
      while curve~=nil do
         print("curve : ",curve.Type)
         curve=curve.Next
      end
   end
end
```

Curve.MinValue

Type : Number
Description : Minimum value for curve data
Note : When setting this value, no scaling is performed. Curve values that fall below the new minimum are set to the minimum.
You are responsible for keeping the minimum value smaller than the maximum.

Curve.MaxValue

Type : Number
Description : Maximum value for curve data
Note : When setting this value, no scaling is performed. Curve values that fall above the new maximum are set to the maximum.
You are responsible for keeping the minimum value smaller than the maximum.

Curve.IsDiscrete

Type : Boolean
Description : true if the curve has discrete values, i.e. there is no smooth variation from one value to another

Curve.Color

Type : Number
Description : Color index for displaying this curve

Curve.IsActive

Type : Boolean
Description : true if the curve is active, i.e. its data are taken into account when playing.

Curve.IsGlobal

Type : Boolean
Description : true if the curve data are applied to the whole score, false if applied to the embedding staff only.
Note : Only processed when Curve.IsActive (above) is true, ignored otherwise

Curve.MidiCommand

Type : String
Description : Midi command to be sent when playing.
Note : Only processed when Curve.IsActive (above) is true, the output device for the staff is a MIDI output, and the curve is one of the 5 available MIDI curves. Ignored otherwise.
This string is made of an even number of characters, describing the MIDI command to be sent, in hexadecimal.
If the character is from '0' to '9' or from 'A' to 'F', the matching fixed nybble (half-byte) is sent as is.
If the character is 'n', the channel number (4-bits) is sent.
If the character is 'l', the lowest part of the value LSB (bits 1-4 of the 14-bits value) is sent.
If the character couple is 'll', the whole value LSB (less significant 7 bits, i.e. bits 1-7 of the 14-bits value) is sent.
If the character is 'm', the lowest part of the value MSB (bits 8-11 of the 14-bits value) is sent.
If the character couple is 'mm', the whole value MSB (most significant 7 bits, i.e. bits 8-14 of the 14-bits value) is sent.

For example, if the string is "Bn8mll", on a staff whose channel is 1, when the program gets a curve value of 1410, the following 3-bytes command will be sent to MIDI device :
B0 (channels are numbered 0 to 15 in internal MIDI packets, so channel 1 matches a 0 value)
8B (value 1410 is notated 00010110000011 in binary. Bits 8-11 are then 1011, i.e. 'B' in hexadecimal)
03 (value 1410 is notated 00010110000011 in binary. Bits 1-7 are then 0000011, i.e. '03' in hexadecimal)

4.5.9.1.3. Managing curve data

Curve.NumberOfValues

Type : Number
Description : Number of values in curve data area. Access to the curve values is made through the matching methods of the Curve object.

4.5.9.2. Curve methods

4.5.9.2.1. Deleting a Curve

Curve.Delete(...)

Description : Deletes the whole Curve
Parameters : None
Return value : true if successfully deleted
Example :

```
-- Deletes all Curves in the first staff
myScore=FrontScore()
if myScore==nil then
   print("Please load a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in this score")
   else
      while myStaff.FirstCurve do
         myStaff.FirstCurve.Delete()
      end
   end
end
```

4.5.9.2.2. Managing curve values

Curve.GetValue(...)

Description : Gets the curve value at a given time position. If there is no curve point at this position, interpolates between the previous and next point.
Parameters : time position
Return value : Number : curve value at this time position
or nil if there is no curve data at this position
Example :

```
Include "MSDefine"
-- Displays volume curve value at the end of bar#1
myScore=FrontScore()
if myScore==nil then
   print("Please load a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in this score")
   else
      volumeCurve=myStaff.Curves[VOLUME_CURVE]
      if volumeCurve then
         value=volumeCurve.GetValue(DURATION_WHOLE)
         if value then
            print("Volume curve value at the end of bar #1 is",value)
         else
            print("Volume curve value is not defined at the end of bar #1")
         end
      else
         print("There is no volume curve in the 1st staff")
      end
   end
end
```

Curve.SetValue(...)

Description : Sets the curve value at a given time position. If there is already curve point at this position, changes its value by the new one, otherwise creates a new point
Parameters : Number : time position
Number : value (can be changed to be kept in the available range of value)
Optional:
Boolean : true if this point is the last of a curve section (does not interpolate with thext point). Default=false
Return value : None
Example :

```
Include "MSDefine"
-- Sets a volume fade-in across first bar
myScore=FrontScore()
if myScore==nil then
   print("Please load a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in this score")
   else
      volumeCurve=myStaff.Curves[VOLUME_CURVE]
      if volumeCurve==nil then
         volumeCurve=myStaff.AddCurve(VOLUME_CURVE)
      end
      if volumeCurve then
         -- Deletes previous values in this bar
         volumeCurve.DeleteValue(0,DURATION_WHOLE,true)
         -- Inserts first point
         volumeCurve.SetValue(0,0,false)
         -- Inserts second point and ends section
         volumeCurve.SetValue(DURATION_WHOLE,100,true)
      else
         print("Impossible to create volume curve in 1st staff")
      end
   end
end
```

Curve.DeleteValue(...)

Description : Deletes one or several values in a curve, according to a time position or a time range.
Parameters : There are two way of using this method :
a) You need to delete a single curve point, in which case you provide only the exact time position of the point to delete, and parameters are :
Number: time position of value to delete
If there is no value at this exact time position, nothing is performed
b) You need to delete a range of curve points between two time positions,
Number: time range beginning position
Number: time range ending position
Optional Boolean: true if the curve stops at the beginning of the range, and restarts at the end of the range. Otherwise, interpolation continues over the range. Default: true.
When a range of values is deleted, two points are added, one at the exact beginning of the range, and the other one at the end of the range.
Return value : None
Example :

```
Include "MSDefine"
-- Deletes volume curve points at first bar
myScore=FrontScore()
if myScore==nil then
   print("Please load a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in this score")
   else
      volumeCurve=myStaff.Curves[VOLUME_CURVE]
      if volumeCurve then
         volumeCurve.DeleteValue(0,DURATION_WHOLE,true)
      end
   end
end
```

Curve.GetIndexedValue(...)

Description : Gets the curve value at a given point.
Parameters : point index, from 1 to Curve.NumberOfValues (page 472)
Return value : Number : value for this point
Number : time position of this point
Boolean: true if this points is a last of a curve section
Example :

```
Include "MSDefine"
-- Displays all points of the volume curve
-- (couple : value - time position)
myScore=FrontScore()
if myScore==nil then
    print("Please load a score first")
else
    myStaff=myScore.FirstStaff
    if myStaff==nil then
        print("There is no staff in this score")
    else
        volumeCurve=myStaff.Curves[VOLUME_CURVE]
        if volumeCurve then
            for i=1,volumeCurve.NumberOfValues do
                value,time,isBreak=volumeCurve.GetIndexedValue(i)
                if isBreak==true then str="/" else str=" " end
                print(i,":",value,time,str)
            end
        else
            print("There is no volume curve in the 1st staff")
        end
    end
end
```

Curve.GetPreviousValueIndex(...)

Description : Finds the index of the nearest curve point whose time position is smaller or equal to the given time position.
Parameters : Number : time position
Return value : point index, from 1 to Curve.NumberOfValues (page 472) to be used in Curve.GetIndexedValue(. ove).
if the given time position is before the first curve point, returns nil.

4.5.10. VSVoice Object Type

General description of this object type : This kind of object include all the Virtual Singer settings related to a staff. When you change a Virtual Singer setting, the staff is tagged to be recomputed next time the music will be launched, so, changing a setting while music is playing won't have any immediate effect.

4.5.10.1. VSVoice values

4.5.10.1.1. Placement on stage

VSVoice.PanningPlacement

Type : Number
Description : Singer panning from 0(left) to 127(right)

VSVoice.VolumePlacement

Type : Number
Description : Singer volume from 0(backstage) to 127(frontmost)

4.5.10.1.2. Attribute

VSVoice.IsActive

Type : Boolean
Description : Return true if a Virtual Singer voice is related to the staff. You can set this value to activate/ deactivate the Virtual Singer voice related to the staff. You can't activate a Virtual Singer voice if the staff don't allow this, for example if the staff don't have lyrics. So you ought to check first .
Example :

```
--- We got the front score
myScore=FrontScore()
if(myScore~=nil) then
   -- We extract the first staff of this score
   myStaff=myScore.FirstStaff
   if(myStaff~=nil) then
      if(myStaff.VirtualSingerVoice.IsPossible==true) then
         if(myStaff.VirtualSingerVoice.IsActive==true) then
               Message("A Virtual Singer voice is related to the staff :"..myStaff.Title.." and is active.")
         else
               Message("A Virtual Singer voice is related to the staff :"..myStaff.Title.." and is inactive.")
         end
      else
         Message("You can't relate a Virtual Singer voice to this staff.")
      end
   end
else
   Message("Please load a score.")
end
```

VSVoice.IsPossible R

Type : Boolean
Description : Return true if a Virtual Singer voice can be related to the staff. Typically it is a staff with lyrics.

VSVoice.Language

Type : Number
Description :
Deprecated, kept for historical/compatibility reasons. Use LanguageCode instead
The language related to the singer, defined as this :
- SINGER_LANGUAGE_UNKNOWN=0
- SINGER_LANGUAGE_NORTH_FRENCH=1
- SINGER_LANGUAGE_SOUTH_FRENCH=2
- SINGER_LANGUAGE_UK_ENGLISH=3
- SINGER_LANGUAGE_US_ENGLISH=4
- SINGER_LANGUAGE_LATIN=5
- SINGER_LANGUAGE_SPANISH=6
- SINGER_LANGUAGE_ITALIAN=7
- SINGER_LANGUAGE_FINISH=8
- SINGER_LANGUAGE_JAPANESE=9
- SINGER_LANGUAGE_GERMAN=10
- SINGER_LANGUAGE_OCCITAN=11

You can get or set this value even a Virtual Singer voice is not related to the staff.
If the language is set to SINGER_LANGUAGE_UNKNOWN the lyrics will be analysed next time the music will be launched and the language switched to the more probable language.

VSVoice.LanguageCode

Type : String
Description : Language code related to the singer, defined as this :
<3-letters ISO 639 code>-<variation number from 0 to 2>
For instance, fre-1 represents French (Southern variation)
Extension to the ISO 639 standard:
"enu-0" represents US English (instead of eng-1)
Note : when several 3-letters code are available for a same language, the bibliographic version is used
Ref: http://www.loc.gov/standards/iso639-2/php/code_list.php

4.5.10.1.3. General Setting

VSVoice.OctaveShift

Type : Number
Description : Octave shift of the voice from -5 to 5.

VSVoice.SemiToneShift

Type : Number
Description : Pitch shift of the voice from -11 to 11 in semitone.

VSVoice.ChoirSize

Type : Number
Description : Choir size from 1 to 20. This emulate a choir from a single staff. Each single note will be played with 1 to 20 different voices. Each voice generated can be detuned using the VSVoice.ChoirDetuning (page 48(pitch value.

4.5.10.1.4. Timbre

Timbre is what makes the difference between a singer and another one. It can be considered as the voice "fingerprint" because it depends on the vocal tract size and shape.

VSVoice.Formant1Ratio

Type : Number
Description : Description of formant 1. From 50 to 180

VSVoice.Formant2Ratio

Type : Number
Description : Description of formant 2. From 50 to 180

VSVoice.Formant3Ratio

Type : Number
Description : Description of formant 3. From 50 to 180

VSVoice.Formant4Ratio

Type : Number
Description : Description of formant 4. From 50 to 180

VSVoice.Formant5And6Ratio

Type : Number
Description : Description of formant 5 and 6. From 50 to 180

VSVoice.TrebleBassRatio

Type : Number
Description : Description of ratio between treble and bass. From -70 to 70.

4.5.10.1.5. Glotal source

VSVoice.GlotalSourceOpenRatio

Type : Number
Description : Glottal source settings. The open ratio enables to select the ratio between vocal folds open and closed time. From 10 to 90.

VSVoice.GlotalSourceSaturation

Type : Number
Description : Saturation enables to apply an huskiness effect on voice. From 0 to 100.

4.5.10.1.6. Interpretation

VSVoice.TimeShift

Type : Number
Description : This is the maximal imprecision of the singer (in millisecond) who do not sing the note exactly when it should be. From 0 to 199.

VSVoice.MinimalRestDuration

Type : Number
Description : In order to avoid a too jerky voice, little rests can be ignored. This setting enables to define the rest duration (in hundredth of second) from which the singer actually stops. Shorter rests will be ignored. From 0 to 199.

VSVoice.SingingFormantRiseTime

Type : Number
Description : Singing formant rise time (100th of second).

VSVoice.SingingFormantSharpness

Type : Number
Description : Singing formant sharpness (signed percentage). -100%=soft, 0%=normal, 100%=sharp

VSVoice.SingingFormantPower

Type : Number
Description : Singing formant power (percentage).

4.5.10.1.7. Volume

VSVoice.Amplification

Type : Number
Description : This is the amplification of the voice. From 50 to 399.

VSVoice.VelocityRatio

Type : Number
Description : Notes in the sung staff include velocity information (loudness). This velocity ratio defines in which proportion the velocity value will be taken into account. From 0 to 100.

VSVoice.JitterMaximum

Type : Number
Description : Jitter is a random variation around the actual value. It simulates unconscious volume variations while a note is sung.
You can define here the maximum of the jitter. From 0 to 19.

VSVoice.JitterSpeed

Type : Number
Description : Jitter is a random variation around the actual value. It simulates unconscious volume variations while a note is sung.
You can define here the speed of the jitter. From 0 to 100.

VSVoice.VolumeEnvelopeMinimum

Type : Number
Description : This is a part of the volume envelope. Define the minimal volume of the beginning and the end of the enveloppe. From 0 to 100.

VSVoice.VolumeEnvelopeConsonantAttack

Type : Number
Description : This is a part of the volume envelope. Define the consonant attack.

VSVoice.VolumeEnvelopeConsonantEnd

Type : Number
Description : This is a part of the volume envelope. Define the end of the consonant and the begin of the vowel attack.

VSVoice.VolumeEnvelopeVowelAttack

Type : Number
Description : This is a part of the volume envelope. Define the vowel attack duration.

VSVoice.VolumeEnvelopeVowelMaximum

Type : Number
Description : This is a part of the volume envelope. Define the maximum level of the vowel.

VSVoice.VolumeEnvelopeVowelDecay

Type : Number
Description : This is a part of the volume envelope. Define the decay of the vowel and the beginning of the sustain.

VSVoice.VolumeEnvelopeVowelSustain

Type : Number
Description : This is a part of the volume envelope. Define the end of the vowel and the beginning of the release.

VSVoice.VolumeEnvelopeRelease

Type : Number
Description : This is a part of the volume envelope. This is the release.

4.5.10.1.8. Pitch Settings

VSVoice.PitchAttack

Type : Number
Description : A singer can start to sing a note slightly more low-pitchged than its written pitch. It can be adjusted though the pitch shift setting, in tone. A negative value (usually the case) makes the syllable begin to be sung more low-pitched than the written note. From -3.6 to 3.6.

VSVoice.ChoirDetuning

Type : Number
Description : When the singer voice is defined as a choir of several singers, (if VSVoice.ChoirSize (page 477) greater than 1), this value gives the maximal imprecision (in pitch) of each choir member.
A big value amplifies the choir effect but makes the pitch less accurate. From 0 to 100.

VSVoice.DefaultFragments

Type : String
Description : Fragments to be played when there are no lyrics to be sung for a note. An empty string means silent.
Fragments have to be written as a list, separated by "+". For instance, "TL2+SQ+AF" for a "Ta" sound.
The actual spelling of the syllable, for pop-up menu display purposes only, can be indicated at the beginning, between parenthesis. For instance "(French 'ta')TL2+SQ+AF"

VSVoice.JitterF0Max

Type : Number
Description : It controls the jitter, in its common meaning, i.e. a quick pitch variation. From 0 to 100.

VSVoice.JitterF0Speed

Type : Number
Description : It controls the jitter, in its common meaning, i.e. a quick pitch variation. From 0 to 25.

VSVoice.DriftF0UpwardRatio

Type : Number
Description : It is the way of moving from one note pitch to another : due to its organic nature, the vocal tract changes from one position to annother through a distortion. From 0 to 100.

VSVoice.DriftF0DownwardRatio

Type : Number
Description : From 0 to 100.

VSVoice.DriftF0MaxTime

Type : Number
Description : From 0 to 200.

VSVoice.DriftF0MinTime

Type : Number
Description : From 0 to 2000.

4.5.10.1.9. Vibrato

Vibrato applies to the fundamental pitch. It is a slow and regular variation of this frequency.
Remark : not to be confused with jitter, which is a quick and random variation.

VSVoice.VibratoMinimalFrequency

Type : Number
Description : Minimal frequency is the base frequency of the vibrato oscillation (in 10th of Hertz). A common vibrato is between 50 and 70 tenth of Hertz (between 5 and 7 Hz). From 0 to 100.

VSVoice.VibratoFrequencyRatio

Type : Number
Description : Frequency ratio enables to speed up vibrato when sung note is more high-pitched. Its value is the number of semitone above A4 (440 Hz) needed to speed up the vibrato frequency by 1 Hz.
From 0 to 120.

VSVoice.VibratoDelayBefore

Type : Number
Description : Delay is the delay before the vibrato starts (in 100th of second).
From 0 to 199.

VSVoice.VibratoRiseTime

Type : Number
Description : During rise time (100th of second), vibrato power incresases smoothly before reaching its maximal value.
From 0 to 199.

VSVoice.VibratoPower

Type : Number
Description : Vibrato power is its maximum depth in 100th of tone.
From 0 to 99.

VSVoice.VibratoPowerFrequencyRatio

Type : Number
Description : Frequency ratio enables to increase the vibrato power when sung note is more high-pitched. Its value is the number of semitones above A4 (440 Hz) needed to increase vibrato power by one semitone.
From 0 to 120.

4.5.10.1.10. Advanced

A jitter on these parameters changes the voice timber while a note is sung. (When singing, the vocal tract and mouth shapes slightly change).

VSVoice.JitterF1Maximum

Type : Number
Description : From 0 to 399.

VSVoice.JitterF1Speed

Type : Number
Description : From 0 to 99.

VSVoice.JitterF2Maximum

Type : Number
Description : From 0 to 399.

VSVoice.JitterF2Speed

Type : Number
Description : From 0 to 99.

VSVoice.FragmentString

Type : String
Description : This string, encoded in ASCII format, contains all the phonetic fragments that have been modified or added by the user.
The data structure is opaque and you should not modify anything in this string. It is provided only for saving, loading, transferring,
preserving or restoring the whole fragment adjustments.
Please note that writing this value will erase all changes previously made in the considered VS voice fragments.

4.5.10.2. VSVoice methods

4.5.10.2.1. Setting voice language to the most probable one

VSVoice.SetDefaultLanguage(...)

Description : Analyze the lyrics content, find the language (by Bayesian filtering) that matches best these syllables, and set it into VSVoice.Language (page 477).
Parameters : None
Return value : None

4.5.11. VirtualInstrument Object Type

General description of this object type : Virtual instrument related to a staff

4.5.11.1. VirtualInstrument values

4.5.11.1.1. Performer: Finger sliding (left hand)

VirtualInstrument.SlideMinCellOffset

Type : Number
Description : Number of cells from which the finger will slide on the string
Note : Reading this member provides the default value for the instrument, regardless to the live changes that could have been made while the instrument is playing
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.LiveSlideMinCellOffset

Type : Number
Description : "Live" Number of cells from which the finger will slide on the string
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

VirtualInstrument.SlideVolume

Type : Number
Description : Volume of sliding noise on strings
Note : Reading this member provides the default value for the instrument, regardless to the live changes that could have been made while the instrument is playing
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.SlidePressure

Type : Number
Description : Finger pressure on string when sliding
Note : Reading this member provides the default value for the instrument, regardless to the live changes that could have been made while the instrument is playing
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.LiveSlideVolume

Type : Number
Description : "Live" sliding noise volume
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

VirtualInstrument.LiveSlidePressure

Type : Number
Description : "Live" Finger pressure on string when sliding
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

VirtualInstrument.SlideSpeed

Type : Number
Description : Sliding speed in centimeters per second
Note : Reading this member provides the default value for the instrument, regardless to the live changes that could have been made while the instrument is playing
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.LiveSlideSpeed

Type : Number
Description : "Live" Sliding speed in centimeters per second
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

VirtualInstrument.SlideMinDuration

Type : Number
Description : Sliding minimal duration. If there is not enough time, doesn't play slide noise
Note : Reading this member provides the default value for the instrument, regardless to the live changes that could have been made while the instrument is playing
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.LiveSlideMinDuration

Type : Number
Description : "Live" Sliding minimal duration. If there is not enough time, doesn't play slide noise
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

VirtualInstrument.SlideMaxDuration

Type : Number
Description : Sliding maximal duration. If sliding takes longer, speed it up to fit
Note : Reading this member provides the default value for the instrument, regardless to the live changes that could have been made while the instrument is playing
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.LiveSlideMaxDuration

Type : Number
Description : "Live" Sliding maximal duration. If sliding takes longer, speed it up to fit
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

VirtualInstrument.SlideNoteDurationRatio

Type : Number
Description : Maximum sliding duration, in ration of the note duration
Note : Reading this member provides the default value for the instrument, regardless to the live changes that could have been made while the instrument is playing
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.LiveSlideNoteDurationRatio

Type : Number
Description : "Live" Maximum sliding duration, in ration of the note duration
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

4.5.11.1.2. Performer: Golpe

VirtualInstrument.GolpeVolume

Type : Number
Description : Golpe (hit on instrument body) volume from 0 (muted) to 1 (maximum volume)
Note : Reading this member provides the default value for the instrument, regardless to the live changes that could have been made while the instrument is playing
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.LiveGolpeVolume

Type : Number
Description : "Live" Golpe (hit on instrument body) volume from 0 (muted) to 1 (maximum volume)
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

VirtualInstrument.GolpeAmbientRatio

Type : Number
Description : Golpe (hit on instrument body) dry sound ratio. It's the amount of clear sound that comes from the direct ambient pick-up
as opposed to the muffled sound that is transmitted by the instrument body.
Varies from 0 (all the sound comes from the body) to 1 (all the sound is ambient pick-up)
Note : Reading this member provides the default value for the instrument, regardless to the live changes that could have been made while the instrument is playing
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.LiveGolpeAmbientRatio

Type : Number
Description : "Live" Golpe (hit on instrument body) dry sound ratio. It's the amount of clear sound that comes from the direct ambient pick-up
as opposed to the muffled sound that is transmitted by the instrument body.
Varies from 0 (all the sound comes from the body) to 1 (all the sound is ambient pick-up)
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

VirtualInstrument.GolpeStringTransmissionRatio

Type : Number
Description : Golpe (hit on instrument body) string transmission ratio. It's the amount of energy that is transmitted to the strings,
making them vibrate. Varies from 0 (no energy makes the strings vibrating) to 1 (the whole amount of the hit energy is transmitted to the strings)
Note : Reading this member provides the default value for the instrument, regardless to the live changes that could have been made while the instrument is playing
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.LiveGolpeStringTransmissionRatio

Type : Number
Description : "Live" Golpe (hit on instrument body) string transmission ratio. It's the amount of energy that is transmitted to the strings,
making them vibrate. Varies from 0 (no energy makes the strings vibrating) to 1 (the whole amount of the hit energy is transmitted to the strings)
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

VirtualInstrument.HumanizeArpeggioMode

Type : Number
Description : VirtualInstrument staff automatic humanization : chord arpeggio mode. 0=no humanization (sounds as written), 1=only notes that have a 0 delay are humanized, 2 all notes are humanized
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.LiveHumanizeArpeggioMode

Type : Number
Description : VirtualInstrument staff automatic humanization : "Live" chord arpeggio mode. 0=no humanization (sounds as written), 1=only notes that have a 0 delay are humanized, 2 all notes are humanized
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

VirtualInstrument.HumanizeArpeggioSpeed

Type : Number
Description : VirtualInstrument staff automatic humanization : chord arpeggio speed, from 0=very slow strumming to 100=all strings strummed together
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.LiveHumanizeArpeggioSpeed

Type : Number
Description : VirtualInstrument staff automatic humanization : "Live" chord arpeggio speed, from 0=very slow strumming to 100=all strings strummed together
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

VirtualInstrument.HumanizeVelocityMode

Type : Number
Description : VirtualInstrument staff automatic humanization : beat power marking mode. 0=no humanization (sounds as written), 1=only notes that have a medium (64) velocity are humanized, 2 all notes are humanized
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.LiveHumanizeVelocityMode R

Type : Number
Description : VirtualInstrument staff automatic humanization : "Live" beat power marking mode. 0=no humanization (sounds as written), 1=only notes that have a medium (64) velocity are humanized, 2 all notes are humanized
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

VirtualInstrument.HumanizeVelocityStrength

Type : Number
Description : VirtualInstrument staff automatic humanization : beat power marking strength, from 0=no marking to 100=strong marking
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.LiveHumanizeVelocityStrength

Type : Number
Description : VirtualInstrument staff automatic humanization : "Live" chord arpeggio speed, from 0=very slow strumming to 100=all strings strummed together
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

VirtualInstrument.HumanizeMuteMode

Type : Number
Description : VirtualInstrument staff automatic humanization : String muting mode. 0=no humanization (sounds as written, strings are muted at the end of note duration),
1=only notes that have a regular (100%) pressure time continue to sound until another note is played on the same string,
2 all notes continue to sound until another note is played on the same string
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.LiveHumanizeMuteMode

Type : Number
Description : VirtualInstrument staff automatic humanization : "Live" string muting mode. 0= no humanization. Notes sound as written, strings are muted at the end of note duration,
1= only notes that have a regular (100%) pressure time continue to sound until another note is played on the same string,
2= all notes continue to sound until another note is played on the same string
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

VirtualInstrument.HumanizeMuteOnRest

Type : Boolean
Description : VirtualInstrument staff automatic humanization: true if a rest makes the currently sounding string to be muted, false if not.
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.LiveHumanizeMuteOnRest

Type : Boolean
Description : VirtualInstrument staff automatic humanization: "Live" value. True if a rest makes the currently sounding string to be muted, false if not.
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

4.5.11.1.3. Performer: Microphone Volume

VirtualInstrument.MicrophoneVolumeRow1

Type : Number
Description : General volume of the first row of microphones, from 0 (disabled) to 1 (maximum volume).
Note : Reading this member provides the default value for the instrument, regardless to the live changes that could have been made while the instrument is playing
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.LiveMicrophoneVolumeRow1

Type : Number
Description : "Live" general volume of the first row of microphones, from 0 (disabled) to 1 (maximum volume).
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

VirtualInstrument.MicrophoneVolumeRow2

Type : Number
Description : General volume of the second row of microphones, from 0 (disabled) to 1 (maximum volume).
Note : Reading this member provides the default value for the instrument, regardless to the live changes that could have been made while the instrument is playing
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.LiveMicrophoneVolumeRow2

Type : Number
Description : "Live" general volume of the first row of microphones, from 0 (disabled) to 1 (maximum volume).
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

VirtualInstrument.MicrophoneVolumeRow3

Type : Number
Description : General volume of the third row of microphones, from 0 (disabled) to 1 (maximum volume).
Note : Reading this member provides the default value for the instrument, regardless to the live changes that could have been made while the instrument is playing
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.LiveMicrophoneVolumeRow3

Type : Number
Description : "Live" general volume of the first row of microphones, from 0 (disabled) to 1 (maximum volume).
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

4.5.11.1.4. Performer: Muting

VirtualInstrument.StartMutePower

Type : Number
Description : Mute power when pinching a string. Generally, the string is not muted, except for instance if the performer lets his palm on the strings
Note : Reading this member provides the default value for the instrument, regardless to the live changes that could have been made while the instrument is playing
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.LiveStartMutePower

Type : Number
Description : "Live" mute power when pinching a string. Generally, the string is not muted, except for instance if the performer lets his palm on the strings
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

VirtualInstrument.StopMutePower

Type : Number
Description : Mute power when stopping a string. When a string is stopped, power at which it is deadened. 0 continues to sound, 1 immediate stop
Note : Reading this member provides the default value for the instrument, regardless to the live changes that could have been made while the instrument is playing
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.LiveStopMutePower

Type : Number
Description : "Live" mute power when stopping a string. When a string is stopped, power at which it is deadened. 0 continues to sound, 1 immediate stop
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

VirtualInstrument.LiveMutePower

Type : Number
Description : "Live" instant mute power . 0 continues to sound, 1 immediate stop. This member enables to set the current mute power instantly while the instrument is playing.
It will be reset to either LiveStartMutePower or LiveStopMutePower next time a note will be respectively started or stopped.
Note : Reading this member is useless and always return 0.
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

4.5.11.1.5. Performer: Slapping

VirtualInstrument.SlapPower

Type : Number
Description : Slap noise power
Note : Writing affects the settings for the next time this instrument is used. Cannot be changed "live" while the instrument is playing

VirtualInstrument.SlapSharpness

Type : Number
Description : Slap noise sharpness
Note : Writing affects the settings for the next time this instrument is used. Cannot be changed "live" while the instrument is playing

VirtualInstrument.SlapDuration

Type : Number
Description : Slap noise duration (ms)
Note : Writing affects the settings for the next time this instrument is used. Cannot be changed "live" while the instrument is playing

4.5.11.1.6. Performer: Strumming (right hand)

VirtualInstrument.StrumMinDistance

Type : Number
Description : Minimum distance of strumming relatively to the bridge
Note : Reading this member provides the default value for the instrument, regardless to the live changes that could have been made while the instrument is playing
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.LiveStrumMinDistance

Type : Number
Description : "Live" minimum distance of strumming relatively to the bridge
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

VirtualInstrument.StrumMaxDistance

Type : Number
Description : Maximum distance of strumming relatively to the bridge
Note : Reading this member provides the default value for the instrument, regardless to the live changes that could have been made while the instrument is playing
Setting his member affects the default value for the instrument as well as the current values if the instrument is playing

VirtualInstrument.LiveStrumMaxDistance

Type : Number
Description : Maximum distance of strumming relatively to the bridge
Note : Reading this member provides the current value for the instrument while playing, as it has been altered by writing or score curves
Setting this member DOES NOT affect the default settings for the instrument. It sets the current value and has effect only if the instrument is currently playing

VirtualInstrument.SimplifiedStrumming

Type : Boolean
Description : True if the strumming is in Simplified mode. In this mode, writing to the Strumming array is forbidden
Note : Setting this value from false to true will result in losing any change you made to the Strumming array
Affects the settings for the next time this instrument is used. Cannot be changed "live" while the instrument is playing

VirtualInstrument.PlectrumMaterial

Type : Number
Description : Plectrum material hardness from 0 (soft) to 1 (hard)
Note : This member can only be written if VirtualInstrument.SimplifiedStrumming (above) is true
Affects the settings for the next time this instrument is used. Cannot be changed "live" while the instrument is playing

VirtualInstrument.PlectrumThickness

Type : Number
Description : Plectrum material thickness from 0 (thin) to 1 (bold)
Note : This member can only be written if VirtualInstrument.SimplifiedStrumming (above) is true
Affects the settings for the next time this instrument is used. Cannot be changed "live" while the instrument is playing

VirtualInstrument.PlectrumGripHeight

Type : Number
Description : Plectrum grip height from 0 (very close to the tip) to 1 (far from the tip)
Note : This member can only be written if VirtualInstrument.SimplifiedStrumming (above) is true
Affects the settings for the next time this instrument is used. Cannot be changed "live" while the instrument is playing

VirtualInstrument.PlectrumPower

Type : Number
Description : Plectrum power from 0 (very soft playing) to 1 (very hard playing)
Note : This member can only be written if VirtualInstrument.SimplifiedStrumming (above) is true
Affects the settings for the next time this instrument is used. Cannot be changed "live" while the instrument is playing

VirtualInstrument.Strumming

Type : Strumming
Description : Array of strumming settings according to note velocities (power)
Affects the settings for the next time this instrument is used. Cannot be changed "live" while the instrument is playing

4.5.11.1.7. Physical: Body

VirtualInstrument.BodySpectrumSize

Type : Number
Description : size of frequency spectrum (FFT) of the body internal resonances. Setting this member to 0 will clear the body spectrum

VirtualInstrument.BodySpectrum

Type : BodySpectrum
Description : frequency spectrum (FFT) data of the body internal resonances. Usually this spectrum is calculated from a recorded impulse response

VirtualInstrument.BodySpectrumStrength

Type : Number
Description : Computing strength of the body frequency spectrum 0=flat, <1=smooth, 1=standard, >1=sharp .

VirtualInstrument.BodyEqualizerSize

Type : Number
Description : number of elements in the body equalizer array

VirtualInstrument.BodyEqualizer

Type : BodyEqualizer
Description : multi-band equalizer array applied to the body frequency spectrum

4.5.11.1.8. Physical: Microphones

VirtualInstrument.MicrophoneRow1

Type : Microphone Array
Description : First row of microphones. Each row is an array of microphones, with as many elements as there are strings.

VirtualInstrument.MicrophoneRow2

Type : Microphone Array
Description : Second row of microphones. Each row is an array of microphones, with as many elements as there are strings.

VirtualInstrument.MicrophoneRow3

Type : Microphone Array
Description : Second row of microphones. Each row is an array of microphones, with as many elements as there are strings.

4.5.11.1.9. Physical: Strings

VirtualInstrument.NumberOfStrings

Type : Number
Description : Current number of strings for the instrument

VirtualInstrument.String

Type : VirtualString
Description : Instrument strings

4.5.11.1.10. Physical: Sympathy resonance

VirtualInstrument.SympathyPower

Type : Number
Description : Sympathy resonance power (multiplier of the natural power of the strings)

VirtualInstrument.SympathyDecay

Type : Number
Description : Sympathy decay (multiplier of the natural decay of the strings)

VirtualInstrument.SympathySharpness

Type : Number
Description : Sympathy sharpness (multiplier of the natural sharpness of the string)

VirtualInstrument.SympathyDetune

Type : Number
Description : Sympathy detune (0-1)

4.5.11.1.11. Unsorted

VirtualInstrument.LiveHumanizeHumanizeVelocityMode

You can only write this value.

4.5.11.2. VirtualInstrument methods

4.5.11.2.1. I/O

VirtualInstrument.Save(...)

Description : Save the whole instrument (Physical data + Play Mode) to a separate file
Parameters : String: complete path + file name for saving. The usual file extension for such files is ".vif"
Return value : Boolean: true if the operation succeeded

VirtualInstrument.Load(...)

Description : Load the whole instrument (Physical data + Play Mode) from a separate file. The usual file extension for such files is ".vif"
Parameters : String: complete path + file name for loading
Return value : Boolean: true if the operation succeeded

VirtualInstrument.SavePhysical(...)

Description : Save the instrument physical data to a separate file
Parameters : String: complete path + file name for saving. The usual file extension for such files is ".vph"
Return value : Boolean: true if the operation succeeded

VirtualInstrument.LoadPhysical(...)

Description : Load the instrument physical data from a separate file. The usual file extension for such files is ".vph"
Parameters : String: complete path + file name for loading
Return value : Boolean: true if the operation succeeded

VirtualInstrument.SavePlayMode(...)

Description : Save the instrument play mode data to a separate file
Parameters : String: complete path + file name for saving. The usual file extension for such files is ".vip"
Return value : Boolean: true if the operation succeeded

VirtualInstrument.LoadPlayMode(...)

Description : Load the instrument play mode data from a separate file. The usual file extension for such files is ".vip"
Parameters : String: complete path + file name for loading
Return value : Boolean: true if the operation succeeded

VirtualInstrument.LoadBodySpectrum(...)

Description : Load the instrument body spectrum from a separate file
Parameters : String: complete path + file name for loading
Return value : Boolean: true if the operation succeeded

VirtualInstrument.SaveBodySpectrum(...)

Description : Save the instrument body spectrum in a separate file
Parameters : String: complete path + file name for saving
Return value : Boolean: true if the operation succeeded

4.5.11.2.2. Unsorted

VirtualInstrument.AddBodyEQ(...)

VirtualInstrument.AddString(...)

VirtualInstrument.DeleteBodyEQ(...)

VirtualInstrument.DeleteString(...)

4.6. Interface

This section describes all objects that enable to build an interface between your script and the user.
You will learn how to create custom dialog boxes, text windows, progress bars...
We recommend to read carefully the chapter about Interface Composer (page 56), which is the most powerful way to create an Interface

4.6.1. Dialog Object Type

General description of this object type : The Dialog Object Type enables to manage dialog boxes to ask choices to the user. The easiest way to create a Dialog Object is to use the Interface Composer (page 56) facilities.But, If you want, althought not recommended, you can create manually a complete dialog by using the Dialog object type methods. In this case, here are the steps you have to follow :

- 1) Create a new dialog object using the NewDialog(...) (page 496) function
- 2) Add items to this dialog
- 3) Show the dialog using dialog.Show(...) (page 503), this validates items and prepares to call standard item methods.
- 4) Change item status and settings using the item fields and methods.
- 5) Manage the dialog using the dialog.Select(...) (page 503) dialog method
- 6) Close the dialog using the dialog.Dispose(...) (page 503) dialog method

Please note : you cannot add custom methods to an item if the dialog has been created this way. This is why it is recommended to use an Interface Composer (page 56) file instead.

A set of predefined and basic dialog boxes is provided : **Alert** displays an alert message to the user, **Message** displays an information message, **Confirmation** asks the user for a Yes/No choice.
Example :
Here is a sample of a dialog built only by script :

```
Include "MSDefine.mys"
--- Creating dialog
local dialog=NewDialog("Demo Dialog",400,400)
local itemOk
local itemCancel
local item=0
local itemRadio
local itemCheck
local itemColor
local itemText
local itemMenu
local itemStyle
local itemHeadShape
local itemDuration
local result =-1

if dialog then
    -- Filling dialog with items
    itemOk=dialog.NewPushButton("Ok",300,400-32)
    itemCancel=dialog.NewPushButton("Cancel",200,400-32)
    itemCheck=dialog.NewCheckBox("Active",16,30,false)
    itemColor=dialog.NewColorSelector(0,140,30,80,20)
    itemMenu=dialog.NewMenu("Item 1","Item 2","Item 3",1,300,60)
    dialog.NewStaticText("This is a test dialog",9,10)
    itemRadio=dialog.NewRadioButtons("Disable ","Enable ","Hide ","Show ",16,60)
    itemText=dialog.NewEditText("default",16,200,180,14)
    itemStyle=dialog.NewStyle(210,180)
    itemHeadShape=dialog.NewHeadShapeSelector(SHAPE_NORMAL,16,240)
    itemDuration=dialog.NewDurationSelector(DURATION_64TH,DURATION_32ND,DURATION_16TH,
    DURATION_EIGHTH,DURATION_QUARTER,DURATION_HALF,DURATION_QUARTER,120,240)
    itemScroll=dialog.NewScrollBar(10,320,380,16)
    itemList=dialog.NewList(280,100)
    --- After dialog.Show() we can't add items ...
    dialog.Show()
    ---... but we can act on them, for example, by filling the list.
    for i=1,20 do
        itemList.SetExtraValue(i,"Choice "..i);
    end
    -- Processing dialog
    while (item ~= itemOk and item ~= itemCancel) do
    item=dialog.Select()
    if item~=nil then
        if item.Name=="Radio 1" then itemDuration.IsActive=false end
        if item.Name=="Radio 2" then itemDuration.IsActive=true end
        if item.Name=="Radio 3" then itemDuration.IsVisible=false end
        if item.Name=="Radio 4" then itemDuration.IsVisible=true end
        end
    if item==itemCheck then
        itemDuration.Value=DURATION_QUARTER
        end
    end
    if item==itemOk then
    Message(itemDuration.Value)
    end
    -- Disposing dialog
    dialog.Dispose()
end
```

4.6.1.1. Dialog global functions

4.6.1.1.1. Predefined

Alert(...)

Description : Display an Alert message. The user ought to click "Ok" to continue. Typicaly, a script use this function when an error occur.
Parameters : The string to display
Return value : None
Example :

```
Alert("Don't PANIC this is ONLY a TEST !!!")
```

Message(...)

Description : Displays an information message. The user has to click "Ok" to continue. Typically, a script uses this function to display a result or warn about something.
Parameters : Undefined number of parameters, of undefined types. This function displays the list of parameters, separated by a space.
Return value : None
Example :

```
Message("I just wanted to tell you a little peekaboo !")
```

Confirmation(...)

Description : Display a choice message. The user ought to click "Yes" to validate, "No" to cancel. The dialog box buttons are automatically translated in the current user's language. Typically, a script use this function to ask the user to do a choice.
Parameters : Undefined number of parameters, of undefined types. This function displays the list of parameters, separated by a space.
Return value : Boolean : true if the user validated the choice, false otherwise.
Example :

```
if(Confirmation("Here is confirmation dialog box, do you understand how it works ?")==true) then
   Message("Nice !")
else
   Message("So, read again the section in the manual")
end
```

4.6.1.1.2. Creating

NewDialog(...)

Description : Create a new empty dialog. Your script will have to add the items to it. The dialog window is centered on the screen but the user can move it.
Parameters : Optional :
- String : the dialog title
- Number : Horizontal size of the window.
- Number : Vericail size of the window.

Return value : A New Dialog object
Example :

```
   myDialog=NewDialog("This is my title",400,400)
```

CreateDialog(...)

Description : Create a new dialog from an Interface Composer (page 56) file.
The file is searched in the current path but you can specify a relative path in the file name.
For example :
myDialog=CreateDialog("MyDialogObject.myc")
or
myDialog=CreateDialog("interface/FrenchMyDialogObject.myc")
are valid.
If an item is named "Ok" it become the default item.
If an item is named "Cancel" it become the cancel item/
Parameters : String : the Interface Composer file name.
Return value : A New Dialog object
Example :

```
   myDialog=CreateDialog("MyDialogObject.myc")
```

4.6.1.2. Dialog values

4.6.1.2.1. Display

Dialog.GraphContext

Type : Graph (page 516)
Description : Graphical access to the dialog.

4.6.1.2.2. Help

Dialog.ContextHelp

Type : Styled string
Description : General content of the help line displayed when the mouse cursor is in the dialog and outside an item with help.
The default value is provided by the --ABSTRACT: tag in the script.

4.6.1.2.3. Items

Dialog.NumberOfDialogItems

Type : Number
Description : Number of dialog items in this dialog

Dialog.DialogItems

Type : DialogItem array
Description : "Root" of item array for this dialog.
Cannot be used as is. Must be indexed.
Example :

```
--- Displays "true" if the second item of a symbol is a push button
item=myDialog.DialogItems[2]
if item~=nil and item.Type=="Button" then print("true") end
```

Dialog.FirstDialogItem

Type : DialogItem
Description : First item in this dialog (nil if none)

Dialog.LastDialogItem

Type : DialogItem
Description : Last item in this dialog (nil if none)

4.6.1.2.4. Path

Dialog.DefaultPath

Type : String: the default path.
Description : This is the path where the script is located. This is useful, for example, if you store extra data in the script folder and what access to them.

4.6.1.2.5. Private data

Dialog.UserTable

Type : Table
Description : User table, provided to store private data for this dialog

4.6.1.2.6. Window appearance

Dialog.Title

Type : String
Description : Dialog box title

Dialog.AreaLeft

Type : Number
Description : Dialog box left edge position, in pixel from the screen left bound

Dialog.AreaTop

Type : Number
Description : Dialog box top edge position, in pixel from the screen top bound

Dialog.AreaWidth

Type : Number
Description : Dialog box width, in pixel

Dialog.AreaHeight

Type : Number
Description : Dialog box height, in pixel

Dialog.MetalAppearance

Type : Boolean
Description : Metal appearance. Available only on some system. (Experimental)

Dialog.IsFrontWindow

Type : Boolean
Description : True if the dialog is the frontmost window, false if it run in background.
You can use this value with Application.IsActive (page 181) to know if the application is active.
Changing this value whill bring the window to the front or to the back.
Floating windows are not taken into account

Dialog.IsFrontFloatingWindow

Type : Boolean
Description : True if the dialog is the frontmost window, false if it run in background.
Changing this value whill bring the window to the front or to the back.
Floating windows are taken into account

4.6.1.3. Dialog methods

4.6.1.3.1. Items

Dialog.NewPushButton(...)

Description : Create a new push button item.
You ought to call this method after the dialog was created but before the first Dialog.Select(...) (page 503) call.
Parameters : Optional:
- String : Caption displayed in the push button (by default "Button")
- Number : Horizontal position, in pixel, in the dialog (by default 0)
- Number : Vertical position, in pixel, in the dialog (by default 0)
- Number : Horizontal size, in pixel (by default 90)
- Number : Vertical size, in pixel (by default 20)

Return value : Item type object, or nil if the creation failed.
Example :

```
itemOk=dialog.NewPushButton("Ok",300,400)
```

Dialog.NewCheckBox(...)

Description : Create a new check box item.
You ought to call this method after the dialog was created but before the first Dialog.Select(...) (page 503) call.
Parameters : Optional:
- String : Caption displayed in the item (by default "Checkbox")
- Number : Horizontal position, in pixel, in the dialog (by default 0)
- Number : Vertical position, in pixel, in the dialog (by default 0)
- Boolean : current status (by default false)
- Number : Horizontal size, in pixel (by default the caption size)
- Number : Vertical size, in pixel (by default 16)

Return value : Item type object, or nil if the creation failed.
Example :
```
itemCheck=dialog.NewCheckBox("Enable activity",100,200)
```

Dialog.NewStaticText(...)

Description : Create a new static text item.
You ought to call this method after the dialog was created but before the first Dialog.Select(...) (page 503) call.
Parameters : Optional:
- String : Caption displayed in the item (by default "Static text")
- Number : Horizontal position, in pixel, in the dialog (by default 0)
- Number : Vertical position, in pixel, in the dialog (by default 0)
- Number : Horizontal size, in pixel (by default the caption size)
- Number : Vertical size, in pixel (by default 12)

Return value : Item, or nil if the creation failed.
Example :
```
dialog.NewStaticText("Please select a value : ",32,32)
```

Dialog.NewMediumStaticText(...)

Description : Create a new medium-sized static text item.
You ought to call this method after the dialog has been created but before the first Dialog.Select(...) (page 503) ca
Parameters : Optional:
- String : Caption displayed in the item (by default "Static text")
- Number : Horizontal position, in pixel, in the dialog (by default 0)
- Number : Vertical position, in pixel, in the dialog (by default 0)
- Number : Horizontal size, in pixel (by default the caption size)
- Number : Vertical size, in pixel (by default 14)

Return value : Item, or nil if the creation failed.
Example :
```
dialog.NewMediumStaticText("Please select a value : ",32,32)
```

Dialog.NewBigStaticText(...)

Description : Create a new big-sized static text item.
You ought to call this method after the dialog has been created but before the first Dialog.Select(...) (page 503) ca
Parameters : Optional:
- String : Caption displayed in the item (by default "Static text")
- Number : Horizontal position, in pixel, in the dialog (by default 0)
- Number : Vertical position, in pixel, in the dialog (by default 0)
- Number : Horizontal size, in pixel (by default the caption size)
- Number : Vertical size, in pixel (by default 16)

Return value : Item, or nil if the creation failed.
Example :
```
dialog.NewBigStaticText("Please select a value : ",32,32)
```

Dialog.NewEditText(...)

Description : Create a new edit text item.
You ought to call this method after the dialog was created but before the first Dialog.Select(...) (page 503) call.
Parameters : Optional:
- String : Text displayed in the item (by default "")
- Number : Horizontal position, in pixel, in the dialog (by default 0)
- Number : Vertical position, in pixel, in the dialog (by default 0)
- Number : Horizontal size, in pixel (by default 64)
- Number : Vertical size, in pixel (by default 14)

Return value : Item, or nil if the creation failed.
Example :
```
editItem=dialog.NewEditText("2040",120,32)
```

Dialog.NewColorSelector(...)

Description : Create a new select color index item.
You ought to call this method after the dialog was created but before the first Dialog.Select(...) (page 503) call.
Parameters : Optional:
- Number : Color index from 0 to 255 (by default 0)
- Number : Horizontal position, in pixel, in the dialog (by default 0)
- Number : Vertical position, in pixel, in the dialog (by default 0)
- Number : Horizontal size, in pixel (by default 80)
- Number : Vertical size, in pixel (by default 20)

Return value : Item, or nil if the creation failed.
Example :
```
colorItem=dialog.NewColorSelector(0,200,150)
```

Dialog.NewRadioButtons(...)

Description : Create a new radio button group. This method don't create one item but many items.
You ought to call this method after the dialog was created but before the first Dialog.Select(...) (page 503) call.
Parameters : Optional:
- Strings : Captions of the items : a list of strings.
- Number : Horizontal position, in pixel, in the dialog (by default 0)
- Number : Vertical position, in pixel, in the dialog (by default 0)
- Number : Horizontal size, in pixel (by default 90)
- Number : Vertical size, in pixel (by default 20)

Return value : Item, or nil if the creation failed.
Example :
```
itemRadio=dialog.NewRadioButtons("Disable ","Enable ","Hide ","Show ",1,16,60)
```

Dialog.NewMenu(...)

Description : Create a new radio button group.
You ought to call this method after the dialog was created but before the first Dialog.Select(...) (page 503) call.
Parameters : Optional:
- Strings : A list of strings which will be displayed in the menu
- Number : Value displayed when dialog open (by default 0)
- Number : Horizontal position, in pixel, in the dialog (by default 0)
- Number : Vertical position, in pixel, in the dialog (by default 0)
- Number : Horizontal size, in pixel (by default 90)
- Number : Vertical size, in pixel (by default 18)

Return value : Item, or nil if the creation failed.
Example :
```
itemMenu=dialog.NewMenu("Other 1","Other 2","Other 3","Other 4","Unselected",1,300,100)
```

Dialog.NewList(...)

Description : Create a new list of strings.
You ought to call this method after the dialog was created but before the first Dialog.Select(...) (page 503) call.
You will be able to build the list after Dialog.Select(...) (page 503) call.
Parameters : Optional:
- Number : Horizontal position, in pixel, in the dialog (by default 0)
- Number : Vertical position, in pixel, in the dialog (by default 0)
- Number : Horizontal size, in pixel (by default 120)
- Number : Vertical size, in pixel (by default 200)

Return value : Item, or nil if the creation failed.
Example :
```
itemList=dialog.NewList(300,100)
```

Dialog.NewTabView(...)

Description : Create a new tab view.
You ought to call this method after the dialog was created but before the first Dialog.Select(...) (page 503) call.
You will be able to build the list after Dialog.Select(...) (page 503) call.
Parameters : Optional:
- Number : Horizontal position, in pixel, in the dialog (by default 0)
- Number : Vertical position, in pixel, in the dialog (by default 0)
- Number : Horizontal size, in pixel (by default 120)
- Number : Vertical size, in pixel (by default 20)

Return value : Item, or nil if the creation failed.
Example :

```
item=dialog.NewTabView(4,4,200)
if item~=nil then
   item.SetExtraValue(1,"Notes")
   item.SetExtraValue(2,"Staves")
   item.SetExtraValue(3,"Score")
end
```

Dialog.NewFrameRect(...)

Description : Create a new frame rect item.
You ought to call this method after the dialog was created but before the first Dialog.Select(...) (page 503) call.
Parameters : Optional:
- String : Caption displayed at the top left edge of the item (by default none)
- Number : Horizontal position, in pixel, in the dialog (by default 0)
- Number : Vertical position, in pixel, in the dialog (by default 0)
- Number : Horizontal size, in pixel (by default the caption size)
- Number : Vertical size, in pixel (by default 12)

Return value : Item, or nil if the creation failed.
Example :

```
dialog.NewFrameRect("Score",0,0,200,120)
```

Dialog.NewText(...)

Description : Create a new int32_t text.
You ought to call this method after the dialog was created but before the first Dialog.Select(...) (page 503) call.
You will be able to build the list after Dialog.Select(...) (page 503) call.
Parameters : Optional:
- Number : Horizontal position, in pixel, in the dialog (by default 0)
- Number : Vertical position, in pixel, in the dialog (by default 0)
- Number : Horizontal size, in pixel (by default 120)
- Number : Vertical size, in pixel (by default 200)

Return value : Item, or nil if the creation failed.
Example :

```
itemList=dialog.NewText(300,100)
```

Dialog.NewHeadShapeSelector(...)

Description : Create a new head shape selector.
You ought to call this method after the dialog was created but before the first Dialog.Select(...) (page 503) call.
Parameters : Optional:
- Number : Current head shape (by default 0)
- Number : Horizontal position, in pixel, in the dialog (by default 0)
- Number : Vertical position, in pixel, in the dialog (by default 0)
- Number : Horizontal size, in pixel (by default 64)
- Number : Vertical size, in pixel (by default 64)

Return value : Item, or nil if the creation failed.
Example :

```
itemHeadShape=dialog.NewHeadShapeSelector(SHAPE_NORMAL,16,240)
```

Dialog.NewDurationSelector(...)

Description : Create a new note duration selector.
You ought to call this method after the dialog was created but before the first Dialog.Select(...) (page 503) call.
Parameters : Optional:
- Numbers : List of duration. This will be the duration displayed in the selector.
- Number : Duration displayed at the beginning
- Number : Horizontal position, in pixel, in the dialog (by default 0)
- Number : Vertical position, in pixel, in the dialog (by default 0)
- Number : Horizontal size, in pixel (by default 64)
- Number : Vertical size, in pixel (by default 64)

Return value : Item, or nil if the creation failed.
Example :
```
itemDuration=dialog.NewDurationSelector(DURATION_64TH,DURATION_32ND,DURATION_16TH,
DURATION_EIGHTH,DURATION_QUARTER,DURATION_HALF,DURATION_QUARTER,120,240)
```

Dialog.NewScrollBar(...)

Description : Create a new scroll bar.
You ought to call this method after the dialog was created but before the first Dialog.Select(...) (page 503) call.
Parameters : Optional:
- Number : Horizontal position, in pixel, in the dialog (by default 0)
- Number : Vertical position, in pixel, in the dialog (by default 0)
- Number : Horizontal size, in pixel (by default 16)
- Number : Vertical size, in pixel (by default 64)

Return value : Item, or nil if the creation failed.
Example :
```
itemScroll=dialog.NewScrollBar(200,200)
```

Dialog.NewSlider(...)

Description : Create a new slider.
You ought to call this method after the dialog was created but before the first Dialog.Select(...) (page 503) call.
Parameters : Optional:
- Number : Horizontal position, in pixel, in the dialog (by default 0)
- Number : Vertical position, in pixel, in the dialog (by default 0)
- Number : Horizontal size, in pixel (by default 16)
- Number : Vertical size, in pixel (by default 64)

Return value : Item, or nil if the creation failed.
Example :
```
itemSlider=dialog.NewSlider(200,200)
```

Dialog.NewKnob(...)

Description : Create a new knob.
You have to call this method after the dialog is created but before the first Dialog.Select(...) (page 503) call.
Parameters : Optional:
- String : Title displayed in the knob, by default none.
- Number : Horizontal position, in pixel, in the dialog (by default 0)
- Number : Vertical position, in pixel, in the dialog (by default 0)
- Number : Horizontal size, in pixel (by default 60)
- Number : Vertical size, in pixel (by default 60)

Return value : Item, or nil if the creation failed.
Example :
```
itemKnob=dialog.NewKnob(200,200)
```

Dialog.NewStyle(...)

Description : Create a new text style selector.
You ought to call this method after the dialog was created but before the first Dialog.Select(...) (page 503) call.
Parameters : Optional:
- Number : Horizontal position, in pixel, in the dialog (by default 0)
- Number : Vertical position, in pixel, in the dialog (by default 0)
- Number : Font number (by default geneva)
- Number : Font face (by default 0 standard)
- Number : Font size (by default 12)

Return value : Item, or nil if the creation failed.
Example :
```
itemScroll=dialog.NewStyle(200,200)
```

4.6.1.3.2. Managing

Dialog.Select(...)

Description : Manages an open dialog box and returns the item the user selected. Call regularly this method.
Parameters : [optional] Number: number of 1/60th of second let to the system if no event has to be processed (default: 5)
A shorter delay will make your events faster, but the CPU charge will be higher when nothing happens on the dialog.
A value of 0 is possible, but not recommended, as the script will take 100% of the CPU charge when idle.
Return value : An item or nil
Example :

```
item=myDialog.Select()
```

Dialog.Show(...)

Description : Show the dialog. Use this method after the items was added to the dialog and before the first Dialog.Select(...) (above) call. You don't need to call this method if you created the dialog with CreateDialog(...) (p e 496).
Parameters : None
Return value : None
Example :

```
  myDialog.Show()
```

Dialog.Hide(...)

Description : Hide the dialog.
Parameters : None
Return value : None
Example :

```
  myDialog.Hide()
```

Dialog.BringToFront(...)

Description : You only need this method if the dialog object is a floating window. The dialog window will be set in front of all other windows except, of course, the palette window.
Parameters : None
Return value : None

Dialog.Dispose(...)

Description : Dispose a dialog object previously created with NewDialog(...) (page 496) or CreateDialog(...) (pag and all the data related to it. The window is closed.
Note : After this, you can't access to the items embedded in the dialog which was disposed too.
Parameters : None
Return value : None
Example :

```
  myDialog.Dispose()
```

Dialog.DrawContent(...)

Description : Draw the entire content of the dialog
Parameters : None
Return value : None
Example :

```
  myDialog.DrawContent()
```

Dialog.GetItem(...)

Description : Locate the item with a given name.
Parameters : String : the item name.
Return value : Item type object, or nil if the search failed.
Example :

```
item=dialog.GetItem("Button 1")
```

Dialog.GetItemUnderMouse(...)

Description : Locate the first item in the dialog which is pointed by the mouse cursor.
Return value : Item type object, or nil if none.
Example :

```
This can be used, for example to display informations about the items in the help window.
Here is an idle method of a dialog which display the item name when mouse move on it.
-- Idle : called when nothing happens to the dialog box
function Idle(dialog,item)
-- Write your function here
item=dialog.GetItemUnderMouse()
   if(item~=nil) then
      Application.Palette.SendHelpMessage(item.Name)
   end
end
```

Dialog.MakeTableFromItems(...)

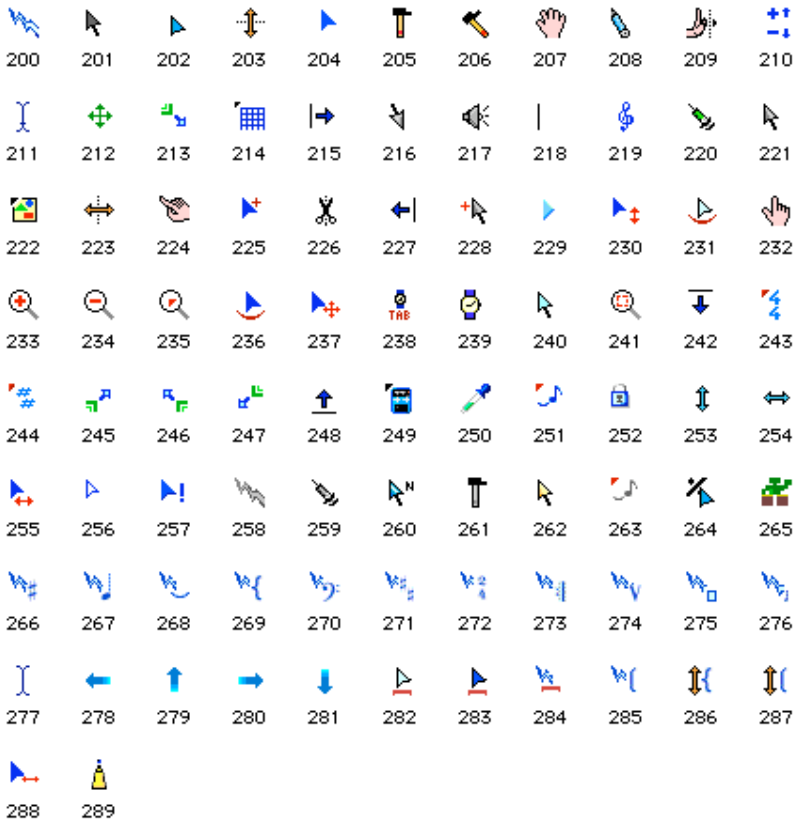
Description : Transfer all the item values into a table.
This enables to collect all the item values into a table variable, to be used later.
This method must be called before the dialog is disposed through Dialog.Dispose(...) (page 503) method, but the table elements remain valid after the dialog is disposed, contrarily to the accesses through method for example.
·Only the items that can have a value are transfered.
For example, if a static text item has the name "StaticText", the "StaticText" element of the returned table won't exist
·For Frame items, the table element does not reflect the item Value, which is an item, but the name of this item. This is because table elements can be accessed after the dialog is disposed, so the item can have no meaning. Therefore, if "frameItem" is a dialog item of Frame type, the matching table element receives dialog.GetItem(" frameItem").Value.Name
·If several items have the same name, the matching element of the table contains the value of one of them (but you cannot rely on which one it will be)
·If the item name complies with Lua variable naming, the table element can be accessed through the structure-like table access (table.field)
Parameters : None
Return value : A table filled with all values of dialog items
Example :

```
   -- We suppose a dialog, containing two items named
   -- "EditText1" and "Max range" has been processed
   myTable=dialog.MSMakeTableFromItems()
   dialog.Dispose()
   value1=myTable.EditText1
   value2=myTable["Max range"]
```

4.6.1.3.3. Mouse

Dialog.SetMouseCursorShape(...)

Description : Relate a mouse cursor shape to the dialog.
Parameters : Number from 200 to 289, the shape to use.



Return value : None
Example :

```
dialog.SetMouseCursorShape(210)
```

4.6.2. DialogItem Object Type

General description of this object type : Dialog item management
Dialog items are indexed elements. It means restrictions about this kind of object apply.

From the Interface composer, you can change the default behaviour of dialog items by writing a set of methods for this object.
Methods are written as regular functions, that receive parameters and return one or several values.
These methods can either be system (DialogItem.Init, DialogItem.Push, DialogItem.Release, DialogItem.Idle, DialogItem.Exit ...) or custom.

System-called methods are called by the system, during Dialog.Select(...) (page 503) processing, and receive the embedding dialog object, and the current dialog item as parameter.
They return a boolean, that tells the system whether the default behaviour of this object has still to be applied (false or nil) or not (true)

Adding a custom method to a dialog item object enriches the property list for this object.
Such methods can be called from the main script, from another method of the same object, or from another method of another object.
Custom method can receive as many parameters you need, and return any number and type of values.

When an item method is called without any parameter, the embedding dialog and current item are sent as parameters.
For example, calling item.MyMethod() will run the "MyMethod" method of the "item" object, and send (dialog,item) as parameters.
It is therefore functionally equivalent to item.MyMethod(dialog,item)
But, for example, calling item.MyMethod(123) will send only the value 123 to the "MyMethod" method of "item" object.
No extra parameter will be added.

Because using global variables is not recommended in object methods (it can provide side effect if this variable name is also used in the main program), each item has its own private storage space.
This field, called "UserTable", returns a private lua table (see lua documentation to learn how to use tables), which is created before the "Init" method of the item is called, and destroyed after the "Exit" method has been called.
A table can contain any number and type of values, so the private storage space of each item is virually infinite.

4.6.2.1. DialogItem values

4.6.2.1.1. Browsing dialog items

DialogItem.Next

Type : DialogItem
Description : Next dialog item
If there is no next dialog item, value is nil.
Note : You can browse dialog items through Next and Previous fields, but you can also access a dialog itel at a given index through, for example, Dialog.DialogItems (page 497)[index]

DialogItem.Previous

Type : DialogItem
Description : Previous dialog item
If there is no previous dialog item, value is nil.
Note : You can browse dialog items through Next and Previous fields, but you can also access a dialog item at a given index through, for example, Dialog.DialogItems (page 497)[index]

DialogItem.Index

Type : Number
Description : Index of this dialog item
Example :

```
-- Displays the index of the "OK" dialog item of a dialog, if exists
item=myDialog.GetItemByName("OK")
if item~=nil then
   print(item.Index)
end
```

4.6.2.1.2. Dialog item general parameters

DialogItem.Type

Type : String
Description : Dialog item type
Note : It is not possible to change the type of a dialog item once created

DialogItem.Source

Type : String
Description : Dialog item source file name
Note : if the dialog item has been created programatically, this value is "Program".
Otherwise, it's the name of the MYC file that contained this item.
Can be written to store private information about the source of the item.

DialogItem.AreaLeft

Type : Number
Description : X position of the item left edge, in pixel

DialogItem.AreaTop

Type : Number
Description : Y position of the item top edge, in pixel

DialogItem.AreaRight

Type : Number
Description : X position of the item right edge, in pixel

DialogItem.AreaBottom

Type : Number
Description : Y position of the item bottom edge, in pixel

DialogItem.AreaWidth

Type : Number
Description : item width, in pixel

DialogItem.AreaHeight

Type : Number
Description : item height, in pixel

DialogItem.Name

Type : String
Description : Item name
Note : Any item has a name.
Be careful not to confuse between item "name", "caption" and "value"
"Name" is the internal name that has been given to this item. It is never displayed.
"Caption" is the fixed text that is displayed in an item, for example "OK" on a button. Caption cannot be changed by the user.
"Value" is what can be changed by the use in this item (for example, state of a check box, content of an edit text...)

DialogItem.Caption

Type : String
Description : Item caption
Note : Only the following item types can have a caption :
push button
check box
radio button
frame
static text

Be careful not to confuse between item "name", "caption" and "value"
"Name" is the internal name that has been given to this item. It is never displayed.
"Caption" is the fixed text that is displayed in an item, for example "OK" on a button. Caption cannot be changed by the user.
"Value" is what can be changed by the use in this item (for example, state of a check box, content of an edit text...

Caption can only be get or set once the dialog has been actually displayed, either :
- when created using CreateDialog(...) (page 496),
- or after Dialog.Show(...) (page 503) has been called
- or Dialog.Select(...) (page 503) has been called at least once.

Otherwise, the returned value is nil, and setting this value has no effect.

DialogItem.Value

Type : Depends on the item type
Description : Item value
Note : Only the following item types can have a value :

- check box (Boolean) : true if selected, false otherwise
- radio button (Boolean) : true if selected, false otherwise
- frame (DialogItem) : selected radio button item in this group
- edit text (String) : edit text content
- pop-up menu (Number) : selected menu item number
- color selector (Number) : selected color
- text style (TextStyle) : selected text style
- note head shape (Number) : selected head shape (see constants in MSDefine (page 100).mys)
  SHAPE_NORMAL=0
  SHAPE_TRIANGLE=1
  SHAPE_DIAMOND=2
  SHAPE_CROSS=3
  SHAPE_PLUS=4
  SHAPE_SQUARE=5
  SHAPE_BEAM=6
  SHAPE_NONE=7
  SHAPE_FLAG=8
  SHAPE_HALF_CIRCLE=9
  SHAPE_DROP=10
  SHAPE_SAND_GLASS=11
  SHAPE_ARC=12
  SHAPE_EMPTY_DIAMOND=13
  SHAPE_CONE=14
  SHAPE_CRESCENT=15
  SHAPE_LEFT_TRIANGLE=16
  SHAPE_BELL=17
  SHAPE_CONCAVE_TRIANGLE=18
  SHAPE_CONVEX_TRIANGLE=19
  SHAPE_JOHNSON_TRIANGLE=20
  SHAPE_JOHNSON_SQUARE=21
  SHAPE_CIRCLE_BOLD_CROSS=22
  SHAPE_BOLD_CROSS=23
  SHAPE_THIN_CROSS=24
  SHAPE_HALF_CIRCLE_THIN_CROSS=25
  SHAPE_EMPTY=26

- note duration (Number) : selected note duration (see constants in MSDefine (page 100).mys)
  For example:
  DURATION_64TH=2520
  DURATION_32ND=DURATION_64TH*2
  DURATION_16TH=DURATION_32ND*2
  DURATION_EIGHTH=DURATION_16TH*2
  DURATION_QUARTER=DURATION_EIGHTH*2
  DURATION_HALF=DURATION_QUARTER*2
  DURATION_WHOLE=DURATION_HALF*2
  DURATION_BREVE=DURATION_WHOLE*2
  DURATION_DOUBLE_BREVE=DURATION_BREVE*2
- scroll bar (Number) : current scroll bar value
- slider (Number) : current slider value
- knob (Number) : current knob value
- list (Number) : selected line from 1 to number of lines in the list. If the list accepts multiple item selection, this value reflects the first selected.

Setting it to a value deselects all other items. To test/set multiple selection, please use DialogItem.SetExtraValue(...) (page 514)

- text (String): the text content.

Be careful not to confuse between item "name", "caption" and "value"
"Name" is the internal name that has been given to this item. It is never displayed.
"Caption" is the fixed text that is displayed in an item, for example "OK" on a button. Caption cannot be changed by the user.
"Value" is what can be changed by the use in this item (for example, state of a check box, content of an edit text..
.

Value can only be get or set once the dialog has been actually displayed, either :
- when created using CreateDialog(...) (page 496),
- or after Dialog.Show(...) (page 503) has been called

- or Dialog.Select(...) (page 503) has been called at least once.
Otherwise, the returned value is nil, and setting this value has no effect.

DialogItem.IsActive

Type : Boolean
Description : true if the dialog item is currently active
Note : This member can only be get or set once the dialog has been actually displayed, either :
- when created using CreateDialog(...) (page 496),
- or after Dialog.Show(...) (page 503)] has been called
- or Dialog.Select(...) (page 503)] has been called at least once.
Otherwise, the returned value is nil, and setting this value has no effect.

DialogItem.IsVisible

Type : Boolean
Description : true if the dialog item is currently visible
Note : This member can only be get or set once the dialog has been actually displayed, either :
- when created using CreateDialog(...) (page 496)],
- or after Dialog.Show(...) (page 503)] has been called
- or Dialog.Select(...) (page 503)] has been called at least once.
Otherwise, the returned value is nil, and setting this value has no effect.

DialogItem.IsDefault

Type : Boolean
Description : true if the dialog item is the default one for validating (Return key)
Note : - Only one item can be default
- Usually, only a push button can be set as default

DialogItem.IsCancel

Type : Boolean
Description : true if the dialog item is the default one for canceling (Esc key)
Note : - Only one item can be alias for cancel
- Usually, only a push button can be set as cancel alias

4.6.2.1.3. Managing extra values

These members can only be get or set once the dialog has been actually displayed, either :
- when created using CreateDialog(...) (page 496)],
- or after Dialog.Show(...) (page 503)] has been called
- or Dialog.Select(...) (page 503)] has been called at least once.
Otherwise, the returned value is nil, and setting this value has no effect.

DialogItem.NumberOfExtraValues

Type : Number
Description : Number of extra values for this item (see DialogItem.GetExtraValue(...) (page 513) and DialogItem.SetExtraValue(...) (page 514) mehods)

DialogItem.UserTable

Type : Table
Description : User table, provided to store private data in an item
Example :

```
-- Storing a private value
myItem.UserTable["myExtraData"]="This is an extra string for this item"
-- Retrieving a private value
str=myItem.UserTable["myExtraData"]
print(str)
```

4.6.2.1.4. Help

DialogItem.ContextHelp

Type : Styled string
Description : Item help line displayed when the mouse cursor is in the item area.

4.6.2.1.5. Radio button / Group parameters

DialogItem.FirstInGroup

Type : DialogItem
Description : First radio button in the group
Note : This member can only be used on a "frame" or "radio button" dialog item type

DialogItem.LastInGroup

Type : DialogItem
Description : Last radio button in the group
Note : This member can only be used on a "frame" or "radio button" dialog item type

DialogItem.NextInGroup

Type : DialogItem
Description : Next radio button in the same group
Note : This member can only be used on a "radio button" dialog item type

DialogItem.PreviousInGroup

Type : DialogItem
Description : Previous radio button in the same group
Note : This member can only be used on a "radio button" dialog item type

DialogItem.Group

Type : DialogItem
Description : Item group that contains the current radio button
Note : This member can only be used on a "radio button" dialog item type

DialogItem.IndexInGroup

Type : Number
Description : Index of the radio button in its group
Note : This member can only be used on a "radio button" dialog item type

4.6.2.1.6. Text item

DialogItem.RelatedText

Type : Text (page 578)Boolean
Description : Text related to the item.true if the list allows multiple item selection

4.6.2.1.7. Display

DialogItem.GraphContext

Type : Graph (page 516)
Description : Graphical access to the dialog which include the dialog..

4.6.2.2. DialogItem methods

4.6.2.2.1. Graphic

DialogItem.Update(...)

Description : Update (redraw) the content of an item.
Parameters : None
Return value : None

DialogItem.DrawRawTable(...)

Description : Draw a rawtable in the content of an item as a curve. (experimental)
Parameters : None
Return value : None

DialogItem.Scroll(...)

Description : Scroll the content of an item (for list, text, scroll, notehead, duration, slider or knob item type)
Parameters : Number: vertical scroll amount
Optional Number: horizontal scroll amount
Return value : None
Note : [.] amounts have to be positive for scrolling up or left (moving to the bottom or the right)
[.] This method is especially useful to manage mouse wheel in floating palettes.
Writing :

```
function OnMouseWheel(dialog,score,amount)
item=dialog.GetItemUnderMouse()
if item then item.Scroll(-amount) end
end
```

as a dialog method will manage vertical scroll through mouse wheel on all items that support this feature.

4.6.2.2.2. Managing extra values

DialogItem.GetExtraValue(...)

Description : Some items needs extra values to be properly managed.
Number and type of these extra values depends on the kind of dialog item.
Generally, these extra values may be set during the processing of the item's Init method.

· A scroll bar has 2 extra values (index : 1 and 2): minimum value (Number) and maximum value (Number) of this scroll bar
· A slider has 2 extra values (index : 1 and 2): minimum value (Number) and maximum value (Number) of this slider
· A knob has 2 extra values (index : 1 and 2): minimum value (Number) and maximum value (Number) of this knob
· A pop-up menu has an undefine'd number of extra values (String), each of them being the displayed text for the menu option.
· A note duration selector has an undefined number of extra values (Number), each of them being the duration (see constants in MSDefine (page 100).mys) that can be selected.
For example:
DURATION_64TH=2520
DURATION_32ND=DURATION_64TH*2
DURATION_16TH=DURATION_32ND*2
DURATION_EIGHTH=DURATION_16TH*2
DURATION_QUARTER=DURATION_EIGHTH*2
DURATION_HALF=DURATION_QUARTER*2
DURATION_WHOLE=DURATION_HALF*2
DURATION_BREVE=DURATION_WHOLE*2
DURATION_DOUBLE_BREVE=DURATION_BREVE*2

· A list has an undefined number of extra values (String), each of them being the displayed text for the given line.
· A text has one extra value : the related Text (page 578).

For example :
```
texte=item.GetExtraValue(1)
print(texte.Data)
```

is identiqual to
```
print(item.Value)
```

Other kind of items do not need any extra value for their internal processing.
Parameters : index of the extra value to get
Return value : Value (type depends on the kind of embedding object) or nil.
If the item is a list, also returns the select state of the cell. It's the only way to test multiple selection when this feature is enabled through [JDialogItem.SetExtraValue(...)].
Example :
```
minScrollValue=mySrollBar.GetExtraValue(1)
```
Note : his method can only be called after the dialog has been actually displayed, either :
- when created using CreateDialog(...) (page 496),
- or after Dialog.Show(...) (page 503) has been called
- or Dialog.Select(...) (page 503) has been called at least once.

Otherwise, the returned value is nil.

DialogItem.SetExtraValue(...)

Description : Some items needs extra values to be properly managed.
Number and type of these extra values depends on the kind of dialog item.
Generally, these extra values may be set during the processing of the item's Init method.

· A scroll bar has 2 extra values (index : 1 and 2): minimum value (Number) and maximum value (Number) of this scroll bar
· A slider has 2 extra values (index : 1 and 2): minimum value (Number) and maximum value (Number) of this slider
· A knob has 2 extra values (index : 1 and 2): minimum value (Number) and maximum value (Number) of this knob
· A pop-up menu has an undefined number of extra values (String), each of them being the displayed text for the menu option. See note 1.
· A note duration selector has an undefined number of extra values (Number), each of them being the duration (see constants in MSDefine (page 100).mys) that can be selected.
For example:
DURATION_64TH=2520
DURATION_32ND=DURATION_64TH*2
DURATION_16TH=DURATION_32ND*2
DURATION_EIGHTH=DURATION_16TH*2
DURATION_QUARTER=DURATION_EIGHTH*2
DURATION_HALF=DURATION_QUARTER*2
DURATION_WHOLE=DURATION_HALF*2
DURATION_BREVE=DURATION_WHOLE*2
DURATION_DOUBLE_BREVE=DURATION_BREVE*2
· A list has an undefined number of extra values (String), each of them being the displayed text for the given line. An extra parameter can then be provided. It controls the selection of each cell in case of multiple selection.
· A tabView has an undefined number of extra values (String), each of them being the displayed text for the given tab.

A text has one extra value : the related Text (page 578), but this extra value cannot be changed.
Other kind of items do not need any extra value for their internal processing.
Parameters : index of the extra value to set, value to set.
For lists, optional extra Boolean value to set/clear the select state of the cell
Except for scroll bar, value can be nil. In this case, the matching value in the extra value list is deleted, and the following values are shifted to the bottom of the list to fill in this empty space.
Return value : None
Example :

```
-- scroll bar values will range from 1 to 100
	mySrollBar.SetExtraValue(1,1)
	mySrollBar.SetExtraValue(2,100)
```

Note : his method can only be called after the dialog has been actually displayed, either :
- when created using CreateDialog(...) (page 496),
- or after Dialog.Show(...) (page 503) has been called
- or Dialog.Select(...) (page 503) has been called at least once.

Otherwise, calling this method has no effect.

Note 1- When setting a menu item title, some characters can be inserted at the beginning of the text to customise the entry:
[.] "(" : the entry will be disabled (grayed out)
[.] "*" : the entry will be checked (check mark in front)
Can also be done through "!"..strchar(18)
[.] "<B" the entry will be displayed bold
[.] "<I" the entry will be displayed italicized
[.] "<O" the entry will be displayed outlined
[.] "<S" the entry will be displayed shadowed
[.] "<U" the entry will be displayed underlined
[.] "/x" where x is any character will display the matching keyboard shortcut Ctrl (or command)+ character
[.] "-" the entry will be a separator line

DialogItem.DeleteExtraValues(...)

Description : Deletes all the extra values for a given dialog item.
This method is only applicable to items that needs a list of parameters as extra values :
- Note duration selector
- Pop-up menu
- List

Parameters : None
Return value : None
Note : Because default extra values are set to a Note duration selector, this method must be called prior setting your own list of values in this kind of item.

4.6.2.2.3. Managing items

DialogItem.Duplicate(...)

Description : Create a new item as a copy of an existing one
Parameters : Optional: dialog to copy item to (default: same dialog)
Optional: x offset , y offset
Return value : New item

DialogItem.Delete(...)

Description : Deletes the item
Parameters : none
Return value : none
Example :

```
myItem.Delete()
```

4.6.3. Graph Object Type

General description of this object type : The Graph object enables to access graphically to the content of a window.
For exemple this enables to display the content of an user item you previously defined in the dialog (page 494) object. Yo
can draw in a TextWindow object using the TextWindow.GraphContext (page 561) value.
You can create a picture object and add it to the score (page 198).

WARNING: When using Graph as an object, you perform the graphical operations on the default graphic context.
This default graphic context cannot be determined with certainty in most cases. It is therefore more than recommended
to use
an explicit graphic context (for instance TextWindow.GraphContext (page 561)) object instead of "Graph" for any operatic
thatr actually draws something.
However methods that do not involve actual drawing can be safely invoked through the generic Graph object.

Drawing in an user item

When an user item needs to be redrawn, its area is first erased with the standard background pattern, then the item "
Draw" method is called.
By default, the default graphic settings are :

- the drawing origin is the left top edge of the item area (the (0,0) coordinate)
- the drawing is clipped to the item area so you can't draw outside an item
- the forecolor is set to Black
- the backcolor is set to White
- the pen location is (0,0)
- the pen size is (1,1)
- the pen mode is "copy"
- the pen pattern is black
- the font is set to geneva, size 10, no face.
- the text mode is set to srcOr

Your "Draw" method does not have to preserve the graphic settings.
You can use in your "Draw" method the direct access to the current Graph object or use the Graph object related to
the item.
For example :

```
item.GraphContext.InvertRect(0,0,100,100)
```

is identical to :

```
Graph.InvertRect(0,0,100,100)
```

Drawing in a Dialog

You can get the current graphic context of a dialog using the Dialog.GraphContext (page 497) value, or the
DialogItem.GraphContext (page 511).
In this case the drawing origine is the left top edge of the dialog and the clipping area is the dialog window area.

Drawing in a TextWindow

You can get the current graphic context of a dialog using the Dialog.GraphContext (page 497) value, or the
DialogItem.GraphContext (page 511).
In this case the drawing origine is the left top edge of the window and the clipping area is the window area.

Creating a Picture object

In this case, all drawing operations will be recorded in a Picture (page 358) and added to the score. First, you activate
recording process using the Staff.BeginRecord(...) (page 288) method, you draw the picture, and you add the picture
using Staff.EndRecord(...) (page 289) method.
As the commands are recorded instead of the bitmap result, the object use very little amount of memory and can be
scaled without distorsion.

Creating a Surface object

A Surface (page 535) object is a pixel array on which many operations can be performed. It supports opacity operation alpha values) and is completely managed offscreen, which means you can perform graphical operations on such an object without displaying anything on screen. Because a Surface is a pixel-based object, scaling might result in aliasing. A Graph object can be retrieved from the Surface, in order to let any of the usual drawing functions be performed on a Surface.
However, because Graph operations do not manage alpha layer (opacity), these functions will let this layer unchanged.
Once all drawings and other graphical operations have been performed on a Surface, it can be either transferred to a Graph through Graph.DrawSurface(...) (page 529), or saved to a graphic file before being disposed.
Please read the Surface (page 535) chapter for more information.

Update consideration

Some operating systems (like Mac OS X or Wacam on Windows) use offscreen drawing. This means the graphic operations are not applied directly to the visible screen but to a hidden area called **offscreen**. Then, this area is transfered to the visible screen. The transfer occurs when a window update is requested. In some case you might need to display the offscreen immediately. Do do this, use the Graph.Update(...) (page 534) method.

Example :

```
---- Testing miscellaneous drawing methods
local x,y
local xSize,ySize
time=Application.TickCount()

---We create the winow only if it don't exist
title="Graph object demo"
MyTextWindow=FindTextWindowByTitle(title)
MyWindow=FindTextWindowByTitle(title)
if(MyWindow==nil) then
   MyWindow=NewTextWindow(title)
end
MyWindow.WindowWidth=400
MyWindow.WindowHeight=400
MyWindow.GraphContext.MoveTo(0,0);
xSize=MyWindow.WindowWidth
ySize=MyWindow.WindowHeight
--- Draw 1000 random Lines
for i=0,1000 do
   MyWindow.GraphContext.ForeColor(random(0,255))
   MyWindow.GraphContext.LineTo(random(0,xSize),random(0,ySize));
   Graph.Update();
end
--- Test user break request
Application.UserBreak()
MyWindow.GraphContext.EraseRect(0,0,xSize,ySize);

--- Draw 1000 random Rectangles
for i=0,1000 do
   MyWindow.GraphContext.ForeColor(random(0,255))
   x=random(0,xSize)
   y=random(0,ySize)
   MyWindow.GraphContext.FrameRect(x,y,x+random(0,100),y+random(0,100));
   Graph.Update();
end
--- Test user break request
Application.UserBreak()
--- Invert 1000 random Rectangles
for i=0,1000 do
   MyWindow.GraphContext.ForeColor(random(0,255))
   x=random(0,xSize)
   y=random(0,ySize)
   MyWindow.GraphContext.InvertRect(x,y,x+random(0,100),y+random(0,100));
   Graph.Update();
end
--- Test user break request
Application.UserBreak()
MyWindow.GraphContext.EraseRect(0,0,MyWindow.WindowWidth,MyWindow.WindowHeight);
--- Paint 1000 random Rectangles
for i=0,1000 do
   MyWindow.GraphContext.ForeColor(random(0,255))
   x=random(0,xSize)
   y=random(0,ySize)
   MyWindow.GraphContext.PaintRect(x,y,x+random(0,100),y+random(0,100));
   Graph.Update();
end
--- Test user break request
Application.UserBreak()
MyWindow.GraphContext.EraseRect(0,0,MyWindow.WindowWidth,MyWindow.WindowHeight);
--- Draw 1000 random Circles
for i=0,1000 do
   MyWindow.GraphContext.ForeColor(random(0,255))
   x=random(0,xSize)
   y=random(0,ySize)
   MyWindow.GraphContext.FrameOval(x,y,x+random(0,100),y+random(0,100));
   Graph.Update();
end
--- Test user break request
Application.UserBreak()
MyWindow.GraphContext.EraseRect(0,0,MyWindow.WindowWidth,MyWindow.WindowHeight);
--- Draw 1000 strings
```

```
MyWindow.GraphContext.TextMode(1) --- srcor mode
for i=0,1000 do
   MyWindow.GraphContext.ForeColor(random(0,255))
   x=random(0,xSize)
   y=random(0,ySize)
   MyWindow.GraphContext.MoveTo(x,y);
   MyWindow.GraphContext.TextSize(random(12,24));
   MyWindow.GraphContext.DrawString("MyrScript");
   Graph.Update();
end
--- Test user break request
Application.UserBreak()
MyWindow.GraphContext.EraseRect(0,0,MyWindow.WindowWidth,MyWindow.WindowHeight);
MyWindow.GraphContext.ForeColor(0)
MyWindow.Write("Total duration :"..((Application.TickCount()-time)/60).." seconds")
--- Test user break request
Application.UserBreak()
```

4.6.3.7. Graph values

4.6.3.7.1. Graph Metrics

Graph.AreaLeft

Type : Number
Description : Graph area left position in pixels

Graph.AreaTop

Type : Number
Description : Graph area top position in pixels

Graph.AreaRight

Type : Number
Description : Graph area right position in pixels

Graph.AreaBottom

Type : Number
Description : Graph area bottom position in pixels

4.6.3.8. Graph methods

4.6.3.8.1. Pen

Graph.MoveTo(...)

Description : Moves the pen to the specified location, in local coordinate.This does no drawing.
Parameters : Number, x position. Number, y position
Return value : None
Example :

```
myGraphContext.MoveTo(0,0)
```

Graph.Move(...)

Description : Moves the pen a specified distance from its current location.This does no drawing.
Parameters : Distance, in pixels, to move the pen horizontally. Positive values moves to the right; negative to the left.
Distance, in pixels, to move the pen vertically. Positive values moves down; negative move up.
Return value : None

Graph.GetPen(...)

Description : provide the current pen position, in local coordinates
Parameters : None
Return value : Horizontal position, vertical position
Example :

```
x,y=myGraphContext.GetPen()
```

Graph.PenSize(...)

Description : Select the height and width, in pixel, of the graphics pen. The inital value is (1,1)
Parameters : Desired width and height of the pen. A seeting of (0,0) makes the pen invisible.
Return value : None
Example :

```
myGraphContext.PenSize(2,2)
```

Graph.PenNormal(...)

Description : Resets the pen size, pattern, mode to the initial state :
size is set to (1,1), mode to "copy", pattern to black
Parameters : None
Return value : None
Example :

```
myGraphContext.PenNormal()
```

Graph.PenMode(...)

Description : Selects the pattern transfer mode to be used in pen drawing and painting operations.
Parameters : A number :
8 (patCopy) overwrite background enirely
9 (patOr) overwrite where pattern is black
10 (patXor) invert where pattern and bacjground are both black
11 (patBic) force white where pattern is black
Return value : None
Example :

```
myGraphContext.PenMode(8)
```

Graph.PenPat(...)

Description : Set the pen pattern, which is used by all subsequent drawing and painting operation.
Parameters : A string:
either a predefined value: white, gray, ltGray, dkGray or black
or an 8-bytes hexadecimal: 0x...
or a 64-bits binary value: 0b...
Return value : None
Example :

```
myGraphContext.PenPat("gray")
```

4.6.3.8.2. Background

Graph.BackNormal(...)

Description : Resets the background pattern and color to the standard drawing state :
pattern is white, color is white
Parameters : None
Return value : None
Example :

```
myGraphContext.BackNormal()
```

Graph.BackPat(...)

Description : Sets the background pattern, which is used by all subsequent erasing operation.
Parameters : A string :
A string:
either a predefined value: white, gray, ltGray, dkGray or black
"white" means the background pattern will be made of plain background color
"black" means the background pattern will be made of plain foreground color
gray pattens will set the background pattern to a mixing between background and foreground color
"default" is the current setting of the background dialog box, depending of the operating system and the current theme.
or an 8-bytes hexadecimal: 0x...
or a 64-bits binary value: 0b...
Return value : None
Example :

```
myGraphContext.BackPat("white")
```

Note : Graph.BackColor(...) (page 522) sets the background pattern to "white" (full background color). Thus, you must call this method BEFORE setting a pattern.

4.6.3.8.3. Color

Graph.IndexToRGB(...)

Description : Convert a color index to a RGB string (red, green, blue) string. The RGB string is a 6 character hexadecimal string build as this : "rrggbb" where rr is the red component from 00 to FF gg is the green component from 00 to FF, bb is the blue component from 00 to FF.
Parameters : A color index from 0 to 255
Return value : A RGB string
Example :

```
local i
i=0
while i<256 do
   print("The RGB color of the color index "..i.." is "..Graph.IndexToRGB(i))
   i=i+1
end
```

Graph.RGBToIndex(...)

Description : Convert a RGB string (red, green, blue) string to a color index which can be used by Graph.ForeColor(...) (page 522) and Graph.BackColor(...) (page 522). The RGB string is a 6 character hexadecin string build as this : "rrggbb" where rr is the red component from 00 to FF gg is the green component from 00 to FF, bb is the blue component from 00 to FF.
Parameters : A RGB string
Return value : A color index from 0 to 255
Example :

```
local color="FF8212"
print("The index of "..color.." is "..Graph.RGBToIndex(color))
```

Graph.RGBToComponents(...)

Description : Convert a RGB string (red, green, blue) string to a triplet of values (from 0 to 255).
The first one is the red component, then the green and blue component.
Parameters : A RGB string
Return value : Number: red
Number: green
Number: blue
Example :

```
local color="FF8212"
print("Components of "..color.." are "..Graph.RGBToComponents(color))
```

Graph.ComponentsToRGB(...)

Description : Convert a triplet of components (red, green, blue) into an RGB string
Parameters : Number: red
Number: green
Number: blue
Return value : A RGB string
Example :

```
local red=100
local green=12
local blue=200
print("RGB String for",red,green,blue,"is",Graph.ComponentsToRGB(red,green,blue))
```

Graph.ForeColor(...)

Description : Selects the color to be used in foreground drawing.
Parameters : A color index from 0 to 255. You can use RGBToIndex to convert an RGB color to the index.
Return value : None
Example :

```
myGraphContext.ForeColor(Graph.RGBToIndex("FF0000")) -- Set the forecolor to red
```

Graph.BackColor(...)

Description : Selects the color to be used in background drawing. You can use RGBToIndex to convert an RGB color to the index.
Parameters : A color index from 0 to 255
Return value : None
Example :

```
myGraphContext.BackColor(Graph.RGBToIndex("808080")) -- Set the backcolor to gray
```

Note : Sets the background pattern to white in order to make the background color viewable.

Graph.RGBForeColor(...)

Description : Selects the color to be used in foreground drawing.
Parameters : A RGB string.
Return value : None
Example :

```
myGraphContext.RGBForeColor("FF0000") -- Set the forecolor to red
```

Graph.RGBBackColor(...)

Description : Selects the color to be used in background drawing. You can use RGBToIndex to convert an RGB color to the index.
Parameters : A RGB string.
Return value : None
Example :

```
myGraphContext.RGBBackColor("808080") -- Set the backcolor to gray
```

Note : Sets the background pattern to white in order to make the background color viewable.

Graph.GetPixel(...)

Description : Gets the pixel value in RGB
Parameters : x and y position in Graph
Return value : RGB Color string
Example :

```
print(myGraphContext.GetPixel(1,1)) -- prints pixel color at position 1,1
```

4.6.3.8.4. Line

Graph.LineTo(...)

Description : Draws a line from the current pen location to the specified location.
Parameters : Number, x position. Number, y position
Return value : None
Example :

```
myGraphContext.LineTo(15,62)
```

Graph.Line(...)

Description : Moves the pen a specified distance from its current location, drawing a line as it is moved.
Parameters : Number, x distance. Number, y distance
Return value : None

Graph.CurveTo(...)

Description : Draws a cubic bezier spline from the pen current location to the given point, drawing a curve.
Parameters : Number: x position of the origin control point
Number: y position of the origin control point
Number: x position of the target control point
Number: y position of the target control point
Number: x position of the target point
Number: y position of the target point
Optional: if the curve is not collected into a polygon but actually drawn, you can specify:
Number: Line cap type (see MSDefine) CAP_ROUND, CAP_SQUARE or CAP_BUTT (default)
Return value : None

4.6.3.8.5. Rectangles

Graph.EraseRect(...)

Description : Fills a rectangle with the background pattern.
Parameters : Numbers : Left position, top position, right position, bottom position.
Return value : None
Example :

```
myGraphContext.EraseRect(0,0,item.AreaWidth,item.AreaHeight)
```

Graph.FrameRect(...)

Description : Draws an outline just inside of the borders of a rectangle.
Parameters : Numbers : Left position, top position, right position, bottom position of the rectangle.
Return value : None
Example :

```
myGraphContext.FrameRect(0,0,item.AreaWidth,item.AreaHeight)
```

Graph.InvertRect(...)

Description : Invert all pixels in the specified rectangle.
Parameters : Numbers : Left position, top position, right position, bottom position.
Return value : None
Example :

```
myGraphContext.InvertRect(0,0,item.AreaWidth,item.AreaHeight)
```

Graph.PaintRect(...)

Description : Fills a rectangle with the current pen pattern, using the current pen transfer mode.
Parameters : Numbers : Left position, top position, right position, bottom position.
Return value : None
Example :

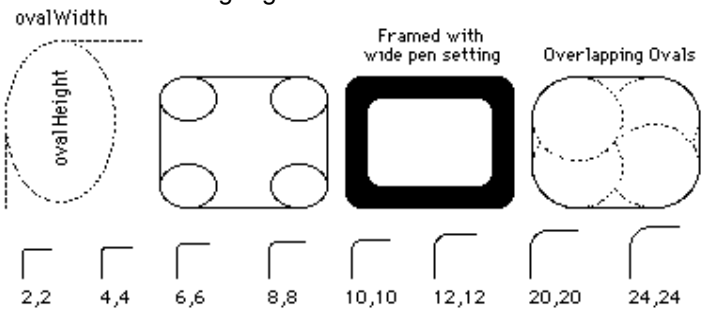
```
myGraphContext.PaintRect(0,0,item.AreaWidth,item.AreaHeight)
```

Graph.CopyRect(...)

Description : Copy a rectangle from another Graph object
Parameters : Graph: source graph object to copy from
Optional:
Numbers : Left position, top position, right position, bottom position in source graph (default: whole graph)
Numbers : Left position, top position, right position, bottom position in target graph (default: same as source)
Return value : None
Note : if the target graph belongs to a Surface (page 535), the opacity values remain unchanged.

4.6.3.8.6. Rounded Rectangles

Drawing a rounded-rectangle is similar to drawing a rectangle, then drawing four identical ovals to replace the rectangle's corners. You can see below a gauge for various curvatures.



Graph.EraseRoundRect(...)

Description : Fills a rounded-corner rectangle with the background pattern.
Parameters : Numbers : Left position, top position, right position, bottom position of the rectangle.
Optional : oval width, oval height (by default 8,8)
Return value : None
Example :

```
myGraphContext.EraseRoundRect(0,0,item.AreaWidth,item.AreaHeight)
```

Graph.FrameRoundRect(...)

Description : Draws an outline just inside of the borders of a rounded-corner rounded-corner rectangle.
Parameters : Numbers : Left position, top position, right position, bottom position of the rectangle.
Optional : oval width, oval height (by default 8,8)
Return value : None
Example :

```
myGraphContext.FrameRoundRect(0,0,item.AreaWidth,item.AreaHeight,4,4)
```

Graph.InvertRoundRect(...)

Description : Invert all pixels in the specified rounded-corner rectangle.
Parameters : Numbers : Left position, top position, right position, bottom position of the rectangle.
Optional : oval width, oval height (by default 8,8)
Return value : None
Example :

```
myGraphContext.InvertRoundRect(0,0,item.AreaWidth,item.AreaHeight,9,4)
```

Graph.PaintRoundRect(...)

Description : Fills a rounded-corner rectangle with the current pen pattern, using the current pen transfer mode.
Parameters : Numbers : Left position, top position, right position, bottom position of the rectangle.
Optional : oval width, oval height (by default 8,8)
Return value : None
Example :

```
myGraphContext.PaintRoundRect(0,0,item.AreaWidth,item.AreaHeight)
```

4.6.3.8.7. Ovals

Graph.EraseOval(...)

Description : Fills an oval with the background pattern.
Parameters : Numbers : Left position, top position, right position, bottom position.
Return value : None
Example :

```
myGraphContext.EraseOval(0,0,item.AreaWidth,item.AreaHeight)
```

Graph.FrameOval(...)

Description : Draws an outline just inside of the borders of an oval.
Parameters : Numbers : Left position, top position, right position, bottom position.
Return value : None
Example :

```
myGraphContext.FrameOval(0,0,item.AreaWidth,item.AreaHeight)
```

Graph.InvertOval(...)

Description : Invert all pixels in the specified oval.
Parameters : Numbers : Left position, top position, right position, bottom position.
Return value : None
Example :

```
myGraphContext.InvertOval(0,0,item.AreaWidth,item.AreaHeight)
```

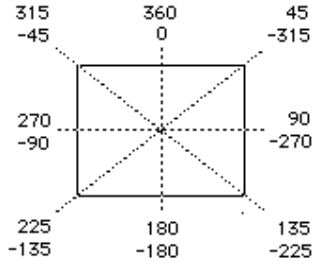
Graph.PaintOval(...)

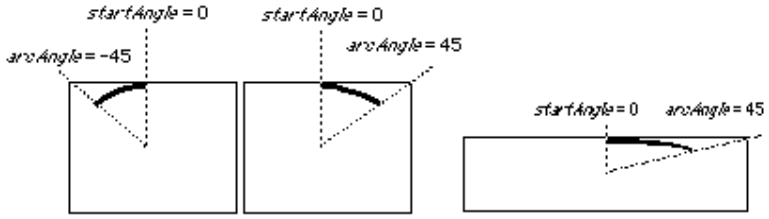
Description : Fills an oval with the current pen pattern, using the current pen transfer mode.
Parameters : Numbers : Left position, top position, right position, bottom position.
Return value : None
Example :

```
myGraphContext.PaintOval(0,0,item.AreaWidth,item.AreaHeight)
```

4.6.3.8.8. Arcs

Arc functions need six parameters : the area of the rectangle enclosing the arc, the start angle, the arc angle.
·The start angle is treated as an angle, modulo 360. it indicates the starting position for the arc, in "rectangular degrees".
Positive values are clockwise from 0 (at 12 o'clock). Negative values are counterclockwise.
·The arc angle specifies the amount of the arc to draw, in "rectangular degrees". See figure below :





Positive values for arc angle cause the arc to be drawn in a clockwise direction; negative values specify counterclockwise.
Notice in the rightmost figure, that if the drawing rectangle is elongated there is a significant change to the actual angle, but 45 degree still aligns with the corner of the rectangle.
The pen position is not changed by this function.

Graph.EraseArc(...)

Description : Fills wedge of an oval with the background pattern.
Parameters : Numbers : Left position, top position, right position, bottom position, start angle, arc angle.
Return value : None
Example :

```
myGraphContext.EraseArc(0,0,item.AreaWidth,item.AreaHeight,90,90)
```

Graph.FrameArc(...)

Description : Draws an outline just inside of the borders of wedge of an oval.
Parameters : Numbers : Left position, top position, right position, bottom, start angle, arc angle.
Return value : None
Example :

```
myGraphContext.FrameArc(0,0,item.AreaWidth,item.AreaHeight,90,90)
```

Graph.InvertArc(...)

Description : Invert all pixels in the specified oval.
Parameters : Numbers : Left position, top position, right position, bottom position, start angle, arc angle.
Return value : None
Example :

```
myGraphContext.InvertArc(0,0,item.AreaWidth,item.AreaHeight,90,90)
```

Graph.PaintArc(...)

Description : Fills wedge of wedge of an oval with the current pen pattern, using the current pen transfer mode.
Parameters : Numbers : Left position, top position, right position, bottom position, start angle, arc angle.
Return value : None
Example :

```
myGraphContext.PaintArc(0,0,item.AreaWidth,item.AreaHeight,90,90)
```

4.6.3.8.9. Text

Graph.TextFont(...)

Description : Set the current text font to the standard font specified. These fonts always exist on the system or are related to a very close font in the system. You can use the following constant :

newYork=2
geneva=3
venice=5
london=6
losAngeles=12
times=20
helvetica=21
courier=22
symbol=23

As the musical font used by the program is not unique, use Graph.SetMusicalFont(...) (page 527) to switch the te in musical font display.
Parameters : Number : the standard font ID
Return value : None
Example :

```
myGraphContext.TextFont(times)
```

Graph.TextFontByName(...)

Description : Set the current text font to the font specified by its name. Don't forget that all computers don't have the same fonts installed._If the font don't exist, this function do nothing. You can test if a font exist on the system using the Graph.IsFontExist(...) (below) function.
To set standard font or musical font use Graph.TextFont(...) (above) or Graph.SetMusicalFont(...) (page 527).
Parameters : String : The font name.
Return value : None
Example :

```
myGraphContext.TextFontByName("Times New Roman Bold")
```

Graph.IsFontExist(...)

Description : Checks whether a font exist in the system, this enables to choose an alternate font if the desired font does not exist. Please note : the only fonts guarantee to exist are provided by Graph.TextFont(...) (above)
Parameters : String : The font name.
Return value : True if the font exist on the system, false otherwise
Example :

```
fontname="FontWhichDontExist"
if(Graph.IsFontExist(fontname)==true) then
print(fontname.." exist")
else
print(fontname.." don't exist")
end
```

Graph.GetFontID(...)

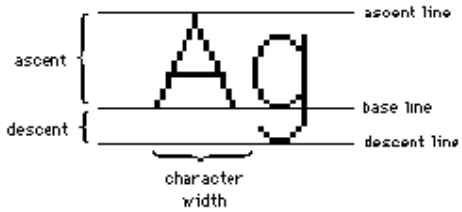
Description : Provide the unique ID of a font from his name.
Parameters : String : The font name.
Return value : Number : The font ID

Graph.GetFontName(...)

Description : Provide the name of a font from his unique ID
Parameters : Number : The font ID
Return value : String : The font name.

Graph.GetCurrentFontHeight(...)

Description : Provide the height of the current font.
Parameters : None
Return value : Number : Ascent, height above base line
Number : Descent, depth below base line
Number : Leading, distance between lines
See picture :



Example :

```
Graph.TextFontByName("Times")
Graph.TextSize(18)
Graph.TextFace(0)
ascent,descent,leading=Graph.GetCurrentFontHeight()
print("Ascent is "..ascent.." Descent is "..descent.." Leading is "..leading)
```

Graph.SetMusicalFont(...)

Description : Set the current text font to the musical font. The font size is provided to allow MyrScript to locate the more close musical font installed , this guarantee the best display quality.
Parameters : Number : The font size.
Return value : None
Example :

```
myGraphContext.SetMusicalFont(72) -- Standard height
```

Graph.TextFace(...)

Description : Selects the text face.
Parameters : A number :
0 standard face
1 **bold**
2 *italic*
4 underline
Return value : None
Example :

```
myGraphContext.TextFace(1+2) -- Set Bold+Italic
```

Graph.TextSize(...)

Description : Selects the text size.
Parameters : The size in pixel
Return value : None
Example :

```
myGraphContext.TextSize(14)
```

Graph.TextMode(...)

Description : Selects the text mode.
Parameters : A number :
0 (srcCopy) overwrite background entirely
1 (srcOr) overwrite where character is black
2 (srcXor) invert where where character is black
3 (srcBic) force white where character is black
Return value : None

Graph.DrawChar(...)

Description : The specified character is drawn using the current font, size, face, color and transfer mode. (See Graph.ForeColor(...) (page 522), Graph.TextFont(...) (page 526), Graph.TextSize(...) (page 527), Graph.TextFace and Graph.TextMode(...) (above).) The character is drawn with the left end of its baseline starting at the current pen location. The pen advances appropriately.
Parameters : The char to draw.
Optional: drawing angle (degrees)
Return value : None
Example :

```
myGraphContext.DrawChar('A')
```

Graph.CharWidth(...)

Description : Obtains the width of the specified character : the amount it would horizontally move the pen if it was displayed using the current font and size.
Parameters : The char.
Return value : None
Example :

```
print(myGraphContext.CharWidth('A'))
```

Graph.DrawRichString(...)

Description : Draw a complex string, including commands, in the specified area. The string is drawn using the current font, size, face, color and transfer mode. (See Graph.ForeColor(...) (page 522), Graph.TextFont(...) (page Graph.TextSize(...) (page 527), Graph.TextFace(...) (page 527) and Graph.TextMode(...) (above).) The pen locati moves appropriately.
Please note that, if larger than the drawing rectangle, the text is clipped. This function doesn't manage multi-line text: CRs are replaced by spaces.
Parameters : · Numbers : Left position, top position, right position, bottom position of the rectangle.
· String : The string to draw. The string can include special commands :
Return value : None
Example :

```
myGraphContext.DrawRichString(0,0,item.AreaWidth,item.AreaHeight,"$([8 8])")
```

Graph.DrawString(...)

Description : The specified string is drawn using the current font, size, face, color and transfer mode. (See Graph.ForeColor(...) (page 522), Graph.TextFont(...) (page 526), Graph.TextSize(...) (page 527), Graph.TextFace and Graph.TextMode(...) (above).) The pen advances appropriately.
Parameters : The string to draw.
Optional: drawing angle in degree
Return value : None
Example :

```
myGraphContext.DrawString("Hello world!")
```

Graph.StringWidth(...)

Description : Obtains the width of the specified string : the amount it would horizontally move the pen if it was displayed using the current font and size.
This method don't take account of line breaks inside the string.
Parameters : The string.
Return value : Number : Width of string

Graph.StringSizes(...)

Description : Obtains the width and height of the specified string if it was displayed using the current font and size.
This method take account of the line breaks to compute the block of text size
Parameters : String : The styled string (page 177)
Return value : Number : Width of string, height of string

4.6.3.8.10. Picture

Graph.DrawPicture(...)

Description : Draw a picture at the position provided. The picture is read from a picture file which can be a PICT, BMP, GIF, TIFF, JPG or PNG file.
The file is searched in the current path but you can specify a relative path in the file name.
For example :
myGraphContext.DrawPicture("BozoTheClown.bmp")
or
myGraphContext.DrawPicture("Images/DjizzTheCat.pct")
are valid.
A PICT file must have the ".pct" extension.
A BMP file must have the ".bmp" extension.
A GIF file must have the ".gif" extension.
A TIFF file must have the ".tif" extension.
A JPG file must have the ".jpg" extension.
A PNG file must have the ".png" extension.
To avoid confusion about same pictures files name used by different scripts, the use of packaged script is recommended.
If a destination rectangle is provided, the picture is scaled to fit into the destination rectangle.
If only horizontal and vertical position are provided, the picture is drawn at this position, without scaling.
If a center rectangle is provided, the picture is drawn centered on this rectangle.
If the picture file is not found a broken icon is drawn instead.
Parameters : String : the file name including optionaly relative path.
Optional :
- Numbers : Destination rectangle in pixel position. Left position, top position, right position, bottom position of the rectangle. If right and bottom are not provided or are equal, the original picture size is taken.
- Numbers : Center rectangle in pixel position . Left position , top position, right position, bottom position of the rectangle.

Return value : Boolean : true if the picture file exist, false otherwise.
Example :

```
-- This will draw a picture file in an user item, using the original picture area, centering on the item area.
myGraphContext.DrawPicture("MyPictureFile.bmp",0,0,0,0,0,0,item.AreaWidth,item.AreaHeight)
```

Graph.GetPictureSize(...)

Description : Get the size in pixels of a graphic file
Parameters : String : the file name including optionaly relative path.
Return value : X size in pixel, Y size in pixel

Graph.DrawSurface(...)

Description : Draw a Surface (page 535) object at the provided position.
Parameters : Surface: the surface object to draw
Optional :
- Numbers : Destination rectangle in pixel position. Left position, top position, right position, bottom position of the rectangle. If right and bottom are not provided or are equal to left and top, the original picture size is taken.
- Numbers : X and Y of the display center. If provided, Left and Top positions of the previous parameter set are not used.

Return value : Boolean : true if the picture file exist, false otherwise.

4.6.3.8.11. Polygon

Graph.OpenPolygon(...)

Description : he OpenPolygon method starts saving lines for processing as a polygon definition. While a polygon is open, all calls to the Line and LineTo methods affect the outline of the polygon. Only the line endpoints affect the polygon definition; the pattern mode, pattern, and size do not affect it. No drawing occurs on the screen while the polygon is open.
A polygon should consist of a sequence of connected lines. The OpenPolygon method stores the definition for a polygon in a local, unique and hidden structure referenced by an identifier. Only one polygon can be recorded at the same time.
Two polygon opened at the same time have two differents identifiers.
When a polygone is released, using the Graph.KillPolygon method, the identifier will be available for a next Graph.OpenPolygon call.
Never use a polygon indentifier which was previously released.
Even though the onscreen presentation of a polygon is clipped, the definition of a polygon is not; you can define a polygon anywhere on the coordinate plane.
When you are finished calling the line-drawing methods that define your polygon, use the ClosePolygon method.
Parameters : None
Return value : Number, Polygone identifier.
Example :

```
Include "MSDefine"
-- Create a picture and add it to the first staff of the frontmost score
myScore=FrontScore()
if myScore==nil then
   print("You must open a score first")
else
   myStaff=myScore.FirstStaff
   if myStaff==nil then
      print("There is no staff in the document")
   else
      --- Beginning the recording process
      myStaff.BeginRecord(64,64)
      --- Recording the polygon
      id=Graph.OpenPolygon()
      Graph.MoveTo(10,10)
      Graph.LineTo(5,20)
      Graph.LineTo(15,20)
      Graph.LineTo(10,10)
      -- Ending the recording
      Graph.ClosePolygon()
      -- We frame the polygon
      Graph.FramePolygon(id)
      -- We move it
      Graph.OffsetPolygon(id,30,0)
      -- We invert the polygon content
      Graph.InvertPolygon(id)
      -- The polygon is released
      Graph.KillPolygon(id)
      --- Close the recording and add the picture
      myStaff.EndRecord(100,-40)
   end
end
```

Graph.ClosePolygon(...)

Description : The ClosePolygon method stops collecting line/curve-drawing commands for the currently open polygon. You should call ClosePolygon only once for every call to the OpenPolygon method.
Warning, this function does not close graphically the polygon by drawing the missing last line, it just stops the collection of lines and curves.
Parameters : None
Return value : None

Graph.ErasePolygon(...)

Description : Erase the interior of the current polygon with the current background pattern or color
This method leaves the location of the graphics pen unchanged.
Parameters : Number: polygon identifier as returned by Graph.OpenPolygon
Return value : None

Graph.EraseCurvePolygon(...)

Description : Erase the interior of the current polygon with the current background color.
This method leaves the location of the graphics pen unchanged.
Parameters : Number: polygon identifier as returned by Graph.OpenPolygon
Optional: Boolean: true if the drawing is antialiased (smoothed). Default: true
Return value : None

Graph.FramePolygon(...)

Description : Using the current pen pattern, pattern mode, and size, the FramePolygon method plays back the line-drawing commands that define the current polygon.
The graphics pen hangs below and to the right of each point on the boundary of the polygon.
Parameters : Number: polygon identifier as returned by Graph.OpenPolygon
Return value : None
Note : [.] This method automatically closes graphically the polygon by drawing the line between the last and the first point.
[.] if the polygon contains curves, (Graph.CurveTo(...)) they are drawn as straigth lines.
To draw a curved polygon frame (or a regular polygon in antialisased mode) use the Graph.FrameCurvePolygon method.

Graph.FrameCurvePolygon(...)

Description : Draw a frame around the polygon using the current pen color and size.
Parameters : Number: polygon identifier as returned by Graph.OpenPolygon
Optional: Boolean true if the drawing is antialiased (smoothed). Default: true
Boolean true if the drawing has to be graphically closed (default), false if drawn open
Number: Line cap type (see MSDefine) CAP_ROUND, CAP_SQUARE or CAP_BUTT (default)
Number: Line join type JOIN_ROUND, JOIN_BEVEL or JOIN_MITER (default)
Number: Miter limit (default=no limit)
Return value : None

Graph.InvertPolygon(...)

Description : The InvertPolygon method inverts the pixels enclosed by the current polygon. Every white pixel becomes black and every black pixel becomes white.
This method leaves the location of the graphics pen unchanged.
Parameters : Number: polygon identifier as returned by Graph.OpenPolygon
Return value : None
Note : it's not possible to invert a polygon that contains curves. If the polygon contains such curves, (Graph.CurveTo(...)) they are drawn as straigth lines.

Graph.PaintPolygon(...)

Description : Using the pen pattern and pattern mode for the current graphics port, the PaintPolygon method draws the interior of the current polygon. The pen location does not change.
Parameters : None
Return value : None
Note : if the polygon contains curves, (Graph.CurveTo(...)) they are drawn as straigth lines.
To paint a curved polygon (or a regular polygon in antialisased mode) use the Graph.PaintCurvePolygon method.

Graph.PaintCurvePolygon(...)

Description : Paint the interior of the polygon using the current pen color.
Parameters : Number: polygon identifier as returned by Graph.OpenPolygon
Optional: Boolean true if the drawing is antialiased (smoothed). Default: true
Return value : None

Graph.OffsetPolygon(...)

Description : The Graph.OffsetPolygon method moves the current polygon by adding the value you specify to the horizontal coordinates of its points, and by adding the value you specify to the vertical coordinates of all points of its region boundary.
If the values of dh and dv are positive, the movement is to the right and down; if either is negative, the corresponding movement is in the opposite direction.
The region retains its size and shape. This does not affect the screen unless you subsequently call a method to draw the region.
OffsetPolygon is an especially efficient operation, because the data defining a polygon is stored relative to the first point of the polygon and so is not actually changed by OffsetPoly.
Parameters : Numbers, the horizontal distance to move the polygon, the vertical distance to move the polygon.
Return value : None

Graph.KillPolygon(...)

Description : Use KillPolygon only when you are completely through with a polygon. KillPolygon free all data created by OpenPolygon.
Parameters : None
Return value : None

4.6.3.8.12. Musical objects

Graph.DrawStaff(...)

Description : Draw a staff in the specified area. This enables to display a model reflecting some user's choice.
Parameters : Numbers : Left position, top position, right position, bottom position of the rectangle, staff to display
Optional :
- Number : time position to display
- Number : Display Scale in percent (default=100)

Return value : None
Example :
This example show the methods of an user item which display a staff with three notes.

```
-- Init : called just after the dialog has been displayed for the first time
function Init(dialog,item)
   -- Creating new score
   item.UserTable["Score"]=NewScore(false)
   score=item.UserTable["Score"]
   if(score~=nil) then
      ---Creating new staff in score
      item.UserTable["Staff"]=item.UserTable["Score"].AddStaff()
      staff=item.UserTable["Staff"]
      if(staff~=nil) then
         --- Inserting the notes
         staff.InsertNote(0,DURATION_QUARTER,5*12)
         staff.InsertNote(DURATION_QUARTER,DURATION_QUARTER,5*12+2)
         staff.InsertNote(2*DURATION_QUARTER,DURATION_QUARTER,5*12+4)
         end
      end
end
-- Draw : called each time the user item need to be redraw
function Draw(dialog,item)
   Graph.FrameRect(0,0,item.AreaWidth,item.AreaHeight)
   Graph.DrawStaff(0,0,item.AreaWidth,item.AreaHeight,item.UserTable["Staff"])
end
-- Exit : called just before the dialog is closed
function Exit(dialog,item)
   -- Disposing score
   item.UserTable["Score"].Dispose()
end
```

4.6.3.8.13. Clipping

When your draw in an item using the items's "Draw" method, drawing outside the item area are not applied.
This is nammed "clipping". The clipping methods of the Graph object allow you to preserve and change this clipping area.

Graph.GetClipRect(...)

Description : Provide the current clipping area.
Parameters : None
Return value : Numbers : Left position, top position, right position, bottom position of the area in the dialog.

Graph.SetClipRect(...)

Description : Change the current clipping area.
Parameters : Numbers : Left position, top position, right position, bottom position of the area in the dialog.
Return value : None

4.6.3.8.14. Palette button

Graph.DrawPaletteButton(...)

Description : Draw a palette button following the current application appearance
Parameters : Numbers : Left position, top position, right position, bottom position of the rectangle.
Optional : Boolean button state true if selected, false by default
Optional : Undermouse state. By default the current state.
Return value : None

4.6.3.8.15. Pict file

Graph.NewPictFile(...)

Description : Start collection of graphical operations to be saved in a graphic file
This pict file can be drawn later on any graph object through Graph.DrawPicture(...) (page 529)
Parameters : Optional : Numbers : left, top, right and bottom of the bounding rectangle (default: 0,0,100,100)
Return value : The Graph Object

Graph.WritePictFile(...)

Description : Stops collection of graphical operations and save them to a graphic file
Parameters : String : Pict file name. Depending on the file extension, the file will be saved in one of the following formats:
[.] PICT (.pct extension) Mac Pict format (vector graphics)
[.] FPCT (.fpct extension) Myriad Pict format (vector graphics, proprietary format)
[.] BMP (.bmp extension)
[.] GIF (.gif extension)
[.] TIFF (.tiff extension)
[.] JPEG (.jpg extension)
[.] PNG (.png extension)

Optional: Number: Quality (0, 1 or 2) only useful if the export format is JPG or BMP
Return value : true if the graphic file could be properly saved, nil otherwise

4.6.3.8.16. Sizes

Graph.ScreenSize(...)

Description : Provide the screen size, in pixel.
Parameters : None
Return value : Horizontal size of the screen, Vertical size of the screen.
Example :

```
x,y=Graph.ScreenSize()
Message("Your screen size is : "..x.." x "..y)
```

Graph.MenuBarHeight(...)

Description : Provide the menu height. Use this function when you move windows on the screen, to avoid moving under the menu bar
Parameters : None
Return value : Vertical size of the menu bar.
Example :

```
Message("The menu bar height is : "..Graph.MenuBarHeight().." pixels")
```

4.6.3.8.17. Miscellaneous

Graph.Update(...)

Description : When drawing occurs on a an offscreen window, transfer the hidden offscreen on the screen
Parameters : None
Return value : None

4.6.4. Surface Object Type

General description of this object type : Pixel surface for high-level graphical operations.
Each pixels use 24 bits for color information and 8 bits for opacity.
A Surface must be created by Surface.New(...) (page 536) (access to the global generic object) and disposed by the Surface.Dispose(...) (page 537) function.

4.6.4.1. Surface values

4.6.4.1.1. Surface Metrics

Surface.Width (R)

Type : Number
Description : Surface Width in pixels

Surface.Height (R)

Type : Number
Description : Surface Height in pixels

Surface.GraphContext (R)

Type : Graph (page 516)
Description : Graph object that matches the surface.
WARNING! The graph object remains valid only until an operation changes the size of the surface (resize, crop ..)
It is therefore not recommended to store the graph object that matches a surface into a local variable.
To avoid any possible error, prefer for instance:
mySurface.GraphContext.MoveTo(x,y)
to
myGraph=mySurface.GraphContext
myGraph.MoveTo(x,y)
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
sgr=s.GraphContext
sgr.MoveTo(40,40)
sgr.TextSize(20)
sgr.DrawString("To be or not to be a cat?")
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

4.6.4.2. Surface methods

4.6.4.2.1. Create/Dispose a Surface

Surface.New(...)

Description : Create a new surface. This surface must be disposed through Surface.Dispose(...) (page 537) afte use.
Parameters : This function accepts several possible parameter sets.
1- To create an empty surface:
Number : width in pixels
Number : height in pixels
Optional:
Number : Red component of the fill color (0-255, default 0)
Number : Green component of the fill color (0-255, default 0)
Number : Blue component of the fill color (0-255, default 0)
Number : Opacity value (0-100, default 100)

2- To create a surface from a graphic file
String : complete file name of the graphic file

3- To create a copy of an existing surface
Surface : surface to copy from

4- To create surface from a Graph (page 516) context
Graph: Graph context to copy from
Return value : Surface (page 535) : added object or nil if not added
Example :

```
s=Surface.New(100,100)
if s then
   -- Create dialog to display surface
   dial=NewDialog("Surface",720,400)
   dial.Show()
   gr=dial.GraphContext
   -- display it
   gr.DrawSurface(s,5,5)
   -- relases it from memory
   s.Dispose()
   -- wait for button
   while Application.GetMouseButton()==0 do dial.Select() end
   -- dialog is disposed
   dial.Dispose()
end
```

```
-- create dialog
dial=NewDialog("Graph demo",720,400)
frame=dial.NewFrameRect(5,5,715,365)
okbut=dial.NewPushButton("OK",340,370,60,20)
dial.Show()
dial.BringToFront()
-- get dialog graph context
gr=frame.GraphContext
-- Draw lines until ok button is clicked
gr.PenNormal()
gr.MoveTo(0,0)
repeat
   gr.ForeColor(random(255))
   gr.PenSize(4,4)
   gr.LineTo(random(720),random(360))
   item=dial.Select(0)
until item==okbut
-- Create a surface to get the dialog content
s=Surface.New(gr)
-- Save the surface
fname=GetUserExportPathName("Graphics Export").."dialogcontent.png"
s.SavePicture(fname)
print("Picture saved to "..fname)
-- Free all
s.Dispose()
dial.Dispose()
```

Surface.Dispose(...)

Description : Dispose the Surface
Parameters : None
Return value : None

Surface.Duplicate(...)

Description : Create a new surface which is the exact copy of the surface in parameter
Parameters : Surface to copy
Return value : New surface
Example :

```
s=Surface.New(100,100)
if s then
   -- Create dialog to display surface
   dial=NewDialog("Surface",720,400)
   dial.Show()
   gr=dial.GraphContext
   -- display it
   gr.DrawSurface(s,5,5)
   -- Duplicate it
   s2=s.Duplicate()
   -- display the copy
   gr.DrawSurface(s2,110,5)
   -- relases it from memory
   s2.Dispose()
   s.Dispose()
   -- wait for button
   while Application.GetMouseButton()==0 do dial.Select() end
   -- dialog is disposed
   dial.Dispose()
end
```

Note : It is functionally equivalent to newSurface=Surface.New(originalSurface)

4.6.4.2.2. Surface blurring and drawing

Surface.Blur(...)

Description : Bidirectional blur
Parameters : Number: radius of the blur in pixels
Return value : Boolean: true if the operation has been successfully completed
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."myriad.png") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.Blur(6)
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Note : the opacity layer remains unchanged. To blur opacity, use Surface.OpacityBlur(...) (page 544)

Surface.Rectangle(...)

Description : Fills a rectangle area with a given color and opacity
Parameters : Numbers: left,top,right,bottom of the rectangle to fill
Numbers: r,v,b components (0-255) of the rectangle color
Number: Opacity of the rectangle
Number: opacity processing mode.

If the opacity processing mode is 0 ("do not process opacity"):
- If the opacity parameter is 100 (opaque), then the rectangle is drawn with its plain color, without changing the surface opacity values.
This operation is then equivalent to Surface.GraphContext.PaintRect(...)
- If the opacity parameter is less than 100 (semi-transparent), the rectangle color is mixed with the existing pixels of the surface but the opacity value remains unchanged.

If the opacity processing mode is 1 ("merge opacity"), the rectangle color is mixed with the existing pixels of the surface according to the opacity parameter, then the surface opacity values are altered
If the opacity processing mode is 2 ("force"), the rectangle color and opacity are forced in the covered area, replacing the previous values
Return value : Boolean: true if the operation has been successfully completed
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."myriad.png") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.Rectangle(20,20,60,60, 0,255,0, 80, true)
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Surface.Embed(...)

Description : Draw a surface inside another surface.
Parameters : Surface: surface to be embedded
Number: X position where to draw the surface
Number: Y position where to draw the surface
Number: Operation mode
[.] 0 = smart copy (transparent copy w/ opacity merging)
[.] 1 = raw copy (rgb data and opacity are copied to the target)
[.] 2 = RGB copy (rgb data copied to the target, no opacity processing)
[.] 3 = RGB Add (no opacity processing)
[.] 4 = RGB Sub (no opacity processing)
[.] 5 = RGB Min (no opacity processing)
[.] 6 = RGB Max (no opacity processing)
[.] 7 = RGB Multiplication (no opacity processing)
Return value : Boolean: true if the operation has been successfully completed
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s2=Surface.New(path.."myriad.png") -- Load a second picture
s2.Scale(60,60)    -- shrink
n=0
for i=0,2 do
   for j=0,2 do
      if n<8 then
         s.Embed(s2,64*j,64*i, n) -- each transfer mode
      end
      n=n+1
   end
end
s2.Dispose()
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

4.6.4.2.3. Surface I/O

Surface.LoadPicture(...)

Description : Load a picture file in an existing surface, replacing the previous data
Parameters : String: complete filename of the picture to load
Note : this is functionnally equivalent to Surface.New("Filename");
Return value : Boolean: true if the picture file has been successfully loaded
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw surface
s.Dispose()                    -- Surface is disposed
while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Surface.SavePicture(...)

Description : Save surface to a graphic file
Parameters : String : Pict file name. Depending on the file extension, the file will be saved in one of the following formats:
[.] BMP (.bmp extension)
[.] GIF (.gif extension)
[.] TIFF (.tiff extension)
[.] JPEG (.jpg extension)
[.] PNG (.png extension)

Optional: Number: Quality (0, 1 or 2) only useful if the export format is JPG or BMP
Return value : true if the graphic file could be properly saved, nil otherwise
Example :

```
-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
s.Rotate(30)                -- Modify surface
path=GetUserExportPathName("Graphics Export") -- path to Myriad Documents/Export/Graphics Export
s.SavePicture(path.."catsout.png")          -- save surface
s.Dispose()                    -- Surface is disposed
print("See result in "..path.."catsout.png")
```

Note : Only the PNG format retains the opacity values.

4.6.4.2.4. Surface Opacity layer

Surface.OpacitySet(...)

Description : Set the values in the opacity layer to a constant
Parameters : Number: Opacity value to set (0-100)
Return value : Boolean: true if the operation has been successfully completed
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.OpacitySet(40) -- in percent
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()             -- Dialog is disposed
```

Surface.OpacityFromColor(...)

Description : The opacity layer is computed according to the rgb components of the surface :
if the color matches the rgb, then the pixel is considered as transparent
if it doesn't match, the pixel is considered as opaque
Parameters : Numbers: red, green and blue component (O-255) of the "transparent" color
Return value : Boolean: true if the operation has been successfully completed
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Erase background in purple
gr.BackNormal()
gr.RGBBackColor("806080")
gr.EraseRect(0,0,720,400)


-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.Highlight(150)
s.OpacityFromColor(255,255,255) --white becomes transparent
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Surface.OpacityFromLuminance(...)

Description : The opacity layer is computed according to the luminance of the surface pixel: rgb components of the surface : dark pixels will be transparent and bright pixels will be opaque.
Parameters : None
Return value : Boolean: true if the operation has been successfully completed
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s2=Surface.New(path.."myriad.png")       -- second surface
s2.OpacityFromLuminance()    -- white pixels are opaque, black are transparent
s2.OpacityFade(120)          -- reinforce opacity a bit
s.Embed(s2,190,110, false)
s2.Dispose()
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Surface.OpacityFade(...)

Description : Apply a constant multiplier to the opacity layer
Parameters : Number: Opacity ratio to apply in percent
Return value : Boolean: true if the operation has been successfully completed
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s2=Surface.New(path.."myriad.png")       -- second surface
s2.OpacityFade(50)                       -- weaken opacity
s.Embed(s2,190,110, false)
s2.Dispose()
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Surface.OpacityProgressiveFade(...)

Description : Apply a variable multiplier to the opacity layer
Parameters : Number: Opacity ratio to apply in the top-left corner (in percent
Number: Opacity ratio to apply in the top-right corner (in percent)
Number: Opacity ratio to apply in the bottom-left corner (in percent)
Number: Opacity ratio to apply in the bottom-right corner (in percent)
Return value : Boolean: true if the operation has been successfully completed
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s2=Surface.New(path.."myriad.png")        -- second surface
s2.OpacityProgressiveFade(0,0,100,100) -- alter opacity
s.Embed(s2,190,110, false)
s2.Dispose()
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Surface.OpacityBlur(...)

Description : Blur the opacity layer
Parameters : Number: blur radius in pixels
Optional:
Boolean: true if the pixels outside the surface are considered as opaque
false if they are considered as transparent (default)
Return value : Boolean: true if the operation has been successfully completed
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."myriad.png") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.OpacityBlur(10)
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Surface.OpacityInnerBlur(...)

Description : Blur the opacity layer, shrinking the opaque data through a smooth "inside feather" operation
Parameters : Number: blur radius in pixels
Optional:
Boolean: true if the pixels outside the surface are considered as opaque
false if they are considered as transparent (default)
Return value : Boolean: true if the operation has been successfully completed
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."myriad.png") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.OpacityInnerBlur(10)
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Surface.OpacityToLuminance(...)

Description : Transfer the opacity layer into pixel brightness. Opaque pixels become white, transparent pixels become black.
The opacity layer is set to opaque.
Parameters : None
Return value : Boolean: true if the operation has been successfully completed
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."myriad.png") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.OpacityToLuminance()
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Surface.OpacityInvert(...)

Description : Invert the opacity layer
Parameters : None
Return value : Boolean: true if the operation has been successfully completed
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."myriad.png") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.OpacityInvert()
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)       -- draw altered surface
s.Dispose()                   -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()                -- Dialog is disposed
```

Surface.OpacityCompute(...)

Description : Computation of the opacity layer
Parameters : Surface: Surface to use for computation
Number: Computation type
0= Copy - the opacity layer is copied from the parameter surface
1= Overlay - the opacity layer is combined with from the parameter surface's (addition)
2= Subtract - the opacity layer is reduced by the parameter surface's (subtraction)
3= Multiply - the opacity layer the result of the multiplication of the two layers
The original surface is modified by the operation. If the two surfaces are not the same size, their centers are aligned.
Return value : Boolean: true if the operation has been successfully completed
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s2=Surface.New(path.."myriad.png")
s2.Resize(s.Width,s.Height) -- resize to get the same size
s.OpacityCompute(s2,0) -- Copy
s2.Dispose()
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

4.6.4.2.5. Surface pixel color change

Surface.Highlight(...)

Description : Highlight the surface colors
Parameters : Number: amount of highlight in percent. 100 means no change
Return value : Boolean: true if the surface could be properly highlighted
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.Highlight(150) -- in percent
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Surface.Glow(...)

Description : Highlight the surface colors non-linearly, making the bright colors glow
Parameters : Number: amount of glow in percent. 100 means no change
Return value : Boolean: true if the surface could be properly glowed
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.Glow(250) -- in percent
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Surface.AdjustBCS(...)

Description : Change Brightness, Contrast and Saturation of the picture
Parameters : Number: Brightness amount to apply in percent from -100 to 100. 0 means no change
Number: Contrast amount in percent from -100 to 100. 0 means no change
Number: Saturation amount in percent from -100 to 100. 0 means no change
Return value : Boolean: true if the operation has completed successfully
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.AdjustBCS(-10,40,30) -- in percent
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Surface.AdjustHSL(...)

Description : Change Hue; Saturation and Lightness of the picture
Parameters : Number: Hue amount to apply in degrees on the hue circle from -180 to 180. 0 means no change
Number: Saturation amount in percent from -100 to 100. 0 means no change
Number: Lightness amount in percent from -100 to 100. 0 means no change
Return value : Boolean: true if the operation has completed successfully
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.AdjustHSL(20,30,-10) -- hue in angle, others in percent
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Surface.BrightnessCurve(...)

Description : Change pixel brightness through a curve
Parameters : Number: first input value from 0 to 255
Number: first output value from 0 to 255
Optional
Number: second input value from 0 to 255
Number: second output value from 0 to 255
...
Return value : Boolean: true if the operation has completed successfully
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.BrightnessCurve(0,0, 96,200, 160,50, 255,255) -- solarization effect
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Note : you can define any number of couples input/output you need.
Intermediate values will be calculated by linear approximation.

Surface.ComponentCurve(...)

Description : Change pixel Red, green or blue component through a curve
Parameters : Number: component to alter (1=red, 2=green, 3=blue)
Number: first input value from 0 to 255
Number: first output value from 0 to 255
Optional
Number: second input value from 0 to 255
Number: second output value from 0 to 255
...
Return value : Boolean: true if the operation has completed successfully
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.ComponentCurve(1,  0,0, 96,200, 160,50, 255,255) -- solarization effect for red
s.ComponentCurve(2,  0,0, 128,100, 255,255) -- darken green
s.ComponentCurve(3,  0,0, 128,150, 255,255) -- lighten blue
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Note : you can define any number of couples input/output you need.
Intermediate values will be calculated by linear approximation.

Surface.ChannelMixer(...)

Description : Apply an RVB channel mixer to the Surface color space.
Parameters : Number: transormed (target) red bias value (0-255)
Number: Ratio (%) of original red component in the transformed (target) red value
Number: Ratio (%) of original green component in the transformed (target) red value
Number: Ratio (%) of original blue component in the transformed (target) red value
Number: transormed (target) green bias value (0-255)
Number: Ratio (%) of original red component in the transformed (target) green value
Number: Ratio (%) of original green component in the transformed (target) green value
Number: Ratio (%) of original blue component in the transformed (target) green value
Number: transormed (target) blue bias value (0-255)
Number: Ratio (%) of original red component in the transformed (target) blue value
Number: Ratio (%) of original green component in the transformed (target) blue value
Number: Ratio (%) of original blue component in the transformed (target) blue value
Return value : Boolean: true if the operation has completed successfully
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.ChannelMixer(0, 0,100,0,   0, 100,0,0,   0, 0,0,100) -- invert red & green
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)       -- draw altered surface
s.Dispose()                   -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()                -- Dialog is disposed
```

4.6.4.2.6. Surface pixels

Surface.GetPixel(...)

Description : Get the color and opacity value of a surface pixel
Parameters : Number: x position of the pixel (from the left)
Number: y position of the pixel (from the top)
Return value : Number : red component (0-255)
Number : green component (0-255)
Number : blue component (0-255)
Number : opacity value (0-100)
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s2=s.Duplicate()
for y=0,s.Height-1 do
   for x=0,s.Width-1 do
      r,g,b,op=s2.GetPixel(x,y)
      s.SetPixel(x+20*sin(y*5),y,r,g,b,op)  -- Wave
   end
end
-- End of test routine
x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)       -- draw altered surface
s2.Dispose()              -- Surface is disposed
s.Dispose()                   -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()                -- Dialog is disposed
```

Note : if the coordinate falls outside the surface, the values 0,0,0,0 are returned

Surface.SetPixel(...)

Description : Set the color and opacity value of a surface pixel
Parameters : Number: x position of the pixel (from the left)
Number: y position of the pixel (from the top)
Number : red component (0-255)
Number : green component (0-255)
Number : blue component (0-255)
Optional:
Number : opacity value (0-100)
Return value : None
Note : if the coordinate falls outside the surface, the function does nothing
if the opacity value is omitted, the pixel opacity remains unchanged

4.6.4.2.7. Surface resizing and deformation

Surface.AddMargins(...)

Description : Add margins around the surface
Parameters : Number: number of pixels to add on the left
Number: number of pixels to add on the top
Number : number of pixels to add on the right
Number : number of pixels to add on the bottom
Optional:
Number : red component (0-255) of the added area
Number : green component (0-255) of the added area
Number : blue component (0-255) of the added area
Number : opacity value (0-100) of the added area
Return value : Boolean: true if the margins could be added properly
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.AddMargins(10,20,30,40, 100,100,0, 100) -- left,top,right,bottom, r,g,b, opacity
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Surface.Crop(...)

Description : Crop the surface to the desired rectangle
Parameters : Numbers: Crop rectangle (left,top,right,bottom)
Return value : Boolean: true if surface has been peoperly cropped
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.Crop(100,10,190,100) -- left,top,right,bottom
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Surface.Scale(...)

Description : Change the size of the picture according to the scale valyes
Parameters : Number: X scale in percent
Number: Y scale in percent
Return value : Boolean: true if the margins could be scaled properly
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.Scale(60,90) -- in %
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Surface.Resize(...)

Description : Resize the surface to the desired rectangle
Parameters : Number: new surface width in pixels
Number: new surface height in pixels
Return value : Boolean: true if surface has been properly resized
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.Resize(168,168) -- in pixels
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Note : if non-integer

Surface.Flip(...)

Description : Flip the surface vertically
Parameters : None
Return value : Boolean: true if the operation is successfull
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.Flip()
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Surface.Mirror(...)

Description : Flip the surface horizontally
Parameters : None
Return value : Boolean: true if the operation is successfull
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.Mirror()
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Surface.Rotate(...)

Description : Rotate the surface
Parameters : Number: number of degrees to rotate (clockwise)
Optional:
Number : red component (0-255) of the added area
Number : green component (0-255) of the added area
Number : blue component (0-255) of the added area
Number : opacity value (0-100) of the added area
Return value : Boolean: true if the surface could be properly rotated
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.Rotate(20, 0,0,0, 0) -- in degrees, transparent margins
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Note : the center of the rotation is the center of the surface

Surface.RLSkew(...)

Description : Skew the surface right or left
Parameters : Number: number of degrees to skew (positive -> to the right)
Optional:
Number : red component (0-255) of the added area
Number : green component (0-255) of the added area
Number : blue component (0-255) of the added area
Number : opacity value (0-100) of the added area
Return value : Boolean: true if the surface could be properly skewed
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.RLSkew(20, 0,0,0, 0) -- in degrees, transparent margins
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Surface.DUSkew(...)

Description : Skew the surface down or up
Parameters : Number: number of degrees to skew (positive -> down)
Optional:
Number : red component (0-255) of the added area
Number : green component (0-255) of the added area
Number : blue component (0-255) of the added area
Number : opacity value (0-100) of the added area
Return value : Boolean: true if the surface could be properly skewed
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.DUSkew(20, 0,0,0, 0) -- in degrees, transparent margins
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Surface.Perspective(...)

Description : Create perspective effect on the surface
Parameters : Number: angle in degrees between the surface and the horizontal (no perspective = 90)
Optional:
Number: X offset between the center of the surface and the center of the perspective (default:0)
Number: Y offset between the center of the surface and the center of the perspective (default:0)
Number: distance between the observer and the picture (default: 1000)
Return value : Boolean: true if the surface could be properly rotated
Number: First Line of the picture (counted from the top) to be non-empty
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.Perspective(30) -- in degrees
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)      -- draw altered surface
s.Dispose()                  -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()               -- Dialog is disposed
```

Note : For performance reasons, this function doesn't antialias result

Surface.Distort(...)

Description : Distort a surface according to a distortion map
Parameters : Surface: distortion map values
Number: X offset for applying map
Number: Y offset for applying map
Number: Refraction index (percent of the air refraction)
Number: highlight ratio (percent)
Number: Shining ratio (percent)
Return value : Boolean: true if the operation has been successfully performed
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."cat.jpg") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s2=Surface.New(path.."myriad.png") -- working surface receives the file
s2.Scale(150,150)                  -- make bigger
s2.AddMargins(10,10,10,10)         -- add margins
s2.OpacityToLuminance()            -- transfer opacity mask into pixel brightness
s2.Blur(12)                        -- blur to get smooth ramps
s2.AdjustBCS(-10,20,0)             -- enhance contrast
s.Distort(s2,20,20, 150, 30,35)
gr.DrawSurface(s2,240,200)             -- show distortion map
s2.Dispose()
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)       -- draw altered surface
s.Dispose()                   -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()                -- Dialog is disposed
```

4.6.4.2.8. Surface shadowing

Surface.DropShadow(...)

Description : Drop a shadow. The opaque parts drop a shadow on the transparent parts of the surface
Parameters : Number: X pixel offset of the shadow
Number: Y pixel offset of the shadow
Number: Red component of the shadow color (0-255)
Number: Green component of the shadow color (0-255)
Number: Blue component of the shadow color (0-255)
Number: Radius of in pixel the shadow blur
Number: Opacity of the shadow (0-100)
Return value : Boolean: true if the operation has been successfully completed
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."myriad.png") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.DropShadow(7,7, 0,0,0, 6, 50)
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)       -- draw altered surface
s.Dispose()                   -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()                -- Dialog is disposed
```

Surface.DropInnerShadow(...)

Description : Drop a shadow inside a picture. The transparent parts drop a shadow on the opaque parts of the surface
Parameters : Number: X pixel offset of the shadow
Number: Y pixel offset of the shadow
Number: Red component of the shadow color (0-255)
Number: Green component of the shadow color (0-255)
Number: Blue component of the shadow color (0-255)
Number: Radius of in pixel the shadow blur
Number: Opacity of the shadow (0-100)
Return value : Boolean: true if the operation has been successfully completed
Example :

```
-- Create dialog to display surface
dial=NewDialog("Surface",720,400)
dial.Show()
gr=dial.GraphContext

-- Load surface
path=GetDocumentationPathName().."pictures/surface/"
s=Surface.New(path.."myriad.png") -- working surface receives the file
gr.DrawSurface(s,5,5)          -- draw original surface
gr.MoveTo(10+s.Width,s.Height/2+10)
gr.TextSize(40)
gr.DrawString("=>")

-- Test routine starts here
s.DropInnerShadow(7,7, 255,255,255, 3, 80)
-- End of test routine

x,y=gr.GetPen()
gr.DrawSurface(s,x+5,5)       -- draw altered surface
s.Dispose()                   -- Surface is disposed

while Application.GetMouseButton()==0 do dial.Select() end -- wait for button
dial.Dispose()                -- Dialog is disposed
```

4.6.5. TextWindow Object Type

General description of this object type : A script can use a TextWindow object to open one (or more) independant text window and write text in it. The TextWindow objects created by a script are not closed when the script end.

You can act precisely on the text related with a TextWindow using the Text (page 578) object method related to the TextWindow.RelatedText (page 561) value.

Note: A script can also output formatted strings to the log window (page 75) using the *print* function. The log window is shared between all the scripts and is regularly cleared. For example, when a script start running, the log window is erased.
Example :

```
MyTextWindow=NewTextWindow("My Title")
MyTextWindow.Write("Hello World !")
```

4.6.5.1. TextWindow global functions

4.6.5.1.1. Creating

NewTextWindow(...)

Description : Create a new empty TextWindow
Parameters : Optional :
· String, the title of the window
· Boolean, true if the text window is visible, false if the text window is invisible. In this last case, you will use TextWindow.Show to show it later.
· Number, number of lines in the text window, 0 if there is no limit. By default the TextWindow will record the 1000 last lines.
Return value : A TextWindow object
Example :

```
--- Here is a demo of a TextWindow limited to one line
MyTextWindow=NewTextWindow("Creating",true,1)
i=0
while (i<1000) do
    MyTextWindow.Write(i)
    i=i+1
end
```

FindTextWindowByTitle(...)

Description : Search for an opened Text window with the given title caption. As Text Window are not closed when the script end, this function enables to check whether the window you need is already opened.
Parameters : String : the title.
Return value : A TextWindow object or nil if not found.
Example :

```
--- Create a new text window only if not already exist
title="My title"
MyTextWindow=FindTextWindowByTitle(title)
if(MyTextWindow==nil) then
    MyTextWindow=NewTextWindow(title)
end
```

4.6.5.2. TextWindow values

4.6.5.2.1. Display

TextWindow.GraphContext

Type : Graph (page 516)
Description : Graphical access to the window area.

TextWindow.RelatedText

Type : Text (page 578)
Description : Text related to the window.

4.6.5.2.2. Window appearance

TextWindow.WindowWidth

Type : Number
Description : Width of the window. This width don't include the margins of the window.

TextWindow.WindowHeight

Type : Number
Description : Height of the window. This height don't include the margins of the window.

TextWindow.WindowHorizontalPosition

Type : Number
Description : Horizontal position of the window on the screen. This width don't include the margins of the window.
Example :

```
MyTextWindow=NewTextWindow("Moving window")
MyTextWindow.Write("This window will move...")
i=0
while (i<80) do
   MyTextWindow.WindowHorizontalPosition=MyTextWindow.WindowHorizontalPosition+1
   MyTextWindow.WindowVerticalPosition=MyTextWindow.WindowVerticalPosition+1
   i=i+1
end
```

TextWindow.WindowVerticalPosition

Type : Number
Description : Vertical position of the score window on the screen. This width don't include the margins of the window.

TextWindow.Title

Type : String
Description : The title of the window.

TextWindow.WindowLeftMargin

Type : Number
Description : Provide the window left margin. You can't change this value. This margin is interface dependant and can change from an Operating System to another. You can use this value to move the window or resize it, taking account of the no-usable area.

TextWindow.WindowRightMargin

Type : Number
Description : Provide the window right margin. You can't change this value.This margin is interface dependant and can change from an Operating System to another. You can use this value to move the window or resize it, taking account of the no-usable area.

TextWindow.WindowTopMargin

Type : Number
Description : Provide the window top margin (the window Title height). You can't change this value.This margin is interface dependant and can change from an Operating System to another. You can use this value to move the window or resize it, taking account of the no-usable area.

TextWindow.WindowBottomMargin

Type : Number
Description : Provide the window bottom margin. You can't change this value.This margin is interface dependant and can change from an Operating System to another. You can use this value to move the window or resize it, taking account of the no-usable area.

4.6.5.3. TextWindow methods

4.6.5.3.1. Managing

TextWindow.Dispose(...)

Description : Dispose the text window and release all data related.
Parameters : None
Return value : None

```
MyTextWindow=NewTextWindow("My Title")
MyTextWindow.Write("We will wait two seconds then the window will close...")
Application.WaitDelay(2*60)
MyTextWindow.Dispose();
```

TextWindow.Show(...)

Description : Show the TextWindow object. The window become the front window.
Parameters : None
Return value : None
Example :

```
--- Here is a demo of a writing on an invisible window, then showing it
MyTextWindow=NewTextWindow("Writing on invisible window",false)
i=0
while (i<100) do
   MyTextWindow.Write(i)
   i=i+1
   end
MyTextWindow.Show()
```

TextWindow.Hide(...)

Description : Hide the TextWindow object. You can write string in it but you will not see the result.
Parameters : None
Return value : None
Example :

```
MyTextWindow=NewTextWindow("Hide demo")
MyTextWindow.Write("We will wait two seconds then the window will hide...")
Application.WaitDelay(2*60)
MyTextWindow.Hide()
Application.WaitDelay(20)
MyTextWindow.Show()
MyTextWindow.Write("Hello it is me again !")
```

TextWindow.BringToFront(...)

Description : The TextWindow object will be displayed in front of all other windows (except, of course, the palettes window)
Parameters : None
Return value : None

4.6.5.3.2. Text

TextWindow.Write(...)

Description : Write text in the text window.
Parameters : String to write
Optional : A boolean value indicating if the next string will be displayed on the next line or not.
If false, the next string will be displayed just behind the other.
If true, a Carriage return will be added after the string (by default)
Return value : None
Example :

```
MyTextWindow=NewTextWindow("Writing in window")
i=0
while (i<80) do
   MyTextWindow.Write(i)
   i=i+1
end
```

TextWindow.Erase(...)

Description : Erase all the text displayed in the TextWindow
Parameters : None
Return value : None

TextWindow.GlobalStyle(...)

Description : Change the style of all the text. (font, size,face)
Parameters : String : a font name
Optional : Number, the size.
Number, the face (normal=0, bold=1, italic=2, underline=4, outline=9, shadow=16)
Return value : None
Example :

```
MyTextWindow=NewTextWindow("Style demo")

MyTextWindow.GlobalStyle("times",14,1)
MyTextWindow.Write("The style is now times, 14, bold.")
```

4.6.5.3.3. Printing

TextWindow.Print(...)

Description : Print the text displayed on the window
Parameters : None
Return value : None
Example :

```
MyTextWindow=NewTextWindow("Printing a window")
i=0
while (i<80) do
   MyTextWindow.Write(i)
   i=i+1
end
MyTextWindow.Print()
MyTextWindow.Dispose()
```

4.6.5.3.4. Saving

TextWindow.Save(...)

Description : Save the text displayed on the window in a text file.
Parameters : None
Return value : None

4.6.6. Palette Object Type

General description of this object type : The palette are little floating windows always in front of all other windows. The user can arrange the palettes on the screen where he want. A script can hide or show the palettes, move them, change the orientation, simulate user's click on the palette...

The Palette Object is always related to the Application Object type. For example to open a palette you will write :

```
Application.Palette.Open("User 1")
```

A palette is identified by its name :

- "Edit" General editing modes
- "Play" Play/Stop music
- "Note" Note Duration selection
- "Accidental" Accidentals
- "Rest" Rest duration selection
- "BarLine" Bar Lines selection
- "Miscellaneous" Slur, Chord diagrams, graphics...
- "Help" Help line window (*)
- "Simplified Mixer" Simplified Mixer
- "Menu" Shortcuts on menus
- "MidiMeters" Midi channels output status
- "Virtual KeyBoard" Virtual keyboard
- "Appearance" Symbols appearance
- "Ornaments 1" First group of ornements
- "Ornaments 2" Second group of ornements
- "Ornaments 3" Third group of ornements
- "User 1" User 's palette (*)
- "User 2" User 's palette (*)
- "User 3" User 's palette (*)
- "User 4" User 's palette (*)
- "User 5" User 's palette (*)
- "User 6" User 's palette (*)
- "User 7" User 's palette (*)
- "User 8" User 's palette (*)
- "Dynamic" Dymamics
- "Clef" Clef, time signature, key signature
- "Gregorian" Gregorian neumes and symbols
- "Ottava" Ottava, tempo...
- "Complete Mixer" Complete mixer palette
- "Record" Record from Midi or Microphone
- "Master" Master palette
- "Virtual Singer" Virtual Singer palette
- "Jump&Target" Jumps and targets symbols
- "Page Mode" mode page preview (*)
- "Time Slider" Time slider (*)
- "Style" Styles (*)

(*) The size of this palette window can be changed by a script.

A script can simulate click on a palette (opened or not) using the Palette.PseudoClick(...) (page 574) method. The include "MSPalette" provide the different values you can use. Because these values can change in future, it is not recommand to access to palettes icons without using theses constants.

```
            --- © Myriad 2003-2013
--- Updated December 2013
-------------------------------------------------
-- Definitions for "Edit" palette icons
-------------------------------------------------
PLT_ADD_MODE=2
PLT_SELECT_MODE=3
PLT_DELETE_MODE=4
PLT_INJECT_MODE=5
PLT_TRY_MODE=6
PLT_SCALE_MODE=7
PLT_DOCUMENTATION_MODE=8
-------------------------------------------------
-- Definitions for "Note" palette icons
-------------------------------------------------
PLT_NOTE_DURATION_LONG=2
PLT_NOTE_DURATION_BREVE=3
PLT_NOTE_DURATION_WHOLE=4
PLT_NOTE_DURATION_HALF=5
PLT_NOTE_DURATION_QUARTER=6
PLT_NOTE_DURATION_HEIGHT=7
PLT_NOTE_DURATION_16TH=8
PLT_NOTE_DURATION_32TH=9
PLT_NOTE_DURATION_64TH=10
PLT_NOTE_DURATION_TUPLET=11
```

```
PLT_NOTE_DURATION_DOT=12
PLT_NOTE_DURATION_DOUBLE_DOT=13
PLT_GRACE_BEFORE=14
PLT_GRACE_AFTER=15
PLT_REPEAT_BAR_SYMBOL=16
-------------------------------------------------
-- Definitions for "Accidental" palette icons
-------------------------------------------------
PLT_SHARP=2
PLT_FLAT=3
PLT_NATURAL=4
PLT_DOUBLE_SHARP=5
PLT_DOUBLE_FLAT=6
-------------------------------------------------
-- Definitions for "Rest" palette icons
-------------------------------------------------
PLT_REST_DURATION_LONG=2
PLT_REST_DURATION_BREVE=3
PLT_REST_DURATION_WHOLE=4
PLT_REST_DURATION_HALF=5
PLT_REST_DURATION_QUARTER=6
PLT_REST_DURATION_HEIGHT=7
PLT_REST_DURATION_16TH=8
PLT_REST_DURATION_32TH=9
PLT_REST_DURATION_TUPLET=10
PLT_REST_DURATION_DOT=11
PLT_REST_DURATION_DOUBLE_DOT=12
-------------------------------------------------
-- Definitions for "BarLine" palette icons
-------------------------------------------------
PLT_BAR_START_MUSIC=2
PLT_BAR_END_MUSIC=3
PLT_BAR_START_LOOP=4
PLT_BAR_END_LOOP=5
PLT_BAR_NONE=6
PLT_BAR_DOUBLE=7
PLT_BAR_BOLD=8
PLT_BAR_DASHED=9
PLT_BAR_STANDARD=10
-------------------------------------------------
-- Definitions for "Miscellaneous" palette icons
-------------------------------------------------
PLT_TEXT_OBJECT=2
PLT_DIAGRAM_OBJECT=3
PLT_GRAPHIC_OBJECT=4
PLT_LINE_OBJECT=5
PLT_SLUR_OBJECT=6
PLT_BEAM_OBJECT=7
PLT_FX_OBJECT=8
PLT_FREE_OBJECT=9
-------------------------------------------------
-- Definitions for "Appearance" palette icons
-------------------------------------------------
PLT_NORMAL_HEAD_SHAPE=2
PLT_EMPTY_HEAD_SHAPE=3
PLT_TRIANGLE_HEAD_SHAPE=4
PLT_DIAMOND_HEAD_SHAPE=5
PLT_OBLIQUE_CROSS_HEAD_SHAPE=6
PLT_OBLIQUE_CROSS_IN_CIRCLE_HEAD_SHAPE=7
PLT_HALF_OBLIQUE_CROSS_IN_CIRCLE_HEAD_SHAPE=8
PLT_CROSS_HEAD_SHAPE=9
PLT_SQUARE_HEAD_SHAPE=10
PLT_SLASH_HEAD_SHAPE=11
PLT_HALF_SQUARE_HEAD_SHAPE=12
PLT_HALF_CIRCLE_HEAD_SHAPE=13
PLT_DROP_HEAD_SHAPE=14
PLT_BOLD_OBLIQUE_CROSS_HEAD_SHAPE=15
PLT_ECLIPSE_HEAD_SHAPE=16
PLT_EMPTY_DIAMOND_HEAD_SHAPE=17
PLT_BIG_OBLIQUE_CROSS_HEAD_SHAPE=18
PLT_NO_HEAD_SHAPE=19
PLT_STEM_DIRECTION=20
PLT_HEAD_DIRECTION=21
-- Color can't be selected because need a long click to open the selector
-- Symbol scale can't be selected because need a long click to open the selector
-------------------------------------------------
-- Definitions for "Ornaments 1" palette icons
-------------------------------------------------
PLT_ORNAMENT_1_POSITION=2
PLT_ORNAMENT_1_ONSTEM=3
PLT_RISE_VOLUME=4
PLT_RELEASE_VOLUME=5
PLT_TRILL=6
```

```
PLT_FLAT_TRILL=7
PLT_SHARP_TRILL=8
PLT_NATURAL_TRILL=9
PLT_HIGH_MORDENT=10
PLT_LOW_MORDENT=11
PLT_DOUBLE_HIGH_MORDENT=12
PLT_DOUBLE_LOW_MORDENT=13
PLT_GRUPPETTO_UP=14
PLT_GRUPPETTO_DOWN=15
PLT_EIGHTH_GROUP=16
PLT_SIXTEENTH_GROUP=17
PLT_32ND_GROUP=18
PLT_64TH_GROUP=19
PLT_VIBRATO=20
PLT_TREMOLO=21
PLT_FERMATA=22
PLT_INVERTED_FERMATA=23
PLT_TENUTO=24
PLT_STACCATO=25
PLT_MARCATO=26
PLT_PAUSE=27
PLT_GRAND_PAUSE=28
PLT_HEEL_PEDAL=29
PLT_PLUS=30
PLT_ARPEGGIO=31
-------------------------------------------------
-- Definitions for "Ornaments 2" palette icons
-------------------------------------------------
PLT_ORNAMENT_2_POSITION=2
PLT_ORNAMENT_2_ONSTEM=3
PLT_BEND_UP=4
PLT_BEND_DOWN=5
PLT_HAMMER=6
PLT_PULLING_OFF=7
PLT_PULL=8
PLT_RELEASE=9
PLT_PULL_RELEASE=10
PLT_MUTED=11
PLT_FLOYD=12
PLT_TILDE=13
PLT_DADI=14
PLT_CHOKE=15
PLT_PALM_MUTE=16
PLT_MEDIATOR_UP=17
PLT_MEDIATOR_DOWN=18
PLT_TAPPING=19
PLT_SLAP=20
PLT_POP=21
PLT_SLIDE=22
PLT_HARMONICS=23
PLT_ARTIFICIAL_HARMONICS=24
PLT_COMMA_1_SHARP=25
PLT_COMMA_1_FLAT=26
PLT_COMMA_5_SHARP=27
PLT_COMMA_5_FLAT=28
PLT_COMMA_8_SHARP=29
PLT_COMMA_8_FLAT=30
-------------------------------------------------
-- Definitions for "Ornaments 3" palette icons
-------------------------------------------------
PLT_ORNAMENT_3_POSITION=2
PLT_ORNAMENT_3_ONSTEM=3
PLT_TEXT_MARK=4
PLT_LINE_TEXT_MARK=5
PLT_ARC_DOWN=6
PLT_ARC_UP=7
PLT_FINGER_0=8
PLT_FINGER_1=9
PLT_FINGER_2=10
PLT_FINGER_3=11
PLT_FINGER_4=12
PLT_FINGER_5=13
PLT_FINGER_THUMB=14
PLT_FINGER_INDEX=15
PLT_FINGER_MEDIUS=16
PLT_FINGER_RING=17
PLT_STRING_1=18
PLT_STRING_2=19
PLT_STRING_3=20
PLT_STRING_4=21
PLT_STRING_5=22
PLT_STRING_6=23
PLT_DOWN_BOW=24
```

```
PLT_STACCATTISSIMO_MARTELATO=25
PLT_SFORZANDO=26
PLT_INVERTED_SFORZANDO=27
PLT_MEZZO_STACCATO_UP=28
PLT_MEZZO_STACCATO_DOWN=29
PLT_STACCATTISSMO_UP=30
PLT_STACCATTISSMO_DOWN=31
PLT_MARCATO_TENUTO_UP=32
PLT_MARCATO_TENUTO_DOWN=33
PLT_MARCATO_STACCATO_UP=34
PLT_MARCATO_STACCATO_DOWN=35
PLT_MARTELATO_TENUTO_UP=36
PLT_MARTELATO_TENUTO_DOWN=37
PLT_MARTELATO_STACCATO_UP=38
PLT_MARTELATO_STACCATO_DOWN=39
-------------------------------------------------
-- Definitions for "User" palette icons
-------------------------------------------------
-- There is no way to access to icons of this palette
-------------------------------------------------
-- Definitions for "Dynamic" palette icons
-------------------------------------------------
PLT_CRESCENDO_DYNAMIC=2
PLT_DECRESCENDO_DYNAMIC=3
PLT_PPP_DYNAMIC=4
PLT_PP_DYNAMIC=5
PLT_P_DYNAMIC=6
PLT_MP_DYNAMIC=7
PLT_MF_DYNAMIC=8
PLT_F_DYNAMIC=9
PLT_FF_DYNAMIC=10
PLT_FFF_DYNAMIC=11
PLT_SF_DYNAMIC=12
PLT_FZ_DYNAMIC=13
PLT_FP_DYNAMIC=14
PLT_FPP_DYNAMIC=15
PLT_SFZ_DYNAMIC=16
PLT_SFFZ_DYNAMIC=17
PLT_SFPP_DYNAMIC=18
-------------------------------------------------
-- Definitions for "Clef" palette icons
-------------------------------------------------
PLT_CLEF=2
PLT_KEY_CHANGE=3
PLT_TIME_SIGNATURE=4
-------------------------------------------------
-- Definitions for "Gregorian" palette icons
-------------------------------------------------
PLT_PUNCTUM=2
PLT_VIRGA=3
PLT_LIQUESCENS=4
PLT_PUNCTUM_MORA=5
PLT_BREATH=6
PLT_GREGORIAN_FLAT=7 -- This also affect the accidental palette
PLT_GREGORIAN_NATURAL=8 -- This also affect the accidental palette
PLT_SPLIT_NEUMES=9
PLT_LINK_NEUMES=10
-------------------------------------------------
-- Definitions for "Ottava" palette icons
-------------------------------------------------
PLT_8VA_OTTAVA=2
PLT_8VB_OTTAVA=3
PLT_15MA_OTTAVA=4
PLT_15MB_OTTAVA=5
PLT_TEMPO=6
PLT_PED_BEGIN=7
PLT_PED_END=8
-------------------------------------------------
-- Definitions for "Complete Mixer" palette icons
-------------------------------------------------
-- There is no way to access to icons of this palette
 -------------------------------------------------
 -- Definitions for "Record" palette icons
 -------------------------------------------------
 PLT_SIMPLIFIED_INPUT=2
 PLT_DIGITAL_INPUT=3
 PLT_MIDI_INPUT=4
 PLT_KEYBOARD_INPUT=5
 PLT_START_END_INPUT=6
-------------------------------------------------
-- Definitions for "Master" palette icons
-------------------------------------------------
-- There is no way to access to icons of this palette because there are only shortcuts on other palettes
```

```
-----------------------------------------------
-- Definitions for "Virtual Singer" palette icons
-----------------------------------------------
-- There is no way to access to icons of this palette
-----------------------------------------------
-----------------------------------------------
-- Definitions for "Jump&Target" palette icons
-----------------------------------------------
PLT_BEGIN_PART=2
PLT_END_PART=3
PLT_JUMP_TO_CODA=4
PLT_FINE=5
PLT_JUMP_TO_SEGNO=6
PLT_DA_CAPO=7
PLT_DA_SEGNO=8
PLT_DC_ALFINE=9
PLT_DS_ALFINE=10
PLT_DC_ALCODA=11
PLT_DS_ALCODA=12
PLT_DA_CODA=13
-----------------------------------------------
-- Definitions for "Page Mode" palette icons
-----------------------------------------------
-- There is no way to access to icons of this palette
-----------------------------------------------
-- Definitions for "Time Slider" palette icons
-----------------------------------------------
-- There is no way to access to icons of this palette
-----------------------------------------------
-- Definitions for "Style" palette icons
-----------------------------------------------
PLT_ADD_STYLE=4
PLT_DEL_STYLE=5
PLT_APPLY_STYLE=6
-----------------------------------------------
-- Definitions for "Play" palette icons
-----------------------------------------------
PLT_PLAY_STOP=2
PLT_PAUSE=3
-- Fast forward and backward are not usable because need a long click
PLT_PLAY_VIEW=5
PLT_FAST_PLAY_SELECTION=6
PLT_PLAY_SELECTION=6
PLT_PLAY_LOOP=7
PLT_METRONOME=8
PLT_ADJUST=9
PLT_KARAOKE=10
PLT_JUKEBOX=11
PLT_RYTHMS=12
PLT_ACCOMPANIEMENTS=13
```

In general, a click on an icon toggle the current state.
The method Palette.Reset(...) (page 575) set the default value of the palette.
Example :

```
-- If you relate this script to a keyboard shortcut (configuration menu)
-- You will toogle the tool mode on one key press.
Include "MSPalette"
local palette
palette=Application.Palette.GetPaletteByName("Ornaments 3")
palette.PseudoClick(PLT_MARCATO_TENUTO_UP)
```

4.6.6.1. Palette values

4.6.6.1.1. Attributes

Palette.Name

Type : String
Description : Palette name.
Example :

```
local thePalette
thePalette=Application.Palette.FirstPalette()
while thePalette~=nil do
   print(thePalette.Name)
   thePalette=thePalette.Next
end
```

Palette.IsOpen

Type : Boolean : true if the palette window is open.
Description : Determinate if the palette window is open.
Example :

```
thePalette=Application.Palette.GetPaletteByName("Play")
if(thePalette.IsOpen==true) then
   Message("'Play' palette is open")
else
   Message("'Play' palette is not open")
end
```

Palette.IsVertical

Type : Boolean : true if the palette window is vertical.
Description : Determinate if the palette window is vertical.
Example :

```
thePalette=Application.Palette.GetPaletteByName("Play")
if(thePalette.IsVertical==true) then
   Message("The 'Play' palette is vertical")
else
   thePalette.InvertOrientation()
end
```

4.6.6.1.2. Browsing

Palette.Next

Type : Palette object type
Description : Provide the next palette in the palette list.
Example :

```
local thePalette
thePalette=Application.Palette.FirstPalette()
while thePalette~=nil do
   print(thePalette.Name)
   thePalette=thePalette.Next
end
```

4.6.6.2. Palette methods

4.6.6.2.1. General

Palette.HideAll(...)

Description : Hide open palettes. The palettes are not closed, but the window palettes are just hidden. Must be balanced with a ShowPalette.
Parameters : None
Return value : None
Example :

```
Application.Palette.HideAll()
```

Palette.ShowAll(...)

Description : Show open palettes.
Parameters : None
Return value : None
Example :

```
Application.Palette.ShowAll()
```

4.6.6.2.2. Appearance

Palette.InvertOrientation(...)

Description : Invert the palette window orientation. If the palette window is vertical it become horizontal and vice versa. If the palette is not open, does nothing.
Parameters : Palette object
Return value : None
Example :

```
thePalette=Application.Palette.Open("Play")
thePalette.InvertOrientation()
```

4.6.6.2.3. Browsing

Palette.FirstPalette(...)

Description : Provide the first palette available. The palette can be open or not, hidden or not.
Parameters : None
Return value : Palette object
Example :

```
palette=Application.Palette.FirstPalette()
Message("The first application palette is : "..palette.Name)
```

4.6.6.2.4. Managing

Palette.Open(...)

Description : Open the palette window. The palette is displayed at his previous location with the previous orientation.
You can use this method even the palette is already open.
Parameters : String : Palette name.
Return value : Palette object type
Example :

```
  Application.Palette.Open("Menu")
```

Palette.GetPaletteByName(...)

Description : Provide the palette object with the given name. The palette object can be in opened or closed state.
Parameters : String : Palette name.
Return value : Palette
Example :

```
local thePalette
thePalette=Application.Palette.GetPaletteByName("Menu")
thePalette.Open()
```

Palette.Close(...)

Description : Close the palette window.
Parameters : String : Palette name.
Return value : None
Closing a palette using a palette object
Example :
Closing a palette using his name.

```
Application.Palette.Close("Menu")
```

Closing a palette using a palette object

```
local thePalette
thePalette=Application.Palette.GetPaletteByName("Menu")
thePalette.Close()
```

4.6.6.2.5. Window Position

Palette.SetPosition(...)

Description : Set the palette position on the screen.
Parameters : · Palette object
· Number : vertical position of the palette window on the screen.
· Number : horizontal position of the palette window on the screen.
Return value : None
Example :

```
thePalette=Application.Palette.Open("Play")
thePalette.SetPosition(0,Graph.MenuBarHeight())
```

Palette.GetPosition(...)

Description : Get the current palette position on the screen. If the palette is not open, return nil.
Parameters : Palette object
Return value : · Number : vertical position of the palette window on screen.
· Number : horizontal position of the palette window on screen.
Example :

```
thePalette=Application.Palette.Open("Play")
x,y=thePalette.GetPosition()
Message("The 'Play' palette is located at : ("..x.." , "..y..")")
```

Palette.Magnetize(...)

Description : Move the palette automatically to avoid overlapping
Parameters : None
Return value : None

4.6.6.2.6. Window size

Palette.SetSize(...)

Description : Set the palette size. Only some palettes can be resized : "User 1" to "User 8", "Help", "Page Mode', "Time Slider","Style", "Complete Mixer".
Parameters : · Number : vertical size of the palette window on the screen, in pixel.
· Number : horizontal size of the palette window on the screen, in pixel.
Return value : None
Example :

```
thePalette=Application.Palette.Open("User 1")
thePalette.SetSize(200,200)
```

Palette.GetSize(...)

Description : Get the current palette window size. If the palette is not open, return nil.
Parameters : Palette object
Return value : · Number : vertical size of the palette window on screen, in pixel.
· Number : horizontal size of the palette window on screen, in pixel.
Example :

```
thePalette=Application.Palette.Open("User 1")
x,y=thePalette.GetSize()
Message("The 'User' palette size is : ("..x.." , "..y..")")
```

Palette.SetScale(...)

Description : Change the scale of the palette (zoom)
Parameters : scale to set in percent of the default scale
Return value : None

Palette.GetScale(...)

Description : Get the current scale of the palette (zoom)
Parameters : None
Return value : Current scale in percent of the default value

4.6.6.2.7. Actions

Palette.PseudoClick(...)

Description : Simulate a click on an icon included in a the palette. The include file "MSPalette" provide the different icon values you can use.
Parameters : Number : Icon number
Return value : None
Example :

```
Include "MSPalette"
local palette
palette=Application.Palette.GetPaletteByName("Play")
palette.PseudoClick(PLT_PLAY_STOP)
```

Palette.SendHelpMessage(...)

Description : Send the message on the user's help window (if opened)
Parameters : String : the message to display (styled)
Boolean (optional) : If true, the height of the window is adjusted. By default,false.
Return value : None
Example :

```
Application.Palette.SendHelpMessage(SetStringStyle("Hello, this is a message on the help line.",0,0,"Times","bi",
14,"FF0000"))
```

Palette.Reset(...)

Description : Set the current state of the palette to the default value, drawing the palette if necessary.
- The "Edit" palette will have no edit mode selected.
- The "Play" palette is not affected.
- The "Note" palette will have the quarter duration selected, no grace, no dot, no tuplet.
- The "Rest" palette will be inactive (no icon selected).
- The "Accidental" palette will be inactive (no icon selected).
- The "BarLine" palette will be inactive (no icon selected).
- The "Miscellaneous" palette will be inactive (no icon selected).
- The "Help" palette is cleared (no message)
- The "Simplified Mixer" palette is not affected.
- The "Complete Mixer" palette is not affected.
- The "Menu" palette is not affected.
- The "MidiMeters" palette is not affected.
- The "Virtual KeyBoard" palette is not affected.
- The "Appearance" palette will be inactive (no icon selected).
- The "Guitar Ornaments" palette will be inactive (no icon selected).
- The "Ornaments 1" palette will be inactive (no icon selected).
- The "Ornaments 2" palette will be inactive (no icon selected).
- The "Ornaments 3" palette will be inactive (no icon selected).
- The "User 1" to "User 8" palettes are not affected.
- The "Dynamic" palette will be inactive (no icon selected).
- The "Clef" palette will be inactive (no icon selected).
- The "Ottava" palette will be inactive (no icon selected).
- The "Record" palette will be inactive (no icon selected).
- The "Master" palette is not affected.
- The "Virtual Singer" palette is not affected.
- The "Jump&Target" palette will be inactive (no icon selected).
- The "Page Mode" palette is not affected.
- The "Time Slider" palette is not affected.
- The "Style" palette is not affected.

Parameters : None
Return value : None
Example :

```
local thePalette
thePalette=Application.Palette.FirstPalette()
while thePalette~=nil do
   thePalette.Reset()
   thePalette=thePalette.Next
end
```

Palette.KeyPressed(...)

Description : Emulate a key pressed on the Virtual Keyboard palette
Parameters : Number : pitch of the key
(optional) Number: fingering (number from 1=thumb to 5=little finger, 0=no fingering)
(optional) String: Color of key highlight ("RRGGBB" components in hexadecimal)
Return value : None

Palette.KeyReleased(...)

Description : Emulate a key released on the Virtual Keyboard palette
Parameters : Number : pitch of the key
Return value : None

Palette.ResetKeyboard(...)

Description : Emulate all keys released on the Virtual Keyboard palette
Parameters : None
Return value : None

Palette.ResetAll(...)

Description : Set the current state of all palettes to their default value, drawing the palettes if necessary.

4.6.7. ProgressBar Object Type

General description of this object type : The ProgressBar is a predefined object you can use to inform the user a special computing is running.
For example, if your script need time to compute, thanks to the ProgressBar the user will understand that the computer is not freezed but is working in background.
Only one ProgressBar can be open at the same time. Its position is always in the middle of the screen.
The user can break your script using the "Stop!" button.
If the script is terminated because an error occurs, the open progress bar is automatically closed.

Warning: Some object methods use automatically the scroll bar to show the work in progress. For example the Score.Preserve() method.
So, this method can clear the scroll bar maximum and current value, always open the scroll bar after calling Score.Preserve().

4.6.7.1. ProgressBar methods

4.6.7.1.1. Creating

ProgressBar.Open(...)

Description : Open the progress bar. If a progress bar is already open, do nothing except changing the text displayed
and the maximum value.
Parameters :
A string value defining the text you want to display in the progress bar. If not specified the default text is "Computing..."
A number value defining the maximum of the scroll bar. If not specified the default value is 1000.
Return value : None.
Example :

```
local i=0
ProgressBar.Open("Displaying value from 0 to 1000",1000)
while i<1000 and ProgressBar.Status(i)==false do
   print (i)
   i=i+1
end
ProgressBar.Close()
```

4.6.7.1.2. Current status

ProgressBar.Write(...)

Description : Change the text displayed in the progress bar.
If your script use different computing status, you can inform the user of the current status.
Parameters : a string
Return value : Nothing
Example :

```
local i=0
local maximum=1000
ProgressBar.Open("Displaying value from 0 to "..maximum,maximum)
while i<maximum and ProgressBar.Status(i)==false do
   ProgressBar.Write("The current status is "..i.." on "..maximum)
   print (i)
   i=i+1
end
ProgressBar.Close()
```

ProgressBar.Status(...)

Description : You ought to invoque regularly this function in your script to change the current value displayed in the progress bar. The value varies from 0 to the maximum value you defined when opening the progress bar.
Parameters : a number
Return value : true if the user click on the Stop button. You ought to terminate your script.

ProgressBar.Maximum(...)

Description : Define the maximum of the progress bar.
Parameters : a number
Return value : None

4.6.7.1.3. Disposing

ProgressBar.Close(...)

Description : Close the progress bar. You ought to call this method one time for each OpenProgressBar call.
Parameters : Nothing
Return value : Nothing

4.6.8. Text Object Type

General description of this object type : The Text object type provides access to values and methods for managing basic text formatting and editing. You can use the Text object type in different places :

· When you create a TextWindow (page 560) object type, a Text object is automatically related to it. You can use the TextWindow.RelatedText (page 561) value to access this Text object.
· When you define a Text item type in a dialog. You can use the DialogItem.GetExtraValue(1) (page 513) method to access to the text object related to the item.

This chapter describes how to use Text object type to perform a range of editing and formatting capabilities including

inserting new text
selecting and highlighting ranges of text
deleting selected text and possibly inserting it elsewhere, or copying text without deleting it
replacing selected text
changing the characteristics of text, including font family, style, and size

Related topics :
· text line (lyrics) related to a staff are managed by Staff.GetLyricsCellIndexAtTime(...) (page 281), Staff.GetLyricsCellTime(...) (page 282), Staff.GetLyricsCellContent(...) (page 282), etc methods.

Example :

```
title="Text object demo"
MyTextWindow=FindTextWindowByTitle(title)
MyWindow=FindTextWindowByTitle(title)
if(MyWindow==nil) then
   MyWindow=NewTextWindow(title)
end
theText=MyWindow.RelatedText
theText.Insert("We insert a line\r")
```

4.6.8.1. Text values

4.6.8.1.1. Data

Text.NumberOfCharacters (R)

Type : Number
Description : Number of characters recorded in the text.

Text.NumberOfLines (R)

Type : Number
Description : Number of lines in the text.

Text.Data (R)

Type : String
Description : Provide all the characters recorded in the text.

Text.SelectedData (R)

Type : String
Description : Provide all the selected characters.

4.6.8.1.2. Selection

Text.SelectionStart (R)

Type : Number
Description : Begin of the selection in the text. Use the Text.SetSelect(...) (page 581) method to change it. If the beginning is equal to the end, the cursor blink.

Text.SelectionEnd

Type : Number
Description : End of the selection in the text. Use the Text.SetSelect(...) (page 581) method to change it. If the beginning is equal to the end, the cursor blink.

4.6.8.1.3. Style

Text.FontName

Type : String
Description : Provide the current font name used to display the text.

Text.FontSize

Type : Number
Description : Provide the current font size used to display the text.

Text.FontFace

Type : Number
Description : Provide the current font face used to display the text.

4.6.8.2. Text methods

4.6.8.2.1. Browsing

Text.GetLinePosition(...)

Description : Provide position of the first character of a line number.
Parameters : Number : The line number from 1 to Text.NumberOfLines (page 578)
If the line number is lower than 1 return the position of the beginning of the text (0).
If the line number is greater than the Text.NumberOfLines (page 578) return the last position in the text Text.NumberOfCharacters (page 578)
Return value : Number : The position of first character of a line.

Text.GetPositionLine(...)

Description : Provide line of a position in character in the text.
Parameters : Number : The position from 0 to Text.NumberOfCharacters (page 578)
If the position is lower than 0 return the first line of the text (1).
If the position is greater than the Text.NumberOfCharacters (page 578) return the number of lines : Text.NumberOfLines (page 578).
Return value : Number : The line from 1 to Text.NumberOfLines (page 578)

4.6.8.2.2. Deleting

Text.DeleteSelection(...)

Description : Delete the selection range .
Parameters : None
Return value : Nothing

4.6.8.2.3. Inserting character

Text.Insert(...)

Description : Insert the specified characters immediatly before the selection range or insertion point. After this the insertion point is positioned immediatly to the right of the inserted text.
Parameters : String : Text to insert.
Return value : Nothing

Text.Key(...)

Description : Insert one character. If characters are selected, there are deleted first. Special characters are interpreted :

Insertion point movement :
30 = line up,
31 = line down
28 = one character left
29 = one character right
4 = end of line
11 = begin of line

deleting:
9 = backspace, delete the character at the left of the insertion point
127 = delete, delete the character at the right of the insertion point

Moving view area
11 = page up
12 = page down

Miscellaneous
9 = tab
10 = new line
5 = switch in insert mode

Parameters : Number : the character.
Return value : Nothing

4.6.8.2.4. Scrap book

Text.Copy(...)

Description : Copies the selection range in the scrap
Parameters : None
Return value : Nothing
Example :

```
-- Creating a new text window or selecting the already open one
title="Text object demo"
MyTextWindow=FindTextWindowByTitle(title)
MyWindow=FindTextWindowByTitle(title)
if(MyWindow==nil) then
    MyWindow=NewTextWindow(title)
end
-- We start on an empty window
 MyWindow.Erase()
--- Inserting data
theText=MyWindow.RelatedText
theText.Insert("We insert a line\r")
theText.Insert("Then a second line\r")
--- We select the first line
theText.SetSelect(theText.GetLinePosition(1),theText.GetLinePosition(2))
--- We copy it in the scrap
theText.Copy()
--- We select the second line
theText.SetSelect(theText.GetLinePosition(2),theText.GetLinePosition(3))
--- We paste the scrap overt it
theText.Paste()
```

Text.Cut(...)

Description : Copies the selection range in the scrap and remove it.
Parameters : None
Return value : Nothing

Text.Paste(...)

Description : Copies the content of the scrap at the current insertion point. If a selection range exist, it is first deleted.
After this the insertion point is positioned immediatly to the right of the inserted text.
Parameters : None
Return value : Nothing

4.6.8.2.5. Style

Text.ChangeStyle(...)

Description : Change the style (font,size,face) of the entire text.
Parameters : String : a font name
Optional :
·Number, the size.
·Number, the face (normal=0, bold=1, italic=2, underline=4, outline=9, shadow=16)
Return value : Nothing
Example :

```
title="Text object demo"
MyTextWindow=FindTextWindowByTitle(title)
MyWindow=FindTextWindowByTitle(title)
if(MyWindow==nil) then
   MyWindow=NewTextWindow(title)
end
theText=MyWindow.RelatedText
theText.Insert("We will change the font.\r")
theText.ChangeStyle("Times",14,0)
```

4.6.8.2.6. Selecting

Text.SetSelect(...)

Description : Set the selection range. If start is equal to end, the selection range will become and insertion point, and the cursor will blink. If the selection range is outside the view area, the text will scroll to show the selection.
Parameters : · Number : start of the selection, in character index from 0 to Text.NumberOfCharacters (page 578)
· Number : end of the selection, in character index from 0 to Text.NumberOfCharacters (page 578)
Return value : Nothing

4.6.9. Jukebox Object Type

General description of this object type : The jukebox window show a list of files which can be played or exported to create automaticaly web pages.
The files can be MUS, MYR, MID, KAR, etc. files.
There is only one jukebox window opened at the same time.
A script can open a jukebox, load a list, add or remove files from the list, save the list, etc.
There is one file by entry (an entry = a line)
Informations about the entry are displayed in columns.
Column identifier are described in MSDefine :

JUKEBOX_COLUMN_FILE_NAME=1
JUKEBOX_COLUMN_FOLDER_NAME=2
JUKEBOX_COLUMN_CATEGORIES=3
JUKEBOX_COLUMN_DATE=4
JUKEBOX_COLUMN_LYRICS=5
JUKEBOX_COLUMN_DURATION=6
JUKEBOX_COLUMN_WEB_PAGE=7
JUKEBOX_COLUMN_EMAIL=8
JUKEBOX_COLUMN_FILE_SIZE=9
JUKEBOX_COLUMN_TITLE=10
JUKEBOX_COLUMN_COMPOSER=11
JUKEBOX_COLUMN_COMMENT=12
JUKEBOX_COLUMN_SECONDS_WORKING_ON=13
JUKEBOX_COLUMN_PROTECT_MODE=14
JUKEBOX_COLUMN_ICON=15
JUKEBOX_COLUMN_SOUND_BASE=16
JUKEBOX_COLUMN_ANNOTATION=17
JUKEBOX_COLUMN_PLAYED_BARS=18
JUKEBOX_COLUMN_FILE_KIND=19
JUKEBOX_COLUMN_FILE_VERSION=20

Entries can be sorted ascending or descending along any column.
There is always a selected entry in the list.

4.6.9.1. Jukebox values

4.6.9.1.1. Attributes

Jukebox.IsChanged

Type : Boolean
Description : Indicate if the current jukebox list had been modified. Return false if the jukebox is not open or if the list was not changed.
Example :

```
if Jukebox.IsChanged==true then
   Message("The jukebox list is modified.")
else
   if Jukebox.IsOpen==true then
      Message("The jukebox list is not modified.")
   else
      Message("The jukebox window is not open.")

   end
end
```

Jukebox.IsOpen

Type : Boolean
Description : Indicate if the jukebox window is open or not. You can use the methods Jukebox.Open(...) (page 58 and Jukebox.Close(...) (page 584) to open or close the Jukebox window.

4.6.9.1.2. Playing modes

Jukebox.IsPlayKaraoke

Type : Boolean
Description : Indicate if the karaoke window will be opened while jukebox is playing.

Jukebox.IsLoopPlaying

Type : Boolean
Description : Indicate if the playing will restart to the begin when all the files will be played.

Jukebox.IsRandomPlaying

Type : Boolean
Description : Indicate if the files will be selected randomly when playing.

Jukebox.IsPlayOnlyFirstSeconds

Type : Boolean
Description : Indicate if only the 15 first seconds of the files will be played.

4.6.9.1.3. Browsing

Jukebox.NumberOfEntries (R)

Type : Number
Description : Provide the number of entries in the jukebox list.
Example :

```
Jukebox.Open()
Message("There is "..Jukebox.NumberOfEntries.." entries in the jukebox.")
```

Jukebox.CurrentEntry (R)

Type : Number
Description : Provide the current selected entry in the list from 1 to NumberOfEntries. The value is 0 if the list is empty or the jukebox window not opened.

4.6.9.1.4. General informations

Jukebox.TotalDuration (R)

Type : Number
Description : Provide the total duration, in seconds : sum of all duration entries in the list.

Jukebox.TotalSize (R)

Type : Number
Description : Provide the total size, in bytes : sum of all files size.

4.6.9.1.5. Jukebox file

Jukebox.FileName (R)

Type : String
Description : Jukebox list file name (.box file).

Jukebox.PathName (R)

Type : String
Description : Path on the folder where the jukebox list is (.box file).

4.6.9.1.6. Window appearance

Jukebox.AreaLeft

Type : Number
Description : Dialog box left edge position, in pixel from the screen left bound

Jukebox.AreaTop

Type : Number
Description : Dialog box top edge position, in pixel from the screen top bound

Jukebox.AreaWidth

Type : Number
Description : Dialog box width, in pixel
Example :

```
--- Setting the Jukebox window to minimal size
Jukebox.Open()
Jukebox.AreaWidth=198
Jukebox.AreaHeight=48
```

Jukebox.AreaHeight

Type : Number
Description : Dialog box height, in pixel

Jukebox.IsFrontWindow

Type : Boolean
Description : True if the dialog is the frontmost window, false otherwise.
Changing this value whill bring the window to the front or to the back.

4.6.9.2. Jukebox methods

4.6.9.2.1. Managing Jukebox

Jukebox.Open(...)

Description : Open the jukebox window. The jukebox windows is displayed at his previous location.
You can use this method even the jukebox is already open to load another jukebox list.
Parameters : optional : String : Full name of the jukebox file to load.
Return value : True if the jukebox was correctly inited. Can return false, for example if you provide a file name which can't be load.
Example :

```
Jukebox.Open()
```

Jukebox.New(...)

Description : Create a new empty jukebox list.
Parameters : None
Return value : True if the jukebox was correctly inited. False, for example, if a list already exist, was modified and the user canceled the operation.
Example :

```
Jukebox.Open()
Jukebox.New()
```

Jukebox.Close(...)

Description : Close the jukebox window. if the current list is not saved, an alert is displayed asking the user to confirm and/or save the current jukebox list.
You can avoid this confirmation using Jukebox.IsChanged (page 582)=false.
Parameters : None
Return value : True if the jukebox was correctly closed. False if not, for example, if it was already closed or if the user had canceled the operation.
Example :

```
Jukebox.Close()
```

Jukebox.Save(...)

Description : Save the jukebox list. If the list don't have name, a name is asked.
Parameters : None
Return value : True if the jukebox was correctly saved.
Example :

```
if Jukebox.IsOpen then
   Jukebox.Save()
end
```

Jukebox.BuildList(...)

Description : Clear the list and collect all files at the same level of the jukebox file, including sub folders.
Parameters : None
Return value : False, if the jukebox window is not open.

Jukebox.CheckList(...)

Description : Check all files in the list. The total duration and total file size is computed. Files not found are displayed in gray in the list.
Parameters : None
Return value : False, if the jukebox window is not open.

4.6.9.2.2. Managing entries

Jukebox.DeleteEntry(...)

Description : Remove the entry from the jukebox list.
Parameters : Number : index of the entry from 1 to Jukebox.NumberOfEntries (page 583)
Optional : a boolean indicating if a confirmation will be asked to the user (false by default)
Return value : False, if the operation failed : for example, jukebox window is not open, user canceled the operation...

Jukebox.AddEntry(...)

Description : Add an entry to the jukebox list after the currently selected entry. The new entry become the current entry.
if the file already exist, it is not added.
Parameters : String : the complete path on the file
Return value : The entry index from 1 to to Jukebox.NumberOfEntries (page 583)
0, if the operation failed : for example, jukebox window is not open, file already exist.
Example :

```
--- Adding the front score to the jukebox
Jukebox.Open()
score=FrontScore()
if score~=nil then
   Jukebox.AddEntry(score.PathName..score.FileName)
end
```

Jukebox.GetEntryFileInfo(...)

Description : Provide miscellaneous informations about the file related to the entry.
As these informations are recorded into the file, you can't change them, except of course by loading the file and changing it.
Parameters : Number : index of the entry from 1 to Jukebox.NumberOfEntries (page 583)
Return value :
String : the complete path name on the file
Boolean : True if the file exist, false if the file don't exist on hard drive on recorded position
String : The full title of the file
String : The composer name of the file
String : The remarks related to the file
String : Author web site
String : Author email address
Number : Size in bytes of the file
Number : Duration in seconds of the music
Number : Number of seconds spend on this file
Boolean : True if the file is protected by his author
Number : Sound base level used to create this file from 0= low quality to 6= GOLD base
Example :

```
-- Displaying informations about files in jukebox list.
Jukebox.Open()
for i=1,Jukebox.NumberOfEntries do
   fileName=Jukebox.GetEntryFileInfo(i);
   print (fileName)
end
```

Jukebox.GetEntryAttribute(...)

Description : Provide the informations related to the entry and stored in the Jukebox list.
You can change these attributes using Jukebox.SetEntryAttribute(...) (above) method
Parameters : Number : index of the entry from 1 to Jukebox.NumberOfEntries (page 583)
Return value :
Number : Color index used to display the entry.
Number : Icon type related to the entry, as defined in MSDefine
JUKEBOX_ICON_STAR=0
JUKEBOX_ICON_SMILEY_LAUGH=1
JUKEBOX_ICON_SMILEY_COOL=2
JUKEBOX_ICON_SMILEY_HAPPY=3
JUKEBOX_ICON_SMILEY_SMILE=4
JUKEBOX_ICON_SMILEY_EMBARRASED=5
JUKEBOX_ICON_SMILEY_ANGRY=6
Number : Number of icons displayed
Boolean : True if the entry was already played in random mode (this avoid the same entry to be played twice)

Jukebox.SetEntryAttribute(...)

Description : Set the informations related to the entry and stored in the Jukebox list.
You can get these attributes using Jukebox.GetEntryAttribute(...) (above) method.
Parameters :
Number : index of the entry from 1 to Jukebox.NumberOfEntries (page 583)
Optional:
Number : Color index used to display the entry.
Number : Icon type related to the entry, as defined in MSDefine
Number : Number of icons displayed
Boolean : True if the entry was already played in random mode (this avoid the same entry to be played twice)
Return value : None

4.6.9.2.3. Managing Columns

Jukebox.GetColumnIdentifierName(...)

Description : Provide the name of a column identifier, in the current application language.
Parameters : Number : Column identifier as described in MSDefine
Return value :
String : The column identifier name
Example :

```
-- Display all column identifier name
Include "MSDefine"
for i=JUKEBOX_COLUMN_FILE_NAME,JUKEBOX_COLUMN_FILE_KIND do
   print(Jukebox.GetColumnIdentifierName(i));
end
```

Jukebox.SetSortingMode(...)

Description : Set the jukebox list sorting mode. When the mode is set the list is automaticaly sorted.
Parameters : Number : the column indentifier as described in MSDefine
(optional) Boolean : true if descending sort, false if ascending sort (true by default)
Return value : None
Example :

```
-- Sorting the jukebox list by date.
Include "MSDefine"
Jukebox.Open()
Jukebox.SetSortingMode(JUKEBOX_COLUMN_DATE,true)
```

Jukebox.GetSortingMode(...)

Description : Get the jukebox list sorting mode.
Parameters : None
Return value : Number : the column indentifier as described in MSDefine
Boolean : true if descending sort, false if ascending sort (true by default)

Jukebox.GetColumnInfo(...)

Description : Extract informations from a column
Parameters : Number : index of the column from 1 to n
Return value :
Boolean : True if the column is displayed, false if the column is not visible
Number : Column identifier as defined in MSDefine or nil if column index is out of range.
Number : Size of the column in pixel
Example :

```
-- Display all informations about columns
Include "MSDefine"
i=1
repeat
   active,type,size=Jukebox.GetColumnInfo(i)
   if active~=nil then
      if active==true then
         print ("Column '"..Jukebox.GetColumnIdentifierName(type).."' is active, the size is "..size)
      else
         print ("Column '"..Jukebox.GetColumnIdentifierName(type).."' is not active")
      end
   end
   i=i+1
until active==nil
```

Jukebox.SetColumnInfo(...)

Description : Set informations from a column.
Parameters : Number : index of the column from 1 to n
Boolean : True if the column is displayed, false if the column is not visible
Optional : Number : Column identifier as defined in MSDefine.
Optional : : Size of the column in pixel
Example :

```
Include "MSDefine"
-- Setting all active columns to the same size
i=1
repeat
    active,type,size=Jukebox.GetColumnInfo(i)
    if active~=nil then
        if active==true then
            Jukebox.SetColumnInfo(i,true,type,64)
        end
    end
    i=i+1
until active==nil
```

4.6.9.2.4. Playing

Jukebox.StartPlaying(...)

Description : Start playing the jukebox list from the current selected file in the list.
When the music will end, the next file will be played along the playing mode selected.
Parameters : None
Return value : True if a file was found and is now playing.
Example :

```
Jukebox.StartPlaying()
```

Jukebox.StopPlaying(...)

Description : Stop playing the jukebox list.
Parameters : None
Return value : True if a file was currently playing.
Example :

```
Jukebox.StopPlaying()
```

4.6.9.2.5. Browsing

Jukebox.SelectEntry(...)

Description : Select the entry
Parameters : Number : index of the entry from 1 to Jukebox.NumberOfEntries (page 583)
Return value : False, if the operation failed : for example, jukebox window is not open, entry not in range.

Jukebox.NextEntry(...)

Description : Skip to the next file in the list along the playing mode. For example, if JukeBox.IsRandomPlaying (page 583) is active then next file will not the next file in the list but a file selected randomly. If you want to selected the next file in the list use :
Jukebox.SelectEntry(Jukebox.CurrentEntry+1)
Parameters : None
Return value : The current entry selected in the list.

Jukebox.PreviousEntry(...)

Description : Skip to the previous file in the list.
Parameters : None
Return value : The current entry selected in the list.

4.6.9.2.6. Export

Jukebox.HTMLExport(...)

Description : Create HTML pages from the jukebox list.
Parameters : Target path
Boolean :false if myrweb.bin is generated, true if myrweb.html are generated (default)
Return value : None

4.7. Devices

This section describe how a script can manage devices like hard-drive, Midi Interface, Internet connexions.

4.7.1. File Object Type

General description of this object type : This object type and this set of global functions are related to files and folders
MyrScript language enables to access files using low-level functions for writing and reading raw data.
MyrScript language provides also high-level functions to read a music file in various format, save or export a music file, browse folders to find files...

Related methods

Some object methods load or save files. See : Score.Save(...) (page 240)

Security

A special mode forbides a script to apply some operations on sensitive folder, like system folders for example.
When this mode is active, opening a file for **writing**, **renaming**, **removing** outside :

- the user's data folder (aka the "Myriad Documents" folder),
- the Temporary folder (provided by GetTemporaryFilesPathName)
- the MyrScript Preferences folder (provided by GetPreferencesPathName)

opens automatically a dialog box, asking the user for a confirmation. The user can cancel the script command and the function will return an error to your script.
You ought to handle correctly this error in your script.
By default, this mode is active but the user can changes its state in the general setup, "Scripts" pannel.
This mode also protects the opening of external files or programs using OpenFileWithDefaultApplication(...) (page 596 any folder.

The access to musical files using the **LoadFile** function then the **Score.Save method** is not restricted.

Note about file names

Some operating systems do not accept file name length above 31 characters, so, be aware of this. You can use accented characters in your file name, but because the ASCII value for such caracters is different between Windows, Macintosh and Unix, it is not recommended to use such.
You are not allowed to use the character '/' (slash) in your file or path name because it is the standard MyrScript character separator.

In the same way, in order to make your file be easily managed on different operating system, it is recommended to give your file or folder a name made of :
· First character : 0 to 9, A to Z or a to z. (be careful, some operating systems are case-sensitive, others are not)
· Next characters (up to 26) : 0 to 9, A to Z, a to z or - (hyphen), _ (underscore) , space
· A dot (.)
· Three last characters, called file extension, that describe the file type, that can be 0 to 9, A to Z or a to z.

For example, you can give your file such a name : "My Custom-File.fil"
but not "My:Custom%File.?\|" because ":", "%", "?","\" and "|" characters are not allowed in file names.

Note about current path

When a script is launched, the current path is set to the script source file path.
When a script ends, the current path is restored.

Note about path names

Path names can change from a computer from another and from an operating system to another. Keep in mind that MyrScript can run on Mac OS, a Window System, or even an UNIX (MacOS X) system... For example, a path like "C:\Windows\Programs\Harmony Assistant" does not have any meaning on Macintosh...
So, MyrScript uses its own and standard way to notate path. The MyrScript path separator is '/'. The path above will become, on MyrScript :
"C:/Windows/Programs/Harmony Assistant".

Always use the standard methods to get the various path names from the user's folders and work relatively to these path.
For example do not perform :

```
SetCurrentPath("My HD With My Own Name/My settings on this computer/Blablabla/Myriad Documents/Settings/Models")
```

but instead :

```
SetCurrentPath(GetUserSettingsPathName("Models"))
```

Recommendation : when you want to change the current path, preserve the current value and restore it at the end of your script.

Note about character encoding

Myrscript provides automatic encoding translation for reading or writing text files in various character formats: MS/DOS, Windows, Mac Roman, Mac Shift-Jis, UTF-8 and UTF-16 in big or little endian.
Most of the time, you won't have to worry about this, because MyrScript will identify the proper encoding of the file you read and converts the data into a usable internal format.
In the same way, when writing a file MyrScript will use the UTF-8 encoding as a default, making your text files (script settings, etc) be fully compatible between Macintosh and Windows.
However, for more information about character encoding, please read the OpenFile(...) (page 600) method description

4.7.1.7. File global functions

4.7.1.7.1. Selecting

GetMusicFileNameLoad(...)

Description : Opens the standard file selector. The user can select any file from there. By default, only the Harmony files are listed but the user can select another file type in the pop-up menu.
Parameters : None
Return value : A complete file location, including the path followed by the name. A nil value if the user canceled the dialog box.
Example :

```
-- Asks for selecting a file and displays its name.
theFileName=GetMusicFileNameLoad()
if theFileName ~=nil then
   Message("Selected file is '"..theFileName.."'")
end
```

Note : Do not assume the file is a musical file, the user can select any kind of file.

GetFileNameLoad(...)

Description : Asks the user for selecting a file name for loading. Opens the standard file selector. The user can select any file. You can define the file type(s) the user will be able to choose
Parameters : A list of extensions separated by ','.
Return value : A complete file location including the path followed by the name. A nil value if the user canceled the dialog box.
Example :

```
-- Asks the user for selecting a .mus or .mu3 file and displays its name.
theFileName=GetFileNameLoad("mus,mu3,myr")
if theFileName ~=nil then
   Message("Selected file is '"..theFileName.."'")
end
```

GetFileNameSave(...)

Description : Asks the user for selecting a file name to save. Opens the standard file selector.
If the file already exists, a confirmation is asked.
Parameters : Optional : String : The default file name. If the file name include a complete path this will be the default path for the file.
string :the extension to add.
If no name is provided the default file name will be "Anonyme"
Return value : A complete file location including the path followed by the name. If an extension is provided the name ends with the extension. A nil value if the user canceled the dialog box.
Example :

```
-- Asks the user for selecting a file name for saving
theFileName=GetFileNameSave("MyMusic","myr")
if theFileName ~=nil then
   Message("Selected file is '"..theFileName.."'")
end
```

GetFolderName(...)

Description : Asks the user for selecting a folder. Open the standard folder selector.
Parameters : Optional : Default folder name. By default, last selected folder.
Return value : A complete path name for the selected folder or nil if the user had canceled.
Example :

```
-- Asks the user for selecting a folder name
theFolderName=GetFolderName()
if theFolderName ~=nil then
   Message("Selected folder is '"..theFolderName.."'")
end
```

4.7.1.7.2. Information

FileExist(...)

Description : Checks whether a file exists.
Parameters : A complete path on a file including its name and extension.
Return value : Boolean : true if the file exists, false if it does not.

FileSize(...)

Description : Provides the file size in bytes.
On Windows platform, because when read as ascii, the two-characters carriage return/line feed at end of line is translated to only one,
the result of this function may not be accurate if the file has been open in ascii mode.
Parameters : A complete path on a file including its name and extension.
Return value : Number : File size in bytes.
Example :

```
-- Asks the user for selecting a file and displays its size.
theFileName=GetMusicFileNameLoad()
if theFileName ~=nil then
   theFileSize=FileSize(theFileName)
   Message("Size of '"..theFileName.."'.." is "..theFileSize.." bytes")
end
```

FileModificationTime(...)

Description : Provides the last modification time of the file in system time format. You can use this value to compare two files and determinate the most recent. You can use the Application.ConvertSystemTimeToDate(...) (age 186) method to get a complete date from a system time.
Parameters : A complete path on a file including its name and extension.
Return value : Number : modification file time or 0 if file don't exist.
Example :

```
-- Asks the user for selecting a file and displays its modification date.
theFileName=GetFileNameLoad()
if theFileName ~=nil then
    Message("System time of the last modification of "..theFileName.."".." is "..FileModificationTime(theFileName)

end
```

4.7.1.7.3. Working on name

SplitName(...)

Description : Splits the complete file name into path, name and extension.
Parameters : A complete path on a file including its name and extension.
Return value : String : File path
String : File name
String : File extention
Example :

```
-- Asks the user for selecting a file and displays its path, name and extension
theFileName=GetMusicFileNameLoad()
if theFileName ~=nil then
    thePath,theName,theExtension=SplitName(theFileName)
    print("Path is "..thePath)
    print("Name is "..theName)
    print("Extension is "..theExtension)
end
```

AddExtension(...)

Description : Adds an extension to a complete file name.
Parameters : A complete path on a file, the extension to add.
Return value : a string
Example :

```
theFileName="HD Myriad/Documents/My documents.now/My music.mus"
theNewFileName=AddExtension(theFileName,"mid");
Message(theNewFileName)
```

ConvertFileExtensionToFileFormat(...)

Description : From a full name of a file (including path), analyse the extension and provide the file format :
FORMAT_STANDARD=1
FORMAT_EXPORT=2
FORMAT_MU3=3
FORMAT_MIDI0=4
FORMAT_MIDI1=5
FORMAT_KAR=6
FORMAT_RMTF=7
FORMAT_CMF=8
FORMAT_ABC=9
FORMAT_TAB=10
FORMAT_MUT=13
FORMAT_AIFF=14
FORMAT_WAV=15
FORMAT_SND=17
FORMAT_RAW=18
FORMAT_MP3=21
FORMAT_TEMPLATE=22
FORMAT_OGG=24 (experimental)
FORMAT_XML=26
FORMAT_XML_COMPRESSED=27
FORMAT_MXL=28
Parameters : A complete path on a file.
Return value : Number : file format or -1 if the extension don't match

4.7.1.7.4. Folders

BrowseFolder(...)

Description : Browses the hard disk and sends file names and optionaly folders name to your callback function. This enables to search for a file on your hard disk or to process all files included in a specific folder. User can break this process by pressing the computer keyboard "Esc" key.
Parameters : String: Folder location in which search will begin.
String: the mask which defines the files to process like "*.mus", "music*.*", etc. An empty string will select all files.

Function: A callback function to be called each time a file name matches the mask.

Callback function has to be declared as is :

function MyCallBackFunction(name)
or
function MyCallBackFunction(name,isFolder)

name is the complete name of the file (or the folder), including path and extension.
isFolder (optional) indicate if the name is a folder name
The callback function has to return true if the search must continue, false to stop browsing.
Optional:
Boolean: If true, sub-folders will be also processed recursively. By default false.
Boolean: If true, folders name will also be send to callback function. By default false.
Return value : true if browsing has explored all the files, false if the user has canceled the search or if your callback function has returned "false" to break the process.
Example :

```
--- This script displays all .myr file in a folder and its subfolders
--- Callback function declaration
function MyCallBackFunction(fileName)
   print(fileName)
return true
end
-- User is aked for selecting a folder name
folderName=GetFolderName()
if(folderName~=nil) then
   BrowseFolder(folderName,"*.myr",MyCallBackFunction,true)
end
```

```
--- This script displays all files and folder in a folder
--- Callback function declaration
function MyCallBackFunction(fileName,isFolder)
   print(fileName.." "..isFolder)
return true
end
-- User is aked for selecting a folder name
folderName=GetFolderName()
if(folderName~=nil) then
   BrowseFolder(folderName,"*.*",MyCallBackFunction,false,true)
end
```

Note : You are not allowed to add or remove files while browsing a folder.

CreateFolder(...)

Description : Creates a folder on your hard disk if this folder does not already exist. If the folder already exists, does nothing and returns no error.
Current path is not changed by this operation.
Parameters : String : the complete path name of this folder.
Return value : An error if the folder cannot be created, 0 means no error.
Example :

```
--- This example will create a "MySettings" folder in the user's "Myriad Documents/Settings" folder
local pathNameOnFolder
local err
pathNameOnFolder="MySettings"
SetCurrentPath(GetUserSettingsPathName())
--- Checks whether this folder already exists
if(IsFolderExist(pathNameOnFolder)==true) then
   Message("The : "..pathNameOnFolder.." folder already exists")
else
   err=CreateFolder(pathNameOnFolder)
   if(err==0) then
      Message(pathNameOnFolder.." folder is now created")
   else
      Alert("Cannot create "..pathNameOnFolder.." folder because of error # "..err)
   end
end
```

Note : For security reasons, a script cannot delete a folder.

IsFolderExist(...)

Description : Checks whether a folder exist on your hard disk.
Parameters : String : the complete path name of this folder.
Return value : Boolean : true if the folder exists, false if it does not

OpenFolderOnDesktop(...)

Description : Tell the operating system to open a window showing the content of a folder.
Parameters : String : the complete path name of this folder.
Return value : None
Example :

```
--- This sample open the  "Myriad Documents/Settings" folder
local pathNameOnFolder
pathNameOnFolder=GetUserExportPathName("Music Export")
OpenFolderOnDesktop(pathNameOnFolder);
```

4.7.1.7.5. Loading

Load(...)

Description : Loads a music file in a new document.
Parameters : A complete path on a file including its name and extension as returned by GetFileNameLoad.
Optional :
- · boolean indicating if the score window has to be automatically resized. By default the current value defined in the general setup.
- · boolean indicating if the score window has to be shown. True by default. If the score is not shown, you will be able to show it later using Score.ShowWindow(...) (page 250) method.
- · boolean indicating if the different warning (unknown fonte, unknow sound...) will be disabled (true by default).
- · boolean indicating if the last code on protected file will be recorded for the next access (false by default).
- · boolean telling whether instruments might be translated using the instrument translation table when current sound base is GOLD (true by default).
- · boolean telling if digital file dialog box will open (true by default).
- · boolean telling if alert must be shown when the file uses output devices that are not currently available (true by default)

Return value : A Score (page 198)
Example :

```
-- Asks the user for selecting a file and loads it
theFileName=GetFileNameLoad("mus,mu3")
if theFileName ~=nil then
   theScore=Load(theFileName)
end
```

Note : To save an already open score use the Score.Save(...) (page 240) method.

OpenFileWithDefaultApplication(...)

Description : Tell the operating system to open a document file with the default application.
For example, if you provide a .bmp picture file name, this will open the file with the application the user defined. If the user had activated the security mode and if the file extension is not a standard extension (mus, mu3, mid, bmp, pct, htm, html, pdf) a confirmation is asked.
Parameters : String : the complete name of the file including path.
Return value : None
Example :

```
-- This sample ask the user to select a
-- document or a program and open it.
local fileName=GetFileNameLoad()
if(fileName~=nil) then
   OpenFileWithDefaultApplication(fileName)
end
```

OpenFileWithApplication(...)

Description : Tell the operating system to open a document file with a specified application.
Parameters : String : the name of the application.
String : the complete name of the file including path.
Return value : None
Example :

```
-- This sample ask the user to select a music file and open it with ITunes.
local theFileName=GetFileNameLoad()
if(theFileName~=nil) then
   OpenFileWithApplication("ITunes",theFileName)
end
```

4.7.1.7.6. Paths

GetCurrentPath(...)

Description : Provides the current path. If you open a file with the standard MyrScript file function OpenFile(...) (page 600) access to the file will be done at this location.
Parameters : None
Return value : String : the full path name of the current location
Example :

```
Message("Current path is : "..GetCurrentPath())
```

SetCurrentPath(...)

Description : Changes the current path. If you open a file with the standard Lua file function OpenFile(...) (page 6 access to the file will be done at this location. Be aware not to use litteral path names in your script : hard disk folder tree changes from a computer to another.
Parameters : String : the full path name
Return value : Number: Error if the current path cannot be gotten, 0 if no error

GetDocumentationPathName(...)

Description : Provides the full path name of the **application documentation** folder. (Typically the "Docs" folder in the application folder.You can use directly this value with the SetCurrentPath(...) (above) method like SetCurrentPath(GetDocumentationPathName()), it is strongly recommended to preserve the current path before using the SetCurrentPath(...) (above) method.
Writing in this folder is forbidden.
Parameters : None
Return value : The complete path on the application documentation folder.
Example :

```
Message("Path of documentation folder is  : "..GetDocumentationPathName())
```

GetTemporaryFilesPathName(...)

Description : Provides the full path name on the **temporary files** folder. Files in this folder are cleared when the computer shuts down.Please note that the user can redefine this folder location using the application general setup.
You can use directly this value with the method SetCurrentPath(...) (above) like SetCurrentPath(GetTemporaryFilesPathName()), it is strongly recommended to preserve the current path before using the SetCurrentPath(...) (above) method.
Parameters : None
Return value : The complete path of the temporary files folder.
Example :

```
Message("Path of temporary files folder is  : "..GetTemporaryFilesPathName())
```

Note : On some operating systems, files in this folder are not visible.

GetPreferencesPathName(...)

Description : Provides the full path name on the **MyrScript preferences** folder. Typically the "MyrScript" folder in the preference folder.This folder is used to store special data you can read when your script start. For example the last values selected by the user.You can use directly this value with the SetCurrentPath(...) (above) method, like SetCurrentPath(GetPreferencesPathName()), it is strongly recommended to preserve the current path before using the SetCurrentPath(...) (above) method.

Important note : Because the content of this folder is shared between scripts, you have to be sure that the name you will use in your script will not be used by another script. Here is how to proceed to get an unique preference file name :
Your file name must begin with the first 5 characters of your personal registration code, followed by the name you want. For example, if my personal registration number is
XHOSZ-XXXX-XXXX-XXXX-XXXX and my script is named "portamento.mys". my script preference file will be called: "XHOSZPortamento.cfg"
Parameters : None
Return value : The complete path of the preference folder.
Example :

```
Message("Path of preference folder is  : "..GetPreferencesPathName())
```

GetUserExportPathName(...)

Description : Provides the full path name on the **user's export** folder. Typically the "???/Myriad Documents/Export" folder.
You can use directly this value with the method SetCurrentPath(...) (page 597) like SetCurrentPath(GetUserExportPathName("Lyrics")), it is strongly recommended to preserve the current path before using the SetCurrentPath(...) (page 597) method.
You can write in this folder.
Parameters : The sub-folder name you want to retrieve.
Predefined names are :
"Graphics Export","Lyrics","Music Export".
But you can create your own sub-folder using the name you want.
Return value : The complete path of the user export folder.
Example :

```
Message("Path of user's lyrics export folder is  : "..GetUserExportPathName("Lyrics"))
```

Note : This method does not ensure the sub-folder exists, it just creates a full path name on it. You can then use the CreateFolder(...) (page 595) method to create this path.

GetUserMusicPathName(...)

Description : Provides the full path name on the **user's music** folder. Typically the "???/Myriad Documents/Music" folder.
You can use directly this value with the method SetCurrentPath(...) (page 597) like SetCurrentPath(GetUserMusicPathName("")), it is strongly recommended to preserve the current path before using the SetCurrentPath(...) (page 597) method.
You can write in this folder.
Parameters : The sub-folder name you want to retrieve. There is no predefined sub-folder name. You can create your own sub-folder using the name you want.
Return value : The complete path on the user's music folder.
Example :

```
Message("Path on user's music folder is  : "..GetUserMusicPathName(""))
```

Note : This method does not ensure the sub-folder exists, it just creates a full path name on it. You can then use the CreateFolder(...) (page 595) method to create this path.

GetUserSettingsPathName(...)

Description : Provides the full path name on the **user's settings** sub-folder. Typically the sub-folder present in "Myriad Documents/Settings".
You can use directly this value with the method SetCurrentPath(...) (page 597) like SetCurrentPath(GetUserSettingsPathName()), it is strongly recommended to preserve the current path before using the SetCurrentPath(...) (page 597) method.
You can write in this folder.
Parameters : The setting sub-folder name you want to retreive.
Predefined names are :
"Accordions","Charsets","Chords","Drums","Engraver","Extra","FXs","Instruments","Keyboards","MidiMap","Models","Organs","Palettes","Rules","Scripts","Voices","Patterns".
But you can create your own setting sub-folder using the name you want.
Return value : The complete path on the user's settings folder.
Example :

```
Message("Path of user's Engraver settings is  : "..GetUserSettingsPathName("Engraver"))
```

Note : This method does not ensure the sub-folder exists, it just creates a full path name on it. You can then use the CreateFolder(...) (page 595) method to create this path.

GetSystemSettingsPathName(...)

Description : Provides the full path name on the **system settings** sub-folder. Typically a sub-folder present in the application install folder.
You can use directly this value with the method SetCurrentPath(...) (page 597) like SetCurrentPath(GetSystemSettingsPathName()), it is strongly recommended to preserve the current path before using the SetCurrentPath(...) (page 597) method.
You can write in this folder.
Parameters : The setting sub-folder name you want to retreive.
Predefined names are :
"Accordions","Charsets","Chords","Drums","Engraver","FXs","Keyboards","MidiMap","Models","Organs","Palettes","Scripts","Voices","Patterns".
Return value : The complete path on the user's settings folder.
Example :

```
Message("Path of system Virtual Singer voice settings is  : "..GetSystemSettingsPathName("Voices"))
```

Note : This method does not ensure the sub-folder exists, it just creates a full path name on it. You can then use the CreateFolder(...) (page 595) method to create this path.
Writing in this folder is forbidden.

4.7.1.7.7. Files

AdjustFileName(...)

Description : Processes a string so that it can be used as a file name
Parameters : String: desired file name (file name only, without path)
Return value : String: modified file name so that it could be used for a file
Example :

```
name="What?\r\n Do you know/think I'm a <*>?.txt"
print(AdjustFileName(name))
```

Note : Length is reduced to 216 characters, characters \/:*?<>"| are replaced by ~ and characters below space (0x20) in the ASCII range are replaced by "_"

AdjustPathName(...)

Description : Processes a string so that it can be used as a path+file name
Parameters : String: desired path name (folder names separated with "/", and ending by the file name)
Return value : String: modified path name so that it could be used as a valid path
Example :

```
name="C:/<Test>/What?\r\n Do you knowlthink I'm a <*>?.txt"
print(AdjustPathName(name))
```

Note : Length of each folder + file name is reduced to 216 characters, characters \/:*?<>"| are replaced by ~ (except : in position 2) and
characters below space (0x20) in the ASCII range are replaced by "_"

OpenFile(...)

Description : Opens a file. All open files **must** be closed using the Close method.
Parameters : String : File name or full path name of the file to open.
Optional : String : open mode. By default, read in text mode but you can specify :

- "r" read mode
- "w" write mode
- "a" append mode
- "r+" update mode, all previous data are preserved
- "w+" update mode, all previous data are erased;
- "a+" append update mode, previous data are preserved, writing is only allowed at the end of file.

An additional character, "b", indicates the opening is made in binary mode. The use of this indicator is deprecated. It just changes the default encoding if not explicitely specified.
It is now highly recommended to specify an encoding (next parameter) when opening a file for reading or writing.

Optional: Number: Character encoding. Can be one of the following constants (MSDefine):

- NO_STR_ENCODING : No conversion is made when reading/writing. Characters are assumed to be 8-bits. -default for "b" modes-
- UNKNOWN_STR_ENCODING : unknown encoding, to be determined by the program by analyzing the string content (warning, not 100% reliable)
- MS_DOS_STR_ENCODING : MS-DOS encoding (8 bits)
- WINDOWS_STR_ENCODING : Windows encoding (Windows 8 bits ANSI, occidental)
- MACINTOSH_STR_ENCODING : Macintosh encoding (Macintosh 8 bits ANSI, occidental) -default for " non-b" modes-
- HTML_STR_ENCODING : HTML encoding (7-bits ASCII entities, Unicode compatible)
- UTF8_STR_ENCODING: UTF-8 encoding (8 bits) recommended for storing characters from any charset.
- MAC_SHIFTJIS_STR_ENCODING : Shift-JIS (Japanese ANSI extension) Macintosh 8 bits encoding
- UTF16_BE_STR_ENCODING : 16 bits Unicode Big Endian encoding
- UTF16_LE_STR_ENCODING : 16 bits Unicode Little Endian encoding

Optional: Number: Carriage return kind. Can be one of the following constants (MSDefine):

- MAC_CR : Macintosh-style carriage return (\r) - default -
- WIN_CR : Windows-style carriage return (\r\n)
- UNKNOWN_CR : Carriage return style is unknown, let MyrScript guess it (for read accesses only)

When reading a file, it is recommended to specify explicitely the character and carriage return encodings of this file, when known.
Use of UNKNOWN_STR_ENCODING and UNKNOWN_CR have to be reserved to text file you are not sure about the format.
When writing a file, use of UNKNOWN_STR_ENCODING and UNKNOWN_CR is strictly forbidden.

A few examples:

- If you need to open for reading a regular text file written on Windows: use WINDOWS_STR_ENCODING and WIN_CR
- If you need to open for reading a regular text file written on Macintosh: use MACINTOSH_STR_ENCODING and MAC_CR
- If you need to open for reading a text file that has been written on an unknown platform: use UNKNOWN_STR_ENCODING and UNKNOWN_CR
- If you need to open for reading an UTF-8 text file that has been written on a Macintosh: use UTF8_STR_ENCODING and MAC_CR
- If you need to open for reading a binary file without doing any character translation: use NO_STR_ENCODING and MAC_CR
- If you need to open for writing an UTF-8 text file (script preferences, etc): use UTF8_STR_ENCODING and MAC_CR
- If you need to open for writing a Windows standard text file: use WINDOWS_STR_ENCODING and WIN_CR

Return value : File object.
Optional : In case of errors, File is equal to nil and a string is provided describing the error.
Example :

```
--- This sample opens a file for writing, writes two values : a string and a number
--- then opens again this file for reading and reads the values.
--- Local variable declarations
local myError
local myFile
local myString
local myNumber
---- Setting the folder where we want to work
SetCurrentPath(GetTemporaryFilesPathName())
--- Opening the file for writing
myFile,myError=OpenFile("MyFileName.txt","w",UTF8_STR_ENCODING)
--- Testing error
if(myFile==nil) then
   Message(myError)
else
   -- We write a string and a number
   myFile.WriteString("Mystring "..25);
   -- Closing the file
   myFile.Close()
   --- Opening the same file for reading
   myFile,myError=OpenFile("MyFileName.txt","r",UTF8_STR_ENCODING)
   --- Testing error
   if(myFile==nil) then
      Message(myError)
   else
      -- Reading the values
      myString,myNumber=myFile.Read("w","n")
      -- Closing the file
      myFile.Close()
      -- Shows the result
      Message("Read values are : "..myString..","..myNumber)
   end
end
```

Note : When opening a file for writing, depending on the kind of encoding you specified, a few bytes (called BOM) can be written automatically
at the beginning of the file.
When reading or writing a file the length or a string you read or write might not match the actual number of bytes (or words) actually read (or stored) in the file.
for example, a single UTF-8 can take up to 6 bytes. Therefore, you can't assume that the current file position +1 or -1 will represent the position of the next
or previous character in the file.

RemoveFile(...)

Description : Removes a file. The file is physically deleted. Of course, be careful when using this function...
Parameters : String : The file name or a full path name of the file, including the file name.
Return value : String : In case of error a string describing the error, nil if no error,
Example :

```
-- Asks the user for selecting a file and removes this file !
theFileName=GetFileNameLoad()
if theFileName ~=nil and Confirmation("DO YOU REALLY WANT TO REMOVE THE FILE "..theFileName)==true
then
   errorStr=RemoveFile(theFileName)
   if(errorStr~=nil) then
      Message(errorStr)
   end
end
```

SetFileSize(...)

Description : Change the physical file size on disk.The file must be an existing file, not opened.
If the size provided is lower than the actual file size, the file is truncated.
If the size is greater than the actual file size, data are added. The content of these data is undefined.
Parameters : String : The file name or a full path name of the file, including the file name.
Number: New size of the file
Return value : Number: error number (system dependent). 0 if no error

RenameFile(...)

Description : Change the name a file. You cannot use this function to move a file from a folder to another.
Parameters : ·String : A full path name of the file to rename
·String : The new file name, without the path name
Return value : In case of error a string describing the error, nil if no error,
Example :

```
-- Asks the user to select a file and renames the file.
theFileName=GetFileNameLoad("txt")
if theFileName ~=nil and Confirmation("DO YOU REALLY WANT TO RENAME THE FILE "'..theFileName..'" to '
test.txt")==true then
    errorStr=RenameFile(theFileName,"test.txt")
    if(errorStr~=nil) then
        Message(errorStr)
    end
end
```

TemporaryFileName(...)

Description : Returns a string with a file name that can safely be used for a temporary file. The file must be explicitly open before its use and removed when no longer needed.
This entry is supposed to provide a file name that does not already exist.
It can only be assumed to be true if you save this file in the temporary folder (GetTemporaryFilesPathName(...) (p
ge 597))
Parameters : None.
Return value : String: File name.
Example :

```
Message(TemporaryFileName());
```

UnpackFile(...)

Description : Unpack a file, from a start point and create an unpacked version
Parameters : String : File name or full path name of the file to unpack.
String : File name or full path name of the file to create.
Optional : Number : Starting position in source file (by default 1)
Return : Number error, 0 no error
- 5 buffer error
-10 Can't open source file
-11 Can't open target file
-12 No more mem

PackFile(...)

Description : Pack a file, from a start point and create a packed version
Parameters : String : File name or full path name of the file to pack.
String : File name or full path name of the file to create.
Optional : Number : Starting position in source file (by default 1)
Return : Number error, 0 no error
- 5 buffer error
-10 Can't open source file
-11 Can't open target file
-12 No more mem

GetFileLabel(...)

Description : Provide the label :
grey=1
green=2
purple=3
blue=4
yellow=5
red=6
orange=7
red=6
or 0 if file don't have label
MacOs only
Parameters : String : File name or full path name of the file.
Return : Label value, negative if error
-1= not implemented

CopyFile(...)

Description : Copies a file from one location to another. If the target file already exists, it is overwritten without warning.
Parameters : String : File name or full path name of the file to copy.
String : File name or full path name of the file to create.
Return : Number error, 0 no error
-10 Can't open source file
-11 Can't open target file
-12 No more mem

4.7.1.7.8. Private

F25092005(...)

Description : Reserved for system.

F05062009(...)

Description : Reserved for system.
Fonction privee pour decodage Finale
Pack a file, from a start point and create a packed version
File : File to read
Number : Size of block
String : File name or full path name of the file to create.
Return : Number error, 0 no error
-1 bad param
-2 can't open target
-3 can't allocate buff

4.7.1.8. File values

4.7.1.8.1. Encoding & charsets

File.Encoding

Type : Number
Description : Value of the current file charset/encoding.
can be one of the following constants (MSDefine):
- NO_STR_ENCODING : No conversion is made when reading/writing. Characters are assumed to be 8-bits. (not to be used for write accesses)
- MS_DOS_STR_ENCODING : MS-DOS encoding (8 bits)
- WINDOWS_STR_ENCODING : Windows encoding (Windows 8 bits ANSI, occidental)
- MACINTOSH_STR_ENCODING : Macintosh encoding (Macintosh 8 bits ANSI, occidental)
- HTML_STR_ENCODING : HTML encoding (7-bits ASCII entities, Unicode compatible)
- UTF8_STR_ENCODING: UTF-8 encoding (8 bits)
- MAC_SHIFTJIS_STR_ENCODING : Shift-JIS (Japanese ANSI extension) Macintosh 8 bits encoding
- UTF16_BE_STR_ENCODING : 16 bits Unicode Big Endian encoding
- UTF16_LE_STR_ENCODING : 16 bits Unicode Little Endian encoding

File.CREncoding

Type : Number
Description : Value of the current file carriage return type.
can be one of the following constants (MSDefine):
· MAC_CR : Macintosh-style carriage return (\r) - default -
· WIN_CR : Windows-style carriage return (\r\n)

4.7.1.8.2. Position

File.Position

Type : Number
Description : Sets and gets current read/write position in file, measured in bytes from the beginning of the file.
Example :

```
--- This sample opens a file for writing, writes 100 lines
--- then opens this file again for reading and display file position of each line.
--- Local variable declarations
local myError
local myFile
local myString
local position
---- Setting the folder
SetCurrentPath(GetTemporaryFilesPathName())
--- Opening file for writing
myFile,myError=OpenFile("MyFileName.txt","w")
--- Testing error
if(myFile==nil) then
   Message(myError)
else
   -- Writing data
   for i=0,100 do
      myFile.WriteString("Line "..i.."\r");
   end
   -- Closing file
   myFile.Close()
   --- Opening the same file for reading
   myFile,myError=OpenFile("MyFileName.txt","r")
   --- Testing error
   if(myFile==nil) then
      Message(myError)
   else
      -- Reading the file
      while(myFile.EndOfFile==false) do
         -- we record  the current position in the file
         position=myFile.Position
         -- we read a line
         myString=myFile.Read("l")
         --- if the line has been read (different from nil), it is printed
         if(myString~=nil) then
            -- Printing line
            print(position.." "..myString)
         end
      end
      -- Closing the file
      myFile.Close()
   end
end
```

File.EndOfFile

Type : Boolean
Description : True if end of file is reached while reading.

4.7.1.9. File methods

4.7.1.9.1. Disposing

File.Close(...)

Description : Closes a file previously open through OpenFile(...) (page 600). All data are flushed on disk. After th
you cannot access the File object.
Parameters : None.
Return value : 0 (noErr) if Close success, -1 if the file cannot be closed.
Example :

```
myFile.Close()
```

4.7.1.9.2. Files

File.Flush(...)

Description : If your file is opened for update, for example with the "r+" mode, you ought to call Flush between a
read and a write operation.
Parameters : None
Return value : None

4.7.1.9.3. Reading/Writing

File.WriteString(...)

Description : Write a string in the file.
Parameters : String : The string to be written. Carriage returns have to be indicated by \r.
(Optional) Boolean : true if the string has to be written as a raw string (default: false)
This overrides the encoding you defined for your file, and will store your string data as is, without any conversion.

For example,
[.] A regular styled MyrScript string will be stored in UTF-8 format, preceded by its whole style information block.
[.] When the file has been opened in HTML encoding, an HTML tag or entity can be explicitely written, without
translation of <, >, \r and & character to its displayable entity.
(Optional) String : Enclosing/Escapement characters (compatible with File.Read(...) (page 606))
[.] If this parameter contains only one character, this character is written before and after the string.
[.] If this parameter contains two characters, the first one is written before the string, and the second one after.
[.] If this parameter contains three character:
the first one is written before the string
the second one is the escapement character and is written before each occurrence of the first, second(
itself) or third character that is found in string.
the third one is written after the string

For example:
writing:
Hello world (and 100% of the universe)
using "/" as third parameter will write:
/Hello world (and 100% of the universe)/
on file
using "[]" as third parameter will write:
[Hello world (and 100% of the universe)]
on file
using "(/)" as third parameter will write:
(Hello world /(and 100% of the universe/))
on file
using "(%)" as third parameter will write:
(Hello world %(and 100%% of the universe%))
on file
Return value : String : if error, a string describing the error , nil if no error.
Example :

```
myFile.WriteString("This string will be written to the file")
```

File.WriteChar(...)

Description : Writes a character into a file. The character can range from 0 to 255. It is the only way to write any character in a binary file. For example, you cannot write a '0' using the File.WriteString(...) (page 605) method because the '0' is the ending character of a string.
Parameters : Number or Character : Character to write. If a number, represents the ASCII code of the character. Boolean : true if this character code is in Unicode range (2-bytes) and has to be translated into the file encoding (default=false)
Return value : String : if error, a string describing the error , nil if no error.

File.Read(...)

Description : Reads a text file according to the given format, which specifies what to read. For each format, the function returns a string (or a number), or nil if it cannot read data using the specified format. When called without format, it uses a default format that reads the next line (see below).
You can use the File.EndOfFile (page 604) value to know whether the end of file is reached.
Parameters : Available formats are

- "n" reads a number; this is the only format that returns a number instead of a string.
- "l" reads the next line (skipping the end of line), or nil at end of file. This is the default format.
- "a" reads the whole file, starting at the current position. From end of file, it returns the empty string.
- "w" reads the next word (maximal sequence of non-white-space characters), skipping spaces if necessary, or nil at end of file.
- any other non-numerical character: reads a string until end of file or the specified character is encountered
- any couple of non-numerical characters: skips data on file until the first character is matched, then reads until end of file or the second character is encountered
- number: reads a string with up to that number of characters, or nil at end of file.

This number is the number of bytes stored in the file, not the length of the resulting string (can depend on encoding)
Return value : 0 (noErr) if no error.
Example :

```
--- This sample asks the user for selecting a .mus file and copies it in the temporary folder under the same name
local myFileCompleteNameSource
local myFileCompleteNameTarget
local mySourcePath
local mySourceName
local mySourceExtension
local myFileSource
local myFileTarget
local myChar

--- Selects file
myFileCompleteNameSource=GetFileNameLoad("mus")
if myFileCompleteNameSource ~=nil then
   -- Opens file for reading in binary mode
   myFileSource=OpenFile(myFileCompleteNameSource,"rb")
   if myFileSource ~=nil then
      -- Extracts file name
      mySourcePath,mySourceName,mySourceExtension=SplitName(myFileCompleteNameSource)
      -- Opens file for writing in the temporary folder
      myFileTarget=OpenFile(GetTemporaryFilesPathName()..mySourceName.."."..mySourceExtension,"wb")
      if(myFileTarget~=nil) then
         -- Copies all data
         repeat
            myChar=myFileSource.ReadChar()
            if(myChar>=0) then
               myFileTarget.WriteChar(myChar)
            end
         until(myChar<0)
         myFileTarget.Close()
         Message("File "..myFileCompleteNameSource.." is now copied into "..GetTemporaryFilesPathName())
      end
      myFileSource.Close()
   end
end
```

File.ReadChar(...)

Description : Reads a character in a file. Character ranges from 0 to 255.
Parameters : (Optional) Boolean : true if this character code has to be in Unicode range (2-bytes) and is translated from the file encoding (default=false)
Return value : Number: character read from file from 0 to 255 (or 0-65535 if optional parameter is true) , -1 if the end of file is reached.
Example :

```
--- This sample opens a file for reading, and counts the number of characters equal to '0' in it
--- Local variable declarations
local myFile
local myFileName
local myCounter
local myChar

myFileName=GetMusicFileNameLoad()
if myFileName ~=nil then
   myFile=OpenFile(myFileName,"rb")
   if myFile ~=nil then
      myCounter=0
      repeat
         myChar=myFile.ReadChar()
         if(myChar==0) then
            myCounter=myCounter+1
         end
      until(myChar<0)
      Message("There are "..myCounter.." zero characters in the"..myFileName.." file")
      myFile.Close()
   end
end
```

File.ReadSignedChar(...)

Description : Reads a character in a file. Character ranges from -127 to 128.
Parameters : None
Return value : Number: character read from file

File.ReadShortIntel(...)

Description : Read a signed short (16 bits) in a file. Intel format (PL,PH).
Parameters : None
Return value : Number

File.ReadUnsignedShortIntel(...)

Description : Read an unsigned short (16 bits) in a file. Intel format (PL,PH).
Parameters : None
Return value : Number

File.ReadShortMotorola(...)

Description : Read a signed short (16 bits) in a file. Motorola format (PH,PL).
Parameters : None
Return value : Number

File.ReadUnsignedShortMotorola(...)

Description : Read an unsigned short (16 bits) in a file. Motorola format (PL,PH).
Parameters : None
Return value : Number

File.ReadLongIntel(...)

Description : Read a signed int32_t (32 bits) in a file, Intel format (PL,PH).
Parameters : None
Return value : Number

File.ReadUnsignedLongIntel(...)

Description : Read an uint32_t (32 bits) in a file, Intel format (PL,PH).
Parameters : None
Return value : Number

File.ReadUnsignedLongMotorola(...)

Description : Read an uint32_t (32 bits) in a file, Motorola format (PL,PH).
Parameters : None
Return value : Number

File.ReadLongMotorola(...)

Description : Read a int32_t (32 bits) in a file, Motorola format (PL,PH).
Parameters : None
Return value : Number

File.ReadRawTable(...)

Description : Reads a binary data block from a file into a RawTable. The RawTable is created and allocated in order to receive the block content.
Parameters : (Optional)
Number: number of data values (not necessarily bytes) to be read or 0 for all data till the end of file(default: 0)
String: Kind of data to read. Available formats are a combination of the following items:

- Data type:
  · "b" for a byte (8 bits). (default)
  · "w" for a word (16 bits).
  · "l" for a int32_t word (32 bits).
  · "f" for an IEEE floating-point value (32 bits).

- Sign (ignored if data type is "f")
  · "s" for a signed value. (default)
  · "u" for an unsigned value.

- Endian (only meaningful for "w", "l" and "f" data types)
  · "n" for a native endian, depending on the platform : big endian for PowerPC, little endian for Intel. (default)
  · "e" for little endian (Intel)
  · "E" for big endian (Motorola, PowerPC)

Return value : RawTable: created rawtable (must be disposed after use through RawTable.Dispose(...) (page 45

File.WriteRawTable(...)

Description : Write a binary data block into a file, from the content of a RawTable.
Parameters : RawTable: raw table to write
(Optional)
String: Kind of data to be written. Available formats are a combination of the following items:

- Data type:
  · "b" for a byte (8 bits). (default)
  · "w" for a word (16 bits).
  · "l" for a int32_t word (32 bits).
  · "f" for an IEEE floating-point value (32 bits).

- Endian (only meaningful for "w", "l" and "f" data types)
  · "n" for a native endian, depending on the platform : big endian for PowerPC, little endian for Intel. (default)
  · "e" for little endian (Intel)
  · "E" for big endian (Motorola, PowerPC)

Number: first index to be processed in raw table (default: 0)
Number: number of data to be written
Return value : String : if error, a string describing the error , nil if no error.

File.XMLParse(...)

Description : Parse a complete XML 1.0 file. The parsing always start from the beginning of the file.
To avoid problems with carriage returns and line feeds, the file must be opened in binary mode.
Managed characters encoding are: US-ASCII,UTF-8,UTF-16,ISO-8859-1.
Parameters : All handlers are optionals.
function: StartTag(name,attributes). This handler will be called for each start tag and empty tag. Each attribute is received as a couple of string in the atttributes array.
function: Text(value) This handler will be called for each text. The received value is the text in UTF-8 format. The line breaks are received in MyrScript format.
function: EndTag(name) This handler will be called for each end tag and empty tag.
For example suppose an XML line like this :

<note type="black" size="12">height</note>

The StartTag handler will be called with name="note", and attributes as an array of strings: "type","black","size","12".
Then, the Text handler will be called with value="height"
Then, the EndTag handler will be called with name="note"
Return value : Boolean, true if no error found in this file. If false,
String : explanation of the error
Example :

```
function ProcessStart(str,t)
   result=""
     for pos=1,getn(t),2 do
       result=result.." "..t[pos].."="..t[pos+1].."""
     end
   print("<"..str..result..">")
end
function ProcessEnd(str)
   print("<"..str.."/>")
end
function ProcessText(str)
   print("["..str.."]")
end
```

```
-- Ask the user to select a file
fileName=GetFileNameLoad("xml")
--- Opening it
file=OpenFile(fileName,"rb")
-- Parsing it
file.XMLParse(ProcessStart,ProcessEnd,ProcessText)
--Close it
file.Close()
```

4.7.1.9.4. Position

File.SetPos(...)

Description : Change the current position in the file
Parameters : Number : position to set
Return value : Boolean, true if the operation successfully completed, false otherwise (for example the position is outside the file data range)

File.GetPos(...)

Description : Return the current file position indicator value, -1 if error.
Parameters : None
Return value : Number : the posiiton

4.7.2. Midi Object Type

General description of this object type : This set of methods enables to manage Midi devices from a script.
You will be able to get "Midi in" events (when the user press a key on the Midi keyboard) and to send "Midi out" events (notes played on the Midi keyboard).
One MIDI input device is managed and twelve MIDI output devices.

4.7.2.1. Midi methods

4.7.2.1.1. Setup

Midi.Setup(...)

Description : Open the standard Midi configuration dialog box.
Parameters : None
Return value : True if the user validated his choice.
Example :

```
Midi.Setup()
```

Midi.EchoInstrument(...)

Description : Sets a program change/bank to the MIDI Echo
Parameters : Number: Channel 1-16 or 0 for all channels (default 0)
Number: Program Change number (default 0)
Number: Bank number (optional)
Return value : None
Example :

```
Midi.EchoInstrument(0,10)
```

4.7.2.1.2. Midi In

Midi.InBegin(...)

Description : Enable the Midi in. After this, all keystroke events on the Midi device are received from the Midi. GetEvent method.
Parameters : None
Return value : true if the Midi In device was correctly inited. false if no Midi device is available. You can use the Midi.Setup method to ask the user to define a valid Midi in device.
Example :

```
--- This sample initiate the Midi in and display the Note on and Note off events until
--- the user press the ESC key on the computer keyboard.
--- Please note the use of Application.ScriptBackGround(true) which allow this script to run in background
Include "MSDefine"
Application.ScriptBackGround(true)
print("Initiating the MIDI input device")
if Midi.InBegin()==true then
   print("Press the ESC Key to Quit")

   repeat

      event,channel,pitch,velocity,posms,postemps=Midi.GetEvent()
      if event>0 then
         if(event==1) then print("Note On "..pitch)
         elseif event==2 then  print("Note Off "..pitch)
         else print("Event "..event.." param1:"..pitch.."param2: "..velocity)  end
         print("ms="..posms.." postemps="..postemps)
      end
      key=Application.KeyStroke()
      if key==nil then key=-1 end
      until key==27

   Midi.InEnd()
else
   print("Can't initiate the Midi in device, please check your Midi setup")
end
print("End")
```

Midi.GetEvent(...)

Description : Scan the Midi in to get event from the Midi device.
Parameters : None
Return value : Type of event : 0 mean no event, 1 mean note on (key pressed), 2 mean not off (key released)
The channel from 0 to 15
The note pitch from 0 to 127
The velocity from 0 to 127
The time position in millisecond of the event
The time position in the playing score (-1 if no score currently playing)
Example :

```
event,channel,pitch,velocity,timeMs,timePos=Midi.GetEvent()
```

Midi.InEnd(...)

Description : Close the Midi in device previously open with Midi.InBegin
Parameters : None
Return value : None
Note : If the Midi.InBegin method failed, you don't need to use Midi.InEnd

4.7.2.1.3. Midi Out

Midi.OutBegin(...)

Description : Enable the Midi out. After this, you can send data to the midi device.
Parameters : None
Return value : None

Midi.OutEnd(...)

Description : Close the Midi out previously open with Midi.OutBegin
Parameters : None
Return value : None

Midi.NoteOn(...)

Description : Send a NoteOn event to the Midi. The note will begin to play on the MIDI device until a NoteOff event /
will be send with the same pitch.
Parameters :
Number : Note pitch from 0 to 127
Optional :
- Number : Note velocity from 0 to 127. (64 by default)
- Number : Channel number from 0 to 15 (0 by default)
- Number : Midi number from 0 to 11 (0 by default)
- Number : Finger number, to be displayed on Virtual Keyboard (default negative=none)
- String: Color of key highlight on Virtual Keyboard palette ("RRGGBB" components in hexadecimal)

Return value : None

Midi.NoteOff(...)

Description : Send a NoteOff event to the Midi. If a note is currently played with the pitch defined, the note will / begin to release.
Parameters :
Number : Note pitch from 0 to 127
Optional :
- Number : Channel number from 0 to 15 (0 by default)
- Number : Midi number from 0 to 11 (0 by default)

Return value : None

Midi.ProgramChange(...)

Description : Send a Program Change event to Midi Out. Will have no effect if Program Change is disabled in the advanced MIDI setup
Parameters :
Number : Program change number (0-127)
Optional :
- Number : Channel number from 0 to 15 (0 by default)
- Number : Midi number from 0 to 11 (0 by default)

Return value : None

Midi.BankChange(...)

Description : Send a Bank Change event to Midi Out. Will have no effect if Bank Change is disabled in the advanced MIDI setup
Parameters :
Number : Bank number (0-16383)
Optional :
- Number : Channel number from 0 to 15 (0 by default)
- Number : Midi number from 0 to 11 (0 by default)

Return value : None

Midi.Chorus(...)

Description : Set the chorus level for the channel
Parameters :
Optional :
- Number : Chorus for this channel from 0 to 127. (0 by default)
- Number : Channel number from 0 to 15 (0 by default)
- Number : Midi number from 0 to 11 (0 by default)

Return value : None

Midi.Reverb(...)

Description : Set the reverb level for the channel
Parameters :
Optional :
- Number : Reverb for this channel from 0 to 127. (0 by default)
- Number : Channel number from 0 to 15 (0 by default)
- Number : Midi number from 0 to 11 (0 by default)

Return value : None

Midi.Modulator(...)

Description : Set the modulator (vibrato) level for the channel
Parameters :
Optional :
- Number : Modulator for this channel from 0 to 127. (0 by default)
- Number : Channel number from 0 to 15 (0 by default)
- Number : Midi number from 0 to 11 (0 by default)

Return value : None

Midi.Expression(...)

Description : Set the expression level for the channel
Parameters :
Optional :
- Number : Expression for this channel from 0 to 127. (0 by default)
- Number : Channel number from 0 to 15 (0 by default)
- Number : Midi number from 0 to 11 (0 by default)

Return value : None

Midi.Panning(...)

Description : Set the panning (stereo) position for the channel
Parameters :
Optional :
- Number : Panning for this channel from 0 to 127. (64 by default)
- Number : Channel number from 0 to 15 (0 by default)
- Number : Midi number from 0 to 2 (0 by default)

Return value : None

Midi.PitchBend(...)

Description : Set the pitch bend value for the channel
Parameters :
Optional :
- Number : PitchBend for this channel from -8192 a 8192. (0 by default)
- Number : Channel number from 0 to 15 (0 by default)
- Number : Midi number from 0 to 11 (0 by default)

Return value : None

Midi.Pedal(...)

Description : Send a pedal effect to this channel
Parameters :
Optional :
- Number : Pedal value for this channel from 0=pedal off to 100=pedal on. (100 by default)
- Number : Channel number from 0 to 15 (0 by default)
- Number : Midi number from 0 to 11 (0 by default)

Return value : None

Midi.RawPacket(...)

Description : Send a raw MIDI data packet of 1 to 3 bytes
Parameters :
- Number : Channel number from 0 to 15
- Number : Midi number from 0 to 11
- Number : First data value to be sent
- Number : 2nd data value to be sent (Optional)
- Number : 3rd data value to be sent (Optional)

Return value : None

Midi.AllNotesOff(...)

Description : Stop all notes currently playing
Parameters : None
Return value : None

4.7.3. Internet Object Type

General description of this object type : The "Internet" object type offer you simple and fast methods to access to the Internet. A script will be able to open a Web page or send a mail.

4.7.3.1. Internet methods

4.7.3.1.1. Email

Internet.SendMailTo(...)

Description : Launch the current email application (as defined in the user system setup) and create a new email. The mail is not directly send but shown to the user. The user can complete the mail, send it or cancel.
Parameters : String : destination email address
Optional :
- String : the subject or nil if none
- String : the email address to send a carbon copy (do nothing on some systems, don't ask me why...)
- String : the body or nil if none
- String : full path of file to send or nil if none (no yet implemented) (do nothing on some systems)

Return value : Nothing
Example :

```
Internet.SendMailTo("John.Doe@myriad-online.com","This is little hello",nil,"Hello John, this is a little hello fro
MyrScript")
```

Note : You can't use this method to send multiple emails without confirmation.

4.7.3.1.2. Internet files

Internet.DownloadFile(...)

Description : Download a file from a web server and save it on the hard drive.
Parameters : String : serveur address (without http) for example "www.myriad-online.com"
String : page address relative to the serveur address for example "score/myscore.mus"
String : Full name (including path) of the downloaded file.
Return value : Error code

4.7.3.1.3. Web Pages

Internet.OpenWebPage(...)

Description : Launch the current Web browser application (as defined in the user system setup) and open the Web page address the script provide. If you omit the "http://" it will be automatically added.
Parameters : String : web page address
Return value : Nothing
Example :

```
Internet.OpenWebPage("www.myriad-online.com/index.htm")
```

4.7.4. Movie Object Type

General description of this object type : Overview
The object type Movie and its related methods allow you to apply operation on movies: displaying and playing movies, extracting tracks, adding sound tracks or pictures to a movie file, create texts synchronized with the movie.

On some systems, some capabilities can't be correctly handled. You must use Application.GetCapability(...) (page 197) to check if the capability is available and adjust your script.

A movie is a file constitued of differents parts (tracks).
Each track can be present or not : video track, sound track, text track.

The video track can be animated or static (pictures).

For a better understanding of these methods, we recommand to you to take a look on the SyncMovie script provided in Scripts>Video

Disposing movie
When the script which created the movie ends, the movie is automatically released, but it is recommanded to use Movie.Close(...) (page 620) method to close the movie.

Using system resources
The Movie object allow you to use system resources and particularly codecs. A codec is a part of the system which can decode and encode.
For example, default codecs on Macintosh encode and decode in various sound file formats (Including AAC with QuickTime pro).

Errors:
Some methods can return an error. Here is the list of errors.
0 no error
1 permission error
2 invalid duration
3 function not implemented
4 file format not managed

Special considerations about file types on Macintosh.
When you work on a .mov file (QuickTime file) you can add text, extract audio, inject audio.
When you work on a non-QuickTime file (like a .mpg), you can't extract the audio track.
On this kind of file (files readable by QuickTime but not real QuickTime file) adding text will mute this file in .mov file format.
Example :

```
-- This sample will load a movie file and play it inside a dialog window.
--
-- Selecting a movie file
theFileName=GetFileNameLoad()
if theFileName~=nil then
    -- Opening the movie file
    movie=OpenMovieFile(theFileName)
    if movie~=nil then
        -- Creating a dialog to display the movie
        myDialog=NewDialog("Esc key to quit",movie.Width,movie.Height)
        -- Showing the dialog
        myDialog.Show()
        -- We attach the movie to the dialog
        movie.AttachToDialog(myDialog)
        -- We will display it at the right size
        movie.SetArea(0,0,movie.Width,movie.Height)
        -- Start playing
        movie.Play()
        -- Looping until the movie reach the end or the user press the "esc' key
        repeat
            -- Managing idles
            movie.Idle()
            myDialog.Select()
        until (movie.TimePosition>= movie.Duration or Application.IsKeyCodePressed(53)==true)
        -- Closing movie
        movie.Close()
        -- Closing dialog
        myDialog.Dispose()
    end
end
```

4.7.4.1. Movie global functions

4.7.4.1.1. Files

OpenMovieFile(...)

Description : Open a movie file.
Parameters : The complete file name
Optional : Boolean, write access allwowed (false by default)
Return value : Movie object or nil if the file can't be handled.

CreateMovieFile(...)

Description : Create a new empty movie file and open it.
Parameters : The complete file name
Return value : Movie object or nil if the file can't be created.

4.7.4.1.2. Sound

SelectSoundCompression(...)

Parameters : Optional : You can provide default values.
Number: Sound compression format (system dependant) by default, none
Number: sample rate,by default 44100Hz
Number: number of channels, by default 1
Number: sample size (16 or 8) , by default 16
Number: bit rate, by default 128
Return value : Number: Sound compression format
Number: sample rate
Number: number of channels
Number: sample size (16 or 8)
Number: bit rate
String: opaque datas. This data are necessary to setup the codec.

If the user cancel the selection, no value is returned.

4.7.4.2. Movie values

4.7.4.2.1. Graphic

Movie.Width R

Type : Number
Description : Standard width of the movie (in pixel).

Movie.Height R

Type : Number
Description : Standard height of the movie (in pixel).

4.7.4.2.2. Managing

Movie.IsVideo R

Type : BooleanBoolean
Description : Check if a video track is stored into the movieCheck if a video track is related into the movie.

Movie.IsSound R

Type : Boolean
Description : Check if a sound track is stored into the movie

Movie.IsText R

Type : Boolean
Description : Check if a text track is stored into the movie

4.7.4.2.3. Sound

Movie.Volume

Type : Number
Description : Current sound volume from 0 (no sound) to 127 (full sound).

4.7.4.2.4. Time

Movie.TimePosition

Type : Number
Description : Sets and gets current time position (in millisecond). Changing the time position make the movie skip to the related frame.

Movie.Duration

Type : Number
Description : This is the total duration of the movie (in millisecond).

4.7.4.3. Movie methods

4.7.4.3.1. Graphic

Movie.SetArea(...)

Description : Specify the area where the movie will be displayed. The movie ratio is always preserved and the movie size is adjusted to the given area.
Parameters : Numbers, Left, Top, Right, Bottom, in pixel.
Return value : Movie object or nil if the file can't be handled.

Movie.AddImage(...)

Parameters : String : The image file name
Optional : Number : time position (in ms) where to insert the image, by default beginning of track
Number : duration (in ms) while the image will be displayed, by default up to the end of the track
Return value : Number : Movie error
Example :

```
-- This script will ask the user to select pictures files and will create a slideshow
-- where each image will be displayed 5 seconds.
-- Creating a movie file
Message("Step 1, select the movie file to create")
theMovieFileName=GetFileNameSave("slideshow","mov")
if theMovieFileName~=nil then
   -- Opening the movie file
   movie=CreateMovieFile(theMovieFileName)
   if movie~=nil then
      Message("Step 2, now select pictures file you want to add then press 'Cancel' to terminate.")
      position=0
      duration=5000 -- In ms
      repeat
         theImageFileName=GetFileNameLoad()
         if theImageFileName~=nil then
            movie.AddImage(theImageFileName,position,duration)
            position=position+duration
         end
      until  theImageFileName==nil
      -- Closing movie
      movie.Close()
      -- Opening with default application
      OpenFileWithDefaultApplication(theMovieFileName)
   end
end
```

Movie.SelectEffect(...)

Description : Open the system dependant dialog box and prompt the user to select an effect.
You can send the returned value to Movie.AddEffect(...) (above)
Parameters : None
Return value : String : Opaque description of the effect or nil if the user canceled the input.

Movie.AddEffect(...)

Description : Add an effect to the movie. Effect can only be applied to static images.
Parameters : String : opaque description of the effect as inputed with Movie.SelectEffect(...) (above)
Optional : Number : time position (in ms) where to insert the image, by default beginning of track
Number : duration (in ms) while the effect will be applied, by default up to the end of the track
Return value : Number : Movie error
Example :

```
-- This script will ask the user to select pictures files and will create a slideshow
-- where each image will be displayed 5 seconds.
-- A transition effect is asked to the user and applied from an image to another
-- Please note, the same effect is applied but you can, if you want, changes this script and
-- ask the user to select a different effect for each picture transition.
-- Creating a movie file
Message("Step 1, select the movie file to create")
theMovieFileName=GetFileNameSave("slideshow","mov")
if theMovieFileName~=nil then
   -- Opening the movie file
   movie=CreateMovieFile(theMovieFileName)
   if movie~=nil then
      Message("Step 2, now select the effect which will be applied from a picture to another.")
      theEffect=movie.SelectEffect()
      if theEffect~=nil then
         Message("Step 3, now select pictures file you want to add then press 'Cancel' to terminate.")
         position=0
         duration=5000 -- In ms
         repeat
            theImageFileName=GetFileNameLoad()
            if theImageFileName~=nil then
               movie.AddImage(theImageFileName,position,duration)
               movie.AddEffect(theEffect,position-1000,2000)
               position=position+duration
            end
         until  theImageFileName==nil
         -- Closing movie
         movie.Close()
         -- Opening with default application
         OpenFileWithDefaultApplication(theMovieFileName)
      end
   end
end
```

Movie.AttachToDialog(...)

Description : Attach the movie to the dialog window. This mean the movie will be displayed in this dialog.
Parameters : Dialog: the dialog
Return value : Number : Movie error

4.7.4.3.2. Managing

Movie.Play(...)

Description : Start playing the movie from the current time position
Parameters : one
Return value : None

Movie.Pause(...)

Description : Pause the movie. The image is frozen on the current frame. You can restart from the current position using Movie.Resume(...) (above) method.
Parameters : one
Return value : None

Movie.Resume(...)

Description : Resume after a pause. The movie start playing again from the current time position.
Parameters : one
Return value : None

Movie.Idle(...)

Description : This method must be called as frequently as possible to be sure the movie is played smoothly.
Parameters : one
Return value : None

Movie.Close(...)

Description : Dispose all datas related to the movie.
Parameters : one
Return value : None

Movie.RemoveTexts(...)

Description : All texts related to the movie are deleted. If no texts are related does nothing.
Parameters : one
Return value : None

Movie.RemoveSounds(...)

Description : All sounds related to the movie are deleted. If no sounds are related does nothing.
Parameters : one
Return value : Number : Movie error

Movie.RemoveVideo(...)

Description : All video related to the movie are deleted. If no video are related does nothing.
Parameters : Optional : a Time Position in ms. If specified, only video which intersect with this time position will be deleted.
Optional : a Time Duration in ms. If specified, only video which intersect with this time intervall will be deleted.
Return value : Number : Movie error

Movie.AddVideo(...)

Description : Experimental : add the video track of a movie to another movie
Parameters : Movie to add
Return value : Number : Movie error

Movie.RemoveEffects(...)

Description : All video effects related to the movie are deleted. If no video effects are related does nothing.
Parameters : Optional : a Time Position in ms. If specified, only effects which intersect with this time position will be deleted.
Optional : a Time Duration in ms. If specified, only effects which intersect with this time intervall will be deleted.
Return value : Number : Movie error

Movie.ValidateChanges(...)

Description : When changes has been applied to a movie, like adding texts, this validate the changes and store them in the movie file.
Parameters : one
Return value : Number : Movie error

4.7.4.3.3. Sound

Movie.ImportDigitalTrack(...)

Description : The digital track related to the movie is imported into the score as a digital staff.
Parameters : The score which will receive the digital track.
Return value : The left digital staff, or if the sound track related to the movie is monphonic, the digital staff
optional:
The right digital staff, if the sound track is stereophonic.
Example :

```
---Selecting file name
theFileName=GetFileNameLoad()
if theFileName ~=nil then
    -- opening the file
    movie=OpenMovieFile(theFileName,false)
    if movie==nil then
        Alert("Can't open this movie file")
    else
        score=NewScore()
        if score~=nil then
            movie.ImportDigitalTrack(score)
        end
        movie.Close()
    end
end
```

Movie.ExportDigitalTrack(...)

Parameters : Score to export. The score will be injected in the movie.
Optional :
· Boolean : The sound track will respect the movie duration : if the sound track duration exceed the movie duration, it will be cropped to the the movie duration.
· Number : Sound compression format (system dependant)
· Number : Sample rate. Values allowed are : 44100 (44 Khz) ,22050 (22 Khz),11025 (11 Khz). Other values will be converted to the closest legal value.
· Number : number of channel,1==mono, 2=stereo.
· Number : sample size. Values allowed are : 16 or 8.
· Number : Bitrate
· String : Opaque data for codec, as returned by SelectSoundCompression(...) (page 617)
Return value : Number, movie error
Example :

```
-- This script will use system resources (codec) to export the front score.
score=FrontScore()
if score==nil then
    Alert("Please load a score to export")
else
    compressionformat,samplerate,nbchannels,samplesize,bitrate,opaqueData=SelectSoundCompression(
    compressionformat,samplerate,nbchannels,samplesize,bitrate)
    if compressionformat~=nil then
        -- Selection du fichier a sauvegarder
        exportPath=GetUserExportPathName("Music Export")
        SetCurrentPath(exportPath)
        theFileName=GetFileNameSave(exportPath..score.FileName,"mov")
        if theFileName ~=nil then
            movie=CreateMovieFile(theFileName)
            if movie~=nil then
                err=movie.ExportDigitalTrack(score,false,compressionformat,samplerate,nbchannels,samplesize,
                bitrate,opaqueData)
                if err~=0 then
                    Alert("Can't write this sound track in this movie, erreur #"..err)
                end
                movie.Close()
            end
        end
    end
end
```

Movie.GetSoundCompression(...)

Description : Extract the compression mode of the sound track related to the movie
Parameters : None
Return value : No value if there is no sound track related to the movie or

Number: Sound compression format
Number: sample rate
Number: number of channels
Number: sample size (16 or 8)
Number: bit rate

4.7.4.3.4. Texts

Movie.AddText(...)

Description : Add a text at the specified time position in the movie. A text is defined by a string, an area where it will be draw,a time position when it will be displayed and a duration.
The movie file is not updated, you ought to call the Movie.ValidateChanges(...) (page 620) method when all texts added.
You can remove all texts in a movie using the Movie.RemoveTexts(...) (page 620) method.
Parameters : String : the text to display
Number : left position of the text area, in pixel (relative to movie area)
Number : top position of the text area, in pixel (relative to movie area)
Number : right position of the text area, in pixel (relative to movie area)
Number : bottom position of the text area, in pixel (relative to movie area)
Number : time position in ms, when the text will be displayed
Number : duration in ms, while the text will be displayed
Optional :
Number : text font
Number : text size
Number : text face
Number : text justification
Number : first character to select in the text, 1 based (will be displayed with the selected text color)
Number : number of character to select in the text (will be displayed with the selected text color)
Number : text color, by default blue
Number : selected text color, by default green (don't work on Macintosh with Quicktime, don't ask me why...
)
Number : frame text color, by default black
Return value : A movie error

5. Appendices

5.1. Keywords abstract

LUA language keywords

abs	acos	and	ascii	asin	assert
atan2	break	call	ceil	closestate	collectgarbage
cos	deg	do	dofile	doremote	dostring
elseif	end	equal	error	exp	floor
foreach	foreachi	format	frexp	function	gcinfo
getinfo	getlocal	getn	getref	gettagmethod	globals
hash	if	in	ldexp	limits	listcode
liststrings	loadlib	local	log	log10	max
mod	newstate	newtag	newuserdata	next	nil
or	print	querystr	querytab	rad	random
rawget	rawgettable	rawset	rawsettable	ref	repeat
setcallhook	setglobal	setlinehook	setlocal	settag	settagmethod
sort	sqrt	strbyte	strchar	strfind	strifind
strlower	strrep	strsub	strupper	tag	tan
tinsert	tonumber	tostring	totalmem	tremove	type
unref	until	while			

MyrScript Objects

AddRawChar(...) (page 176)|BinAnd(...) (page 174)
BinOr(...) (page 174)|BinShl(...) (page 174)
BinShr(...) (page 174)|BinXor(...) (page 174)
BitClr(...) (page 175)|BitSet(...) (page 175)
BitTst(...) (page 175)|ConvertStringEncoding(...) (page 175)
DeleteRawChar(...) (page 177)|DeleteStringStyle(...) (page 178)
GetLocalCopy(...) (page 177)|GetRawChar(...) (page 176)
GetStringStyle(...) (page 178)|InsertStringStyle(...) (page 178)
PurgeStringStyle(...) (page 177)|PutLocalCopy(...) (page 177)
RawStrlen(...) (page 176)|SetErrorMessage(...) (page 174)
SetRawChar(...) (page 176)|SetStringStyle(...) (page 178)

'Score' Object type

Global functions:

FrontScore(...) (page 199)
GetAlreadyOpenedScore(...) (page 200)
NewScoreFromTemplate(...) (page 199)

Values

Score.AccidentalAppliesToAllOctaves (page 211) [RW]
Score.AmbitusHorizontalOffset (page 200) [RW]
Score.AmbitusLineStyle (page 200) [RW]
Score.AmbitusLineThickness (page 200) [RW]
Score.AuthorWebPage (page 204) [RW]
Score.AutomaticTargetCaptions (page 220) [RW]
Score.BeamSplitOnHalfBeat (page 213) [RW]
Score.BraceCenterOffset (page 218) [RW]
Score.BracketEdgeOffset (page 218) [RW]
Score.BracketWidth (page 218) [RW]
Score.BreakSymbolDashLineLength (page 212) [RW]
Score.BreakSymbolDashSpaceLength (page 212) [RW]
Score.BreakSymbolSpaceBetweenDoubleBarlines (page 212) [RW]
Score.BreakSymbolSpaceBetweenHeavyLineAndThinLine (page 212) [RW]
Score.ChordDiagramBarreFretNumberTextScale (page 217) [RW]
Score.ChordDiagramFingerBarreScale (page 217) [RW]
Score.ChordDiagramFingerTextScale (page 217) [RW]
Score.ChordDiagramFontColor (page 217) [RW]

Score.ChordDiagramFontName (page 217) [RW]
Score.ChordDiagramFontSize (page 217) [RW]
Score.ChordDiagramNameTextScale (page 217) [RW]
Score.ChordDiagramNutSize (page 217) [RW]
Score.ChordGridFontColor (page 218) [RW]
Score.ChordGridFontFace (page 218) [RW]
Score.ChordGridFontSize (page 218) [RW]
Score.ChordGridSetup (page 209) [R_]
Score.CreationDate (page 204) [R_]
Score.CurrentDisplayDpi (page 201) [RW]
Score.DisplayGhostRests (page 211) [RW]
Score.DotScale (page 212) [RW]
Score.DownwardTieDeltaXEnd (page 216) [RW]
Score.DownwardTieDeltaXStart (page 215) [RW]
Score.DownwardTieDeltaYEnd (page 216) [RW]
Score.DownwardTieDeltaYStart (page 215) [RW]
Score.DownwardTieWidth (page 215) [RW]
Score.DrumAsRect (page 218) [RW]
Score.DrumGridCellBarLineSize (page 219) [RW]
Score.DrumGridTimeBarLineSize (page 218) [RW]
Score.EffectWidth (page 217) [RW]
Score.EngraverSetup (page 202) [R_]
Score.ExtendParts (page 211) [RW]
Score.FileName (page 203) [RW]
Score.FirstFreeObject (page 202) [R_]
Score.FirstInstrument (page 206) [R_]
Score.FirstStaff (page 208) [R_]
Score.FirstStavesGroup (page 219) [R_]
Score.FirstView (page 221) [R_]
Score.FrameTargetCaptions (page 220) [RW]
Score.GeneralVolume (page 210) [RW]
Score.GraceNoteGap (page 213) [RW]
Score.HairPinWidth (page 218) [RW]
Score.HorizontalDotAdjustementForUpstemFlag (page 212) [RW]
Score.Instruments (page 206) [R_]
Score.IsAccompaniementsMatrixDisplayed (page 201) [RW]
Score.IsChanged (page 203) [RW]
Score.IsChordGridComputedOutsideModalArray (page 200) [RW]
Score.IsChordGridDisplayed (page 201) [RW]
Score.IsChordGridLeftAligned (page 201) [RW]
Score.IsDisplayComments (page 204) [RW]
Score.IsDisplayControlHandles (page 201) [RW]
Score.IsKeepFirstPageAside (page 209) [RW]
Score.IsLoopPlayActive (page 210) [RW]
Score.IsMusicPaused (page 210) [R_]
Score.IsPlayAccompanimentsActive (page 210) [RW]
Score.IsPlaySelectionActive (page 210) [RW]
Score.IsPlayViewActive (page 210) [RW]
Score.IsProtectedAgainstExport (page 203) [R_]
Score.IsProtectedAgainstGraphicalExport (page 203) [R_]
Score.IsProtectedAgainstSaving (page 203) [R_]
Score.IsRelatePageTwoByTwo (page 209) [RW]
Score.IsRythmMatrixDisplayed (page 201) [RW]
Score.IsSelected (page 219) [R_]
Score.JoinParts (page 211) [RW]
Score.LastBarAction (page 205) [R_]
Score.LastInstrument (page 206) [R_]
Score.LastSelectedStaff (page 219) [RW]
Score.LastTarget (page 220) [R_]
Score.LastView (page 221) [R_]
Score.LyricDashLineLength (page 215) [RW]
Score.LyricDashLineThickNess (page 214) [RW]
Score.LyricMinimalSpaceBetweenSyllable (page 215) [RW]
Score.LyricSpaceBetweenDashLineAndLetter (page 215) [RW]
Score.LyricUnderLineThickNess (page 215) [RW]
Score.MetronomeInstrumentForBarTick (page 207) [RW]
Score.MetronomePitchForBarTick (page 207) [RW]
Score.MetronomePitchForQuarterNoteTick (page 207) [RW]
Score.MinimumVerticalSpacingBetweenAccidental (page 211) [RW]
Score.ModificationDate (page 204) [R_]
Score.MultiBarTextSize (page 214) [RW]
Score.MultiRestEdgeLineHeight (page 214) [RW]
Score.MultiRestTextSize (page 214) [RW]
Score.Next (page 205) [R_]
Score.NumberOfBars (page 202) [R_]
Score.NumberOfChordDiagrams (page 201) [R_]
Score.NumberOfInstruments (page 205) [R_]
Score.NumberOfPlayedBars (page 202) [R_]
Score.NumberOfStaves (page 208) [R_]
Score.NumberOfStavesGroups (page 220) [R_]
Score.OldAccidentalCancellation (page 211) [RW]
Score.OldQuarterRest (page 211) [RW]
Score.PaperHeight (page 209) [RW]
Score.PaperLeftMargin (page 209) [RW]
Score.PaperRightMargin (page 209) [RW]
Score.PaperTopMargin (page 209) [RW]
Score.PathName (page 203) [R_]
Score.Previous (page 200) [R_]

Score.Remarks (page 202) [R_]
Score.SlurBendModerator (page 217) [RW]
Score.SlurMaximalBend (page 216) [RW]
Score.SlurMinimalBend (page 216) [RW]
Score.SpaceAfterKeySignature (page 213) [RW]
Score.SpaceAfterTimeSignature (page 213) [RW]
Score.SpaceBeforeKeySignature (page 212) [RW]
Score.SpaceBeforeMultiBar (page 214) [RW]
Score.SpaceBeforeTimeSignature (page 213) [RW]
Score.SpaceBetweenAccidentalAndNote (page 211) [RW]
Score.SpaceBetweenDotAndDot (page 212) [RW]
Score.SpaceBetweenDotAndNote (page 211) [RW]
Score.SpaceBetweenKeySignatureAccidentals (page 213) [RW]
Score.StaccatoWidth (page 217) [RW]
Score.StaffLineThickNess (page 214) [RW]
Score.StaffRightLineThickNess (page 214) [RW]
Score.StavesGroups (page 219) [R_]
Score.StemLineThickNess (page 213) [RW]
Score.StorageArea (page 202) [R_]
Score.StreetOrganSetup (page 208) [R_]
Score.TabHarmonicaArrowLineThickNess (page 215) [RW]
Score.TabHarmonicaTriangleLineThickNess (page 215) [RW]
Score.TargetCaptionsMode (page 220) [RW]
Score.TargetCaptionsTextStyle (page 220) [R_]
Score.TextFrameMargin (page 218) [RW]
Score.TextFrameThickness (page 218) [RW]
Score.TimeBeginSelection (page 219) [RW]
Score.TimeEndSelection (page 219) [RW]
Score.Title (page 202) [R_]
Score.TransposedMode (page 201) [RW]
Score.UpwardTieDeltaXCurve (page 216) [RW]
Score.UpwardTieDeltaXEnd (page 216) [RW]
Score.UpwardTieDeltaYCurve (page 216) [RW]
Score.UpwardTieDeltaYEnd (page 216) [RW]
Score.UpwardTieTension (page 216) [RW]
Score.UpwardTieWidth (page 216) [RW]
Score.VerticalOffsetMultiBar (page 214) [RW]
Score.VerticalOffsetMultiBarNumber (page 214) [RW]
Score.VerticalOffsetMultiRestNumber (page 214) [RW]
Score.Views (page 220) [R_]
Score.WindowHeight (page 221) [RW]
Score.WindowHorizontalPosition (page 221) [RW]
Score.WindowRightMargin (page 221) [R_]
Score.WindowTopMargin (page 222) [R_]
Score.WindowWidth (page 221) [RW]

Methods

Score.AbsolutePixelToWrittenTime(...) (page 227)
Score.AddChordDiagram(...) (page 225)
Score.AddSelection(...) (page 241)
Score.AddStaff(...) (page 243)
Score.AddTemporaryStaff(...) (page 244)
Score.AdjustLocalMetrics(...) (page 235)
Score.AskPassword(...) (page 240)
Score.AutoScale(...) (page 250)
Score.AutoView(...) (page 238)
Score.BarRangeInSystem(...) (page 245)
Score.BarToTime(...) (page 226)
Score.BeamSelection(...) (page 242)
Score.BringToFront(...) (page 250)
Score.ClearTagFreeObjects(...) (page 247)
Score.ClearTagPictures(...) (page 247)
Score.ClearTagSymbols(...) (page 247)
Score.ComputeNormalizedVolume(...) (page 238)
Score.ComputePageSetup(...) (page 239)
Score.CopySelection(...) (page 241)
Score.CopyTo(...) (page 235)
Score.DeleteBars(...) (page 223)
Score.DeleteChordDiagrams(...) (page 225)
Score.DeleteLocalMetrics(...) (page 235)
Score.DeleteTemporaryStaves(...) (page 243)
Score.DeselectAll(...) (page 240)
Score.DeselectIndividually(...) (page 240)
Score.Dispose(...) (page 235)
Score.DoPrinterJob(...) (page 239)
Score.EditBarActions(...) (page 224)
Score.EditChordGrid(...) (page 224)
Score.EditGeneralTempo(...) (page 248)
Score.EditGlobalDigitalEffects(...) (page 237)
Score.EditViews(...) (page 250)
Score.EraseSelection(...) (page 241)
Score.GetBarTimeSignatureChange(...) (page 234)
Score.GetChordDiagramFromIndex(...) (page 225)
Score.GetCurrentTempo(...) (page 248)
Score.GetCurrentView(...) (page 249)
Score.GetGlobalBarSetting(...) (page 234)
Score.GetInstrumentByName(...) (page 236)
Score.GetPlayedBars(...) (page 237)
Score.GetPlayedDuration(...) (page 226)

Score.GetPrintMode(...) (page 251)
Score.GetStaffInSystem(...) (page 245)
Score.GetWrittenDuration(...) (page 226)
Score.GraphicalExport(...) (page 233)
Score.HumanizeSelection(...) (page 242)
Score.InsertBarAction(...) (page 223)
Score.InsertTarget(...) (page 247)
Score.JumpToBar(...) (page 238)
Score.LoadEngraverSetup(...) (page 228)
Score.MoveWindow(...) (page 250)
Score.MsToPlayedTime(...) (page 227)
Score.MusicExport(...) (page 229)
Score.NewView(...) (page 249)
Score.NumberOfStavesInSystem(...) (page 245)
Score.OutputLevel(...) (page 237)
Score.PasteSelection(...) (page 241)
Score.PlayedBarToPlayedTime(...) (page 226)
Score.PlayedBarToWrittenBar(...) (page 227)
Score.PlayedTimeToPlayedBar(...) (page 226)
Score.PlayedTimeToWrittenTime(...) (page 227)
Score.PositionView(...) (page 238)
Score.PreparePageForPrinting(...) (page 239)
Score.Print(...) (page 239)
Score.PrintPendingPages(...) (page 239)
Score.QuantizeSelection(...) (page 242)
Score.RegenerateTargets(...) (page 248)
Score.Save(...) (page 240)
Score.SaveEngraverSetup(...) (page 228)
Score.ScrollMusic(...) (page 237)
Score.SearchFreeObjectsByTag(...) (page 247)
Score.SearchPicturesByTag(...) (page 247)
Score.SearchSymbolsByTag(...) (page 246)
Score.SelectPaperArea(...) (page 234)
Score.SetBackGroundPicture(...) (page 222)
Score.SetGeneralTempo(...) (page 248)
Score.SetPageMode(...) (page 251)
Score.SetPlayMark(...) (page 237)
Score.SetPositionAtPage(...) (page 239)
Score.ShowWindow(...) (page 250)
Score.SizeWindow(...) (page 250)
Score.SubstTokenString(...) (page 240)
Score.SystemArea(...) (page 246)
Score.TimeToBeat(...) (page 226)
Score.TimeToChord(...) (page 225)
Score.UserChooseStaves(...) (page 242)
Score.WebExport(...) (page 230)
Score.WrittenTimeToAbsolutePixel(...) (page 227)
Score.WrittenTimeToPlayedTime(...) (page 227)

'GlobalBarSetting' Object type

Values

GlobalBarSetting.ActualTimeSignatureDenominator (page 318) [RW]
GlobalBarSetting.ActualTimeSignatureNumerator (page 318) [RW]
GlobalBarSetting.DisplayedTimeSignatureDenominator (page 318) [RW]
GlobalBarSetting.DisplayedTimeSignatureNumerator (page 318) [RW]
GlobalBarSetting.DontBreakLine (page 320) [RW]
GlobalBarSetting.ForcedNumbering (page 320) [RW]
GlobalBarSetting.IsBeamedTuplets (page 319) [RW]
GlobalBarSetting.IsBeamEighthsByFour (page 319) [RW]
GlobalBarSetting.IsDisplayedTimeSignature (page 318) [RW]
GlobalBarSetting.IsEndOfLine (page 320) [RW]
GlobalBarSetting.IsFirstOfLine (page 320) [R_]
GlobalBarSetting.IsForcedDisplay (page 319) [RW]
GlobalBarSetting.IsInvisible (page 320) [R_]
GlobalBarSetting.IsSplitMultiRest (page 320) [RW]
GlobalBarSetting.LeftMargin (page 319) [RW]
GlobalBarSetting.NumberOfBeat (page 319) [R_]
GlobalBarSetting.Scale (page 319) [RW]

Methods

GlobalBarSetting.CopyTo(...) (page 320)

'Staff' Object type

Values

Staff.AutoChordNumberOfNotes (page 258) [RW]
Staff.AutoSizeLyrics (page 257) [RW]
Staff.BarNumberingFirstBar (page 255) [RW]
Staff.BarNumberingFont (page 254) [RW]
Staff.BarNumberingPlaceToPlace (page 255) [RW]
Staff.BarNumberingPrefix (page 255) [RW]

Staff.BarNumberingSize (page 254) [RW]
Staff.BarNumberingType (page 254) [RW]
Staff.BarSettings (page 261) [R_]
Staff.BaseLine (page 270) [RW]
Staff.BeamGap (page 266) [RW]
Staff.BeamThickness (page 266) [RW]
Staff.CharSet (page 265) [RW]
Staff.ChordAlignment (page 260) [RW]
Staff.ChordDiagramNeck (page 259) [RW]
Staff.ChordFace (page 259) [RW]
Staff.ChordFrequencyType (page 260) [RW]
Staff.ChordFrequencyValue (page 260) [RW]
Staff.ChordRGBColor (page 260) [RW]
Staff.ChordSize (page 259) [RW]
Staff.ChordTitle (page 259) [RW]
Staff.ChordTitleFace (page 259) [RW]
Staff.ChordTitleRGBColor (page 259) [RW]
Staff.ChordTitleSize (page 259) [RW]
Staff.ChordYPosition (page 260) [RW]
Staff.Curves (page 262) [R_]
Staff.DefaultStemDirection (page 266) [RW]
Staff.DiagramColor (page 259) [RW]
Staff.DigitalTrack (page 255) [R_]
Staff.DisplayableLyricsText (page 257) [R_]
Staff.Dynamics (page 261) [R_]
Staff.ExtraSpaceBetweenLyricLines (page 266) [RW]
Staff.FirstCurve (page 262) [R_]
Staff.FirstDiagram (page 258) [R_]
Staff.FirstFXProcessor (page 261) [R_]
Staff.FirstMergedStaff (page 268) [R_]
Staff.FirstRule (page 263) [R_]
Staff.FirstSelectedSymbol (page 269) [RW]
Staff.FirstText (page 266) [R_]
Staff.FXProcessors (page 261) [R_]
Staff.GraphRecord (page 256) [R_]
Staff.GroupsMargin (page 271) [R_]
Staff.HiddenTransposition (page 268) [RW]
Staff.InChordGrid (page 270) [RW]
Staff.Instrument (page 267) [RW]
Staff.InUserChoice (page 270) [R_]
Staff.IsAutoChord (page 258) [RW]
Staff.IsAutoChordBarreFirst (page 258) [RW]
Staff.IsAutoChordFromTablature (page 258) [RW]
Staff.IsAutoChordOnePerBar (page 258) [RW]
Staff.IsBarNumberingAfterKey (page 255) [RW]
Staff.IsBarNumberingFrame (page 255) [RW]
Staff.IsChanged (page 268) [RW]
Staff.IsChord (page 258) [RW]
Staff.IsChordFrame (page 260) [RW]
Staff.IsDisplayOnlyIfFirstStaffOfGroup (page 255) [RW]
Staff.IsDrawChordDiagram (page 258) [RW]
Staff.IsDrawChordDiagramAbove (page 259) [RW]
Staff.IsDrawLeftLimit (page 263) [RW]
Staff.IsDrawOrnaments (page 265) [RW]
Staff.IsGhostRestsDisplayedAsWholeRests (page 265) [RW]
Staff.IsGraceNoteStemsAligned (page 265) [RW]
Staff.IsHideChord (page 267) [RW]
Staff.IsHideChordDiagram (page 267) [RW]
Staff.IsHideFXProcessor (page 267) [RW]
Staff.IsHideGraphic (page 267) [RW]
Staff.IsHideRests (page 265) [RW]
Staff.IsHideText (page 267) [RW]
Staff.IsLyrics (page 256) [RW]
Staff.IsLyricsAutoFormat (page 257) [RW]
Staff.IsMergedWithNext (page 268) [R_]
Staff.IsMultiVoice (page 268) [R_]
Staff.IsPlayChord (page 259) [RW]
Staff.IsPlayed (page 270) [RW]
Staff.IsRestCentered (page 265) [RW]
Staff.IsSelected (page 270) [R_]
Staff.IsUncutNotes (page 265) [RW]
Staff.LastBarSetting (page 261) [R_]
Staff.LastDiagram (page 258) [R_]
Staff.LastDynamic (page 261) [R_]
Staff.LastMergedStaff (page 268) [R_]
Staff.LastPicture (page 263) [R_]
Staff.LastSelectedSymbol (page 269) [RW]
Staff.LastSymbol (page 264) [R_]
Staff.LedgerLineSize (page 264) [RW]
Staff.LineColor (page 264) [RW]
Staff.LineSpacing (page 270) [R_]
Staff.LowerBarlineLimit (page 264) [RW]
Staff.LyricsFace (page 256) [RW]
Staff.LyricsFont (page 256) [RW]
Staff.LyricsRGBColor (page 256) [RW]
Staff.LyricsSize (page 256) [RW]
Staff.LyricsTitle (page 256) [RW]
Staff.LyricsTitleFace (page 256) [RW]

Staff.LyricsTitleRGBColor (page 256) [RW]
Staff.LyricsTitleSize (page 256) [RW]
Staff.LyricsYPosition (page 257) [RW]
Staff.MaximumBeamTilt (page 266) [RW]
Staff.NeverHide (page 268) [RW]
Staff.Next (page 252) [R_]
Staff.NumberOfChords (page 260) [R_]
Staff.NumberOfDiagrams (page 257) [R_]
Staff.NumberOfFXProcessor (page 261) [R_]
Staff.NumberOfPictures (page 262) [R_]
Staff.NumberOfSelectedSymbols (page 269) [R_]
Staff.NumberOfSelectedSymbolsIndividually (page 269) [R_]
Staff.NumberOfText (page 266) [R_]
Staff.Pictures (page 263) [R_]
Staff.Rules (page 263) [R_]
Staff.Score (page 267) [R_]
Staff.ShapeNoteType (page 265) [RW]
Staff.ShortTitle (page 253) [RW]
Staff.ShortTitleFont (page 254) [RW]
Staff.ShortTitleRGBColor (page 254) [RW]
Staff.ShortTitleSize (page 254) [RW]
Staff.ShortTitleXOffset (page 254) [RW]
Staff.SpokenVoiceGender (page 269) [RW]
Staff.SpokenVoiceLanguage (page 269) [RW]
Staff.SpokenVoiceName (page 269) [RW]
Staff.SpokenVoicePitchOffset (page 269) [RW]
Staff.StorageArea (page 267) [R_]
Staff.SymbolDefaultColor (page 265) [RW]
Staff.Tablature (page 267) [R_]
Staff.Texts (page 266) [R_]
Staff.TitleAlignment (page 253) [RW]
Staff.TitleFace (page 253) [RW]
Staff.TitleOrigin (page 253) [RW]
Staff.TitlePrintMode (page 254) [RW]
Staff.TitleRotation (page 253) [RW]
Staff.TitleSize (page 253) [RW]
Staff.TitleYOffset (page 253) [RW]
Staff.Type (page 269) [RW]
Staff.UpperBarlineLimit (page 264) [RW]
Staff.VirtualSingerVoice (page 267) [R_]

Methods

Staff.AddChordDiagram(...) (page 279)
Staff.AddCurve(...) (page 272)
Staff.AddDynamic(...) (page 280)
Staff.AddFXProcessor(...) (page 280)
Staff.AddPicture(...) (page 287)
Staff.AddRule(...) (page 289)
Staff.AddText(...) (page 293)
Staff.AdjustLocalMetrics(...) (page 274)
Staff.BeginRecord(...) (page 288)
Staff.ClearTagOrnaments(...) (page 296)
Staff.ClearTagSymbols(...) (page 296)
Staff.CompactLinesInLyrics(...) (page 281)
Staff.ComputeNotesCaptionAlongRules(...) (page 294)
Staff.ComputePitchDisplayedHeight(...) (page 294)
Staff.CreateDefaultDrumRules(...) (page 273)
Staff.CreateLyricsArray(...) (page 285)
Staff.DeleteAllSymbols(...) (page 292)
Staff.DeleteChordDiagrams(...) (page 279)
Staff.Duplicate(...) (page 290)
Staff.DuplicateChordDiagrams(...) (page 279)
Staff.EditChordDisplay(...) (page 273)
Staff.EditChordLine(...) (page 273)
Staff.EditLyrics(...) (page 273)
Staff.EditTablature(...) (page 274)
Staff.GetChordBar(...) (page 278)
Staff.GetChordDiagram(...) (page 278)
Staff.GetChordDiagramFromName(...) (page 279)
Staff.GetChordIndexAtBar(...) (page 278)
Staff.GetChordName(...) (page 278)
Staff.GetChordTime(...) (page 278)
Staff.GetLocalBarNumberingYPosition(...) (page 276)
Staff.GetLocalBaseLine(...) (page 275)
Staff.GetLocalHeight(...) (page 274)
Staff.GetLocalLeftMargin(...) (page 276)
Staff.GetLocalLyricsYPosition(...) (page 276)
Staff.GetLocalRightMargin(...) (page 277)
Staff.GetLyricAtTime(...) (page 283)
Staff.GetLyricsAtBar(...) (page 284)
Staff.GetLyricsCellIndexAtTime(...) (page 281)
Staff.GetLyricsCellTime(...) (page 282)
Staff.HideLineInLyrics(...) (page 281)
Staff.InsertBarSetting(...) (page 271)
Staff.InsertNote(...) (page 291)
Staff.InsertRest(...) (page 292)
Staff.LoadVirtualSingerVoice(...) (page 297)
Staff.MergeWithNext(...) (page 294)
Staff.OutputLevel(...) (page 295)
Staff.OverlayNote(...) (page 291)

Staff.RemoveUnusedLyrics(...) (page 287)
Staff.SaveVirtualSingerVoice(...) (page 296)
Staff.SearchPicturesByTag(...) (page 296)
Staff.SearchSymbolsByTag(...) (page 296)
Staff.SetDefaultTablature(...) (page 296)
Staff.SetLocalBarNumberingYPosition(...) (page 276)
Staff.SetLocalChordYPosition(...) (page 275)
Staff.SetLocalHeight(...) (page 274)
Staff.SetLocalLyricLineVisibility(...) (page 276)
Staff.SetLocalLyricsYPosition(...) (page 276)
Staff.SetLocalTopMargin(...) (page 277)
Staff.SetLyricAtTime(...) (page 284)
Staff.SetLyricsCellContent(...) (page 282)
Staff.TimeToSymbol(...) (page 293)
Staff.Unmerge(...) (page 294)
Staff.VirtualSingerVoiceKind(...) (page 296)

'StavesGroup' Object type

Values

StavesGroup.AllowOnSingleStaff (page 331) [RW]
StavesGroup.HorizontalOffset (page 331) [RW]
StavesGroup.JoinMode (page 330) [RW]
StavesGroup.Next (page 328) [R_]
StavesGroup.ShortTitle (page 330) [RW]
StavesGroup.ShortTitleFace (page 330) [RW]
StavesGroup.ShortTitleRGBColor (page 330) [RW]
StavesGroup.ShortTitleRotation (page 330) [RW]
StavesGroup.ShortTitleXOffset (page 330) [RW]
StavesGroup.ShortTitleYOffset (page 330) [RW]
StavesGroup.StaffEnd (page 331) [RW]
StavesGroup.Symbol (page 330) [RW]
StavesGroup.TitleFace (page 329) [RW]
StavesGroup.TitleFont (page 329) [RW]
StavesGroup.TitleRotation (page 329) [RW]
StavesGroup.TitleSize (page 329) [RW]
StavesGroup.TitleYOffset (page 329) [RW]
StavesGroup.VerticalBottomOffset (page 331) [RW]

Methods

StavesGroup.CopyTo(...) (page 331)
StavesGroup.Delete(...) (page 331)

'BarSetting' Object type

Values

BarSetting.Bar (page 333) [R_]
BarSetting.Clef (page 332) [R_]
BarSetting.FirstOrnament (page 334) [R_]
BarSetting.IsClefVisible (page 333) [RW]
BarSetting.IsTimeVisible (page 333) [RW]
BarSetting.KeySignature (page 333) [R_]
BarSetting.Next (page 332) [R_]
BarSetting.NumberOfOrnaments (page 333) [R_]
BarSetting.Previous (page 332) [R_]

Methods

BarSetting.AddOrnament(...) (page 335)
BarSetting.CopyTo(...) (page 334)
BarSetting.DeleteAllOrnaments(...) (page 335)

'BarAction' Object type

Values

BarAction.Action (page 325) [RW]
BarAction.ActionActivation (page 326) [RW]
BarAction.Bar (page 323) [RW]
BarAction.BarlineColorIndex (page 325) [RW]
BarAction.Next (page 322) [R_]
BarAction.Previous (page 322) [R_]
BarAction.TextStyle (page 324) [R_]
BarAction.TextXOffset (page 324) [RW]
BarAction.TriggerParameter (page 326) [RW]
BarAction.TriggerType (page 326) [RW]

Methods

BarAction.CopyTo(...) (page 327)
BarAction.Delete(...) (page 326)

'Clef' Object type

Values

Clef.BaseNote (page 336) [R_]
Clef.IsDisplayStaffLines (page 336) [RW]
Clef.OctaveOffset (page 336) [RW]
Clef.Scale (page 336) [RW]

Methods

Clef.Transpose(...) (page 337)

'KeySignature' Object type

Values

KeySignature.Name (page 338) [RW]
KeySignature.RelativeDisplay (page 338) [RW]

Methods

KeySignature.GetAccidental(...) (page 339)
KeySignature.SetAccidental(...) (page 340)

'Symbol' Object type

Global functions:

FrequencyToPitch(...) (page 299)
PitchToFrequency(...) (page 299)
StringToPitch(...) (page 299)

Values

Symbol.Accidental (page 308) [RW]
Symbol.AccidentalShift (page 310) [RW]
Symbol.AccordionRow (page 305) [RW]
Symbol.AngloSaxonName (page 300) [RW]
Symbol.BeamSource (page 303) [RW]
Symbol.BeamTarget (page 303) [RW]
Symbol.ChromaticDegree (page 301) [RW]
Symbol.ChromaticOctave (page 301) [RW]
Symbol.ComputeTab (page 304) [RW]
Symbol.Delay (page 308) [RW]
Symbol.DiatonicOctave (page 301) [RW]
Symbol.DiatonicOffset (page 300) [RW]
Symbol.DisplayedHeight (page 301) [RW]
Symbol.Duration (page 307) [RW]
Symbol.FirstOrnament (page 302) [R_]
Symbol.GraceNotes (page 302) [R_]
Symbol.GraceNoteStdDisplayDuration (page 302) [RW]
Symbol.GraceNoteStdPlayDuration (page 302) [RW]
Symbol.HoleNumber (page 304) [RW]
Symbol.HorizontalPixelPosition (page 309) [R_]
Symbol.Id (page 312) [R_]
Symbol.IsAccidentalParenthesis (page 308) [RW]
Symbol.IsAccordionPush (page 305) [RW]
Symbol.IsBeamed (page 307) [RW]
Symbol.IsCautionaryAccidental (page 308) [RW]
Symbol.IsDeleted (page 307) [R_]
Symbol.IsGhostRest (page 306) [R_]
Symbol.IsGraceNoteBeamHorizontal (page 302) [RW]
Symbol.IsGraceNoteDisplayed (page 303) [RW]
Symbol.IsGraceNoteSlur (page 302) [RW]
Symbol.IsHiddenAccidental (page 308) [RW]
Symbol.IsInChord (page 307) [R_]
Symbol.IsInvertedHead (page 311) [RW]
Symbol.IsInvertedHeadDisplayed (page 311) [R_]
Symbol.IsInvisibleDot (page 311) [RW]
Symbol.IsInvisibleRest (page 312) [RW]
Symbol.IsParenthesis (page 311) [RW]
Symbol.IsRealDurationRest (page 312) [RW]
Symbol.IsRepeatTwoLastBars (page 307) [RW]
Symbol.IsRest (page 306) [RW]
Symbol.IsSelectedIndividually (page 306) [RW]
Symbol.IsSlide (page 304) [RW]
Symbol.IsTupletBracket (page 310) [RW]
Symbol.IsTupletDontDisplayBeam (page 310) [RW]
Symbol.IsTupletFull (page 310) [RW]
Symbol.IsTupletInvertedSymbol (page 310) [RW]
Symbol.LastGraceNote (page 302) [R_]
Symbol.LastOrnament (page 302) [R_]
Symbol.Name (page 300) [RW]
Symbol.Next (page 299) [R_]
Symbol.NumberOfOrnaments (page 301) [R_]
Symbol.OffsetXDot (page 311) [RW]

Symbol.Ornaments (page 301) [R_]
Symbol.Pitch (page 300) [RW]
Symbol.PlayedPitch (page 300) [RW]
Symbol.PlayedVelocity (page 306) [R_]
Symbol.Previous (page 300) [R_]
Symbol.RestVerticalOffset (page 309) [RW]
Symbol.Scale (page 311) [RW]
Symbol.Score (page 308) [R_]
Symbol.SlurSource (page 303) [RW]
Symbol.SlurTarget (page 303) [RW]
Symbol.Staff (page 308) [R_]
Symbol.StemHeight (page 309) [RW]
Symbol.TabCell (page 303) [RW]
Symbol.TabLine (page 303) [RW]
Symbol.Time (page 301) [R_]
Symbol.TupletDiv (page 310) [RW]
Symbol.TupletSymbolCurveTension (page 311) [RW]
Symbol.TupletSymbolHorizontalOffset (page 310) [RW]
Symbol.TupletSymbolVerticalSize (page 311) [RW]
Symbol.TupletValueVerticalOffset (page 311) [RW]
Symbol.VerticalPixelPosition (page 309) [R_]
Symbol.VerticalPixelSize (page 309) [R_]

Methods

Symbol.AddGraceNote(...) (page 313)
Symbol.AddOrnament(...) (page 314)
Symbol.ComputeStemDirection(...) (page 312)
Symbol.Delete(...) (page 312)
Symbol.DeleteAllOrnaments(...) (page 314)
Symbol.Draw(...) (page 312)
Symbol.GetBeamSettings(...) (page 316)
Symbol.GetRelatedRule(...) (page 316)
Symbol.GetTiedElements(...) (page 316)
Symbol.SetBeamSettings(...) (page 316)
Symbol.SetSlurSettings(...) (page 315)
Symbol.StartEcho(...) (page 313)

'Ornament' Object type

Values

Ornament.AlongStemDirection (page 342) [RW]
Ornament.Color (page 342) [RW]
Ornament.Font (page 342) [RW]
Ornament.GraphicScale (page 343) [RW]
Ornament.Index (page 341) [R_]
Ornament.IsFramed (page 343) [RW]
Ornament.IsPlayed (page 343) [RW]
Ornament.Next (page 341) [R_]
Ornament.Previous (page 341) [R_]
Ornament.Rotation (page 342) [RW]
Ornament.SlideTension (page 343) [RW]
Ornament.SlideThickness (page 343) [RW]
Ornament.Tag (page 342) [RW]
Ornament.Text (page 342) [RW]
Ornament.XOffset (page 342) [RW]
Ornament.YOffset (page 342) [RW]

Methods

Ornament.AddPicture(...) (page 344)
Ornament.AutoPosition(...) (page 348)
Ornament.Delete(...) (page 344)
Ornament.EndRecord(...) (page 346)
Ornament.SetPlayParams(...) (page 348)

'GraceNote' Object type

Values

GraceNote.Accidental (page 350) [RW]
GraceNote.Divider (page 350) [RW]
GraceNote.IsAfter (page 349) [RW]
GraceNote.IsInChord (page 350) [RW]
GraceNote.Next (page 349) [R_]
GraceNote.PitchOffset (page 349) [RW]
GraceNote.Velocity (page 349) [RW]

Methods

GraceNote.Delete(...) (page 350)

'Dynamic' Object type

Values

Dynamic.AppliedTo (page 354) [RW]
Dynamic.AreaBottom (page 353) [RW]
Dynamic.AreaLeftAbsolutePixel (page 353) [RW]
Dynamic.AreaRight (page 352) [RW]
Dynamic.AreaTop (page 352) [RW]
Dynamic.Color (page 353) [RW]
Dynamic.Index (page 352) [R_]
Dynamic.IsAppliedToMergedStaves (page 354) [RW]
Dynamic.IsCumulative (page 354) [RW]
Dynamic.IsGlobal (page 354) [RW]
Dynamic.IsPrinted (page 354) [RW]
Dynamic.IsSelected (page 352) [RW]
Dynamic.IsTempoSoftChange (page 356) [RW]
Dynamic.LeftPixelOffset (page 353) [RW]
Dynamic.Previous (page 351) [R_]
Dynamic.RightPixelOffset (page 353) [RW]
Dynamic.Scale (page 356) [RW]
Dynamic.StartValue (page 355) [RW]
Dynamic.TempoCurvePoint1 (page 356) [RW]
Dynamic.TempoCurvePoint2 (page 356) [RW]
Dynamic.TempoFont (page 355) [RW]
Dynamic.TempoNoteScale (page 356) [RW]
Dynamic.TempoSize (page 355) [RW]
Dynamic.TempoTitle (page 355) [RW]
Dynamic.Title (page 353) [RW]
Dynamic.TitleColor (page 354) [RW]
Dynamic.TitleFont (page 353) [RW]
Dynamic.TitleSize (page 354) [RW]
Dynamic.Value (page 355) [RW]

Methods

Dynamic.CopyTo(...) (page 357)
Dynamic.Delete(...) (page 356)
Dynamic.GetActivationState(...) (page 356)
Dynamic.SetActivationState(...) (page 356)

'Picture' Object type

Values

Picture.AreaBottom (page 359) [RW]
Picture.AreaLeft (page 359) [RW]
Picture.AreaRight (page 359) [RW]
Picture.AreaRightAbsolutePixel (page 359) [RW]
Picture.Color (page 360) [RW]
Picture.GraphicLineBeginKind (page 360) [RW]
Picture.GraphicLineBeginYPos (page 360) [RW]
Picture.GraphicLineBracketSize (page 360) [RW]
Picture.GraphicLineCurveWidth (page 360) [RW]
Picture.GraphicLineEndKind (page 360) [RW]
Picture.GraphicLineEndYPos (page 360) [RW]
Picture.GraphicLineKind (page 358) [RW]
Picture.HorizontalPixelSize (page 359) [R_]
Picture.HorizontalScale (page 359) [RW]
Picture.IsEraseBackground (page 361) [RW]
Picture.IsFixedScale (page 361) [RW]
Picture.IsSelected (page 358) [RW]
Picture.Next (page 358) [R_]
Picture.Rotation (page 361) [RW]
Picture.Staff (page 359) [R_]
Picture.Type (page 358) [R_]
Picture.VerticalPixelSize (page 359) [R_]

Methods

Picture.Delete(...) (page 361)
Picture.DrawPicture(...) (page 361)

'Diagram' Object type

Values

Diagram.ChordName (page 364) [RW]
Diagram.ChordNumber (page 364) [RW]
Diagram.Index (page 364) [R_]
Diagram.IsLocal (page 364) [R_]
Diagram.NumberOfFingers (page 365) [R_]
Diagram.NumberOfStrings (page 364) [R_]
Diagram.StringValue (page 365) [RW]

Methods

Diagram.AddFinger(...) (page 365)
Diagram.ClearFingers(...) (page 366)
Diagram.Duplicate(...) (page 366)
Diagram.Edit(...) (page 365)

Diagram.GetFinger(...) (page 366)
Diagram.GetFingerNumberForString(...) (page 366)

'Tablature' Object type

Values

Tablature.Accordion (page 371) [R_]
Tablature.AdditionalFont (page 370) [RW]
Tablature.AdditionalFontFace (page 370) [RW]
Tablature.AdditionalFontSize (page 370) [RW]
Tablature.BaseLine (page 369) [RW]
Tablature.CharacterColor (page 370) [RW]
Tablature.Font (page 369) [RW]
Tablature.FontColor (page 369) [RW]
Tablature.FontSize (page 369) [RW]
Tablature.Fretted (page 371) [R_]
Tablature.GraceFontColor (page 370) [RW]
Tablature.GraceFontFace (page 370) [RW]
Tablature.Harmonica (page 371) [R_]
Tablature.IsActive (page 370) [RW]
Tablature.IsBeamMode (page 368) [RW]
Tablature.IsComputed (page 371) [R_]
Tablature.IsDisplayRest (page 368) [RW]
Tablature.IsDisplaySlurs (page 368) [RW]
Tablature.IsEraseBackgroundNote (page 368) [RW]
Tablature.IsStemCentered (page 369) [RW]
Tablature.NeedComputing (page 371) [RW]
Tablature.Priority (page 371) [RW]
Tablature.ShowStandardStaff (page 368) [RW]
Tablature.StemVerticalOffset (page 369) [RW]
Tablature.Type (page 371) [RW]
Tablature.UseOttava (page 371) [RW]

Methods

Tablature.Activate(...) (page 372)
Tablature.Compute(...) (page 372)
Tablature.GetLocalBaseLine(...) (page 373)
Tablature.SearchClosestSound(...) (page 373)
Tablature.SetLocalBaseLine(...) (page 373)

'FrettedTab' Object type

Values

FrettedTab.CalculationDepth (page 374) [RW]
FrettedTab.CapodastreFret (page 374) [RW]
FrettedTab.FretChangePenalty (page 374) [RW]
FrettedTab.ImplicitTransposition (page 374) [RW]
FrettedTab.IsDisplayTAB (page 375) [RW]
FrettedTab.IsDisplayTuning (page 375) [RW]
FrettedTab.NumberOfStrings (page 375) [RW]
FrettedTab.SpecialMode (page 374) [RW]
FrettedTab.VoidStringPenalty (page 374) [RW]

Methods

FrettedTab.GetString(...) (page 375)
FrettedTab.SetString(...) (page 375)

'HarmonicaTab' Object type

Values

HarmonicaTab.BaseNote (page 376) [RW]
HarmonicaTab.DisplayMode (page 377) [RW]
HarmonicaTab.IsComputeBend (page 376) [RW]
HarmonicaTab.IsComputeOverBend (page 376) [RW]
HarmonicaTab.IsDisplayUnresolvedNotes (page 376) [RW]
HarmonicaTab.IsOptimiseBreath (page 376) [RW]
HarmonicaTab.Name (page 376) [RW]
HarmonicaTab.NumberOfHoles (page 376) [RW]
HarmonicaTab.SymbolMode (page 377) [RW]
HarmonicaTab.Type (page 376) [R_]

Methods

HarmonicaTab.GetHole(...) (page 377)
HarmonicaTab.SetHole(...) (page 378)

'AccordionTab' Object type

Values

AccordionTab.BassLineCaption (page 379) [RW]
AccordionTab.Comment (page 379) [RW]
AccordionTab.FirstRowRightHand (page 379) [RW]
AccordionTab.IsDisplayLines (page 379) [RW]
AccordionTab.LeftHandChordDuration (page 380) [RW]
AccordionTab.Name (page 379) [RW]
AccordionTab.NumberOfRowLeftHand (page 379) [RW]
AccordionTab.NumberOfRowRightHand (page 379) [RW]
AccordionTab.PushCaption (page 379) [RW]
AccordionTab.PushPullChange (page 380) [RW]

Methods

AccordionTab.GetCaptionRightHand(...) (page 381)
AccordionTab.GetPitchLeftHand(...) (page 380)
AccordionTab.SetCaptionRightHand(...) (page 381)
AccordionTab.SetPitchLeftHand(...) (page 380)

'StaffText' Object type

Values

StaffText.AreaBottom (page 384) [RW]
StaffText.AreaLeft (page 383) [RW]
StaffText.AreaRight (page 383) [RW]
StaffText.AreaRightAbsolutePixel (page 383) [RW]
StaffText.Content (page 383) [RW]
StaffText.Face (page 384) [RW]
StaffText.Index (page 382) [R_]
StaffText.IsEraseBackground (page 384) [RW]
StaffText.LeftPixelOffset (page 384) [RW]
StaffText.Next (page 382) [R_]
StaffText.RGBColor (page 384) [RW]
StaffText.RightPixelOffset (page 384) [RW]
StaffText.Size (page 384) [RW]
StaffText.Type (page 382) [RW]

Methods

StaffText.Delete(...) (page 385)
StaffText.Draw(...) (page 385)

'Chord' Object type

Values

Chord.Accidental (page 387) [R_]
Chord.BarNumber (page 388) [R_]
Chord.Fifth (page 387) [R_]
Chord.Inversion (page 388) [RW]
Chord.Ninth (page 388) [R_]
Chord.Octave (page 388) [RW]
Chord.RootNote (page 387) [R_]
Chord.Seventh (page 387) [R_]
Chord.TimePosition (page 389) [R_]
Chord.User (page 388) [RW]

Methods

Chord.CopyTo(...) (page 390)
Chord.Delete(...) (page 390)
Chord.Split(...) (page 389)

'FreeObject' Object type

Values

FreeObject.Bottom (page 392) [RW]
FreeObject.Color (page 393) [RW]
FreeObject.EmbeddedScorePage (page 393) [RW]
FreeObject.EmbeddedScoreView (page 393) [RW]
FreeObject.IsEraseBackground (page 393) [RW]
FreeObject.IsForeground (page 393) [RW]
FreeObject.Justify (page 393) [RW]
FreeObject.Left (page 392) [RW]
FreeObject.Next (page 391) [R_]
FreeObject.NoWrap (page 393) [RW]
FreeObject.OriginInPage (page 392) [RW]
FreeObject.PageNumber (page 392) [RW]
FreeObject.RelatedFileName (page 394) [RW]
FreeObject.Right (page 392) [RW]
FreeObject.State (page 392) [R_]
FreeObject.Tag (page 394) [RW]
FreeObject.TextStyle (page 393) [R_]
FreeObject.Thickness (page 393) [RW]
FreeObject.Type (page 391) [RW]
FreeObject.ViewIndex (page 394) [RW]

Methods

FreeObject.BeginRecord(...) (page 395)
FreeObject.Delete(...) (page 394)

'View' Object type

Values

View.DisplayDynamics (page 401) [RW]
View.DisplayTempo (page 401) [RW]
View.Index (page 401) [R_]
View.Name (page 401) [RW]
View.Previous (page 400) [R_]
View.SpecificConcertMode (page 401) [RW]
View.SpecificFinePosition (page 401) [RW]
View.SpecificPageSetup (page 401) [RW]
View.SpecificScoreAppearance (page 401) [RW]
View.SpecificStaffAppearance (page 401) [RW]

Methods

View.CopyTo(...) (page 402)
View.Delete(...) (page 402)

'StorageArea' Object type

Methods

StorageArea.CountItems(...) (page 396)
StorageArea.GetIndexedItemInfo(...) (page 396)
StorageArea.StoreItem(...) (page 397)

'EngraverSetup' Object type

Global functions:

CopyEngraverSetup(...) (page 403)
DefaultEngraverSetup(...) (page 403)
EditEngraverSetup(...) (page 403)
NewEngraverSetup(...) (page 403)

Values

EngraverSetup.Accidental (page 405) [RW]
EngraverSetup.Before (page 405) [RW]
EngraverSetup.EmptyBar (page 404) [RW]
EngraverSetup.Grace (page 405) [RW]
EngraverSetup.Height (page 404) [RW]
EngraverSetup.IsActive (page 404) [RW]
EngraverSetup.Lyrics (page 404) [RW]
EngraverSetup.Quarter (page 404) [RW]
EngraverSetup.SixtyFourth (page 404) [RW]
EngraverSetup.SmallestBar (page 404) [RW]
EngraverSetup.Unbeam (page 405) [RW]
EngraverSetup.Whole (page 404) [RW]

'PageSetup' Object type

Global functions:

CopyPageSetup(...) (page 406)
DefaultPageSetup(...) (page 406)
EditPageSetup(...) (page 406)
NewPageSetup(...) (page 406)

Values

PageSetup.AddChordDiagrams (page 408) [RW]
PageSetup.AddChordGrid (page 408) [RW]
PageSetup.AddStavesChordGrid (page 408) [RW]
PageSetup.BottomMargin (page 407) [RW]
PageSetup.ClefOnPreviousLine (page 409) [RW]
PageSetup.DistanceBetweenGroup (page 408) [RW]
PageSetup.EmptyStavesOnFirstGroup (page 409) [RW]
PageSetup.FirstBar (page 408) [RW]
PageSetup.InfosOnFirstPage (page 407) [RW]
PageSetup.JumpAndTargetOnAllStaves (page 409) [RW]
PageSetup.JustifyGroup (page 409) [RW]
PageSetup.JustifyGroupButNotLastPage (page 409) [RW]
PageSetup.LastBar (page 408) [RW]
PageSetup.LeftMargin (page 407) [RW]
PageSetup.MultiRestBar (page 409) [RW]
PageSetup.NumberOfMultiRestBar (page 409) [RW]
PageSetup.PrintAccompanimentsMatrix (page 408) [RW]
PageSetup.PrintChordGrid (page 408) [RW]

PageSetup.PrintRythmsMatrix (page 408) [RW]
PageSetup.RemoveEmptyGroups (page 409) [RW]
PageSetup.RepeatClef (page 409) [RW]
PageSetup.ReservedPages (page 407) [RW]
PageSetup.Scale (page 407) [RW]
PageSetup.SwapLeftRightMargin (page 407) [RW]
PageSetup.TopPageHeader (page 407) [_W]

'PageHeader' Object type

Global functions:

CopyPageHeader(...) (page 410)
DefaultPageHeader(...) (page 410)
EditPageHeader(...) (page 410)
NewPageHeader(...) (page 410)

Values

PageHeader.AlternateOddEven (page 411) [RW]
PageHeader.DontPrintOnFirstPage (page 411) [RW]
PageHeader.LeftTextString (page 411) [RW]
PageHeader.LeftTextStyle (page 411) [R_]
PageHeader.LineBelow (page 411) [RW]
PageHeader.MiddleTextString (page 412) [RW]
PageHeader.NotOnFirstPage (page 411) [RW]
PageHeader.RightMargin (page 411) [RW]
PageHeader.RightTextStyle (page 412) [R_]
PageHeader.VerticalMargin (page 412) [RW]

'Comment' Object type

Values

Comment.HorizontalOffset (page 413) [RW]
Comment.Justification (page 413) [RW]
Comment.String (page 413) [RW]
Comment.Style (page 413) [R_]

'TextStyle' Object type

Global functions:

CopyTextStyle(...) (page 414)
DefaultTextStyle(...) (page 414)
EditTextStyle(...) (page 414)
NewTextStyle(...) (page 414)

Values

TextStyle.Color (page 415) [RW]
TextStyle.Face (page 415) [RW]
TextStyle.Size (page 415) [RW]

'Rule' Object type

Values

Rule.AngloSaxonPitchCriterion (page 418) [RW]
Rule.AppliedAccidental (page 421) [RW]
Rule.AppliedColor (page 422) [RW]
Rule.AppliedDelay (page 422) [RW]
Rule.AppliedGraceNotes (page 423) [R_]
Rule.AppliedGraceNoteStdDisplayDuration (page 423) [R_]
Rule.AppliedHeadShift (page 422) [RW]
Rule.AppliedInstrument (page 420) [RW]
Rule.AppliedMinVelocity (page 420) [RW]
Rule.AppliedOrnaments (page 424) [R_]
Rule.AppliedShape (page 422) [RW]
Rule.AppliedSplitDuration (page 422) [RW]
Rule.AppliedStemType (page 423) [RW]
Rule.ColorCriterion (page 419) [RW]
Rule.FirstAppliedOrnament (page 424) [R_]
Rule.Index (page 416) [R_]
Rule.IsAccidentalShiftApplied (page 421) [RW]
Rule.IsAnyOctave (page 418) [RW]
Rule.IsAppliedGraceNoteCrossAlone (page 424) [R_]
Rule.IsAppliedGraceNoteSlur (page 424) [R_]
Rule.IsColorCriterion (page 419) [RW]
Rule.IsContinueProcessing (page 420) [RW]
Rule.IsDelayApplied (page 422) [RW]
Rule.IsDifferentiateEnharmonics (page 419) [RW]
Rule.IsHeadShiftApplied (page 422) [RW]
Rule.IsKeyRelative (page 418) [RW]

Rule.IsOrnamentCriterionNotPlayed (page 420) [RW]
Rule.IsPitchCriterion (page 418) [RW]
Rule.IsRootNote (page 420) [RW]
Rule.IsRootRelative (page 419) [RW]
Rule.IsShapeCriterion (page 417) [RW]
Rule.IsSplitDurationApplied (page 422) [RW]
Rule.IsStemTypeApplied (page 423) [RW]
Rule.IsUseful (page 424) [R_]
Rule.LastAppliedGraceNote (page 423) [R_]
Rule.LastAppliedOrnament (page 424) [R_]
Rule.MaxVelocityCriterion (page 419) [RW]
Rule.MinVelocityCriterion (page 419) [RW]
Rule.NumberOfAppliedGraceNotes (page 423) [R_]
Rule.NumberOfAppliedOrnaments (page 424) [R_]
Rule.PitchCriterion (page 418) [RW]
Rule.Previous (page 416) [R_]
Rule.Title (page 417) [RW]
Rule.TitleDisplayMode (page 417) [RW]

Methods

Rule.AddAppliedGraceNote(...) (page 426)
Rule.AddAppliedOrnament(...) (page 427)
Rule.DeleteAllAppliedGraceNotes(...) (page 426)
Rule.DeleteAllAppliedOrnaments(...) (page 427)

'Instrument' Object type

Global functions:

ClosestAvailableSoundID(...) (page 428)
GetSoundRange(...) (page 429)

Values

Instrument.Chorus (page 434) [RW]
Instrument.DigitalDrumPitch (page 436) [RW]
Instrument.DigitalSounds (page 436) [R_]
Instrument.DigitalVolume (page 436) [RW]
Instrument.FirstDigitalSound (page 436) [R_]
Instrument.FXProcessor (page 437) [R_]
Instrument.HighVelocitySecondFilter (page 436) [R_]
Instrument.Index (page 429) [R_]
Instrument.IsChorus (page 434) [RW]
Instrument.IsCustomDigitalSounds (page 436) [RW]
Instrument.IsCustomInstrument (page 431) [RW]
Instrument.IsCustomTremolo (page 437) [RW]
Instrument.IsDrum (page 431) [RW]
Instrument.IsFading (page 432) [RW]
Instrument.IsPanning (page 432) [RW]
Instrument.IsReverberation (page 434) [RW]
Instrument.IsVelocityFiltering (page 435) [RW]
Instrument.IsVirtualInstrument (page 438) [RW]
Instrument.KeyShift (page 432) [RW]
Instrument.LastDigitalSound (page 436) [R_]
Instrument.Layer1MaxOutputPower (page 433) [RW]
Instrument.Layer1MinOutputPower (page 433) [RW]
Instrument.Layer2MaxOutputPower (page 433) [RW]
Instrument.Layer2MinInputValue (page 433) [RW]
Instrument.Layer3MaxInputValue (page 433) [RW]
Instrument.Layer3MaxOutputPower (page 434) [RW]
Instrument.Layer3MinOutputPower (page 433) [RW]
Instrument.Layer4MaxOutputPower (page 434) [RW]
Instrument.Layer4MinOutputPower (page 434) [RW]
Instrument.LayerType (page 432) [RW]
Instrument.LowVelocitySecondFilter (page 435) [R_]
Instrument.MaxVelocityFiltering (page 435) [RW]
Instrument.MidiChannel (page 434) [RW]
Instrument.MidiDrumKey (page 435) [RW]
Instrument.MinVelocityFiltering (page 435) [RW]
Instrument.Name (page 430) [RW]
Instrument.NumberOfDigitalSounds (page 436) [R_]
Instrument.OutputDevice (page 431) [RW]
Instrument.Previous (page 429) [R_]
Instrument.Reverberation (page 434) [RW]
Instrument.SoundID (page 431) [RW]
Instrument.TremoloDelay (page 437) [RW]
Instrument.TremoloPeriod (page 437) [RW]
Instrument.TremoloRaiseDuration (page 437) [RW]
Instrument.VelocityOffset (page 432) [RW]
Instrument.VibratoDelay (page 437) [RW]
Instrument.VibratoPeriod (page 437) [RW]
Instrument.VibratoRaiseDuration (page 437) [RW]
Instrument.Volume (page 432) [RW]

Methods

Instrument.CopyTo(...) (page 439)
Instrument.Delete(...) (page 438)
Instrument.InsertDigitalSound(...) (page 438)
Instrument.SortDigitalSounds(...) (page 438)

'VelocityFilter' Object type

Values

VelocityFilter.CutOffHarmonic (page 440) [RW]
VelocityFilter.IsHighPass (page 440) [RW]
VelocityFilter.Resonance (page 440) [RW]

'DigitalSound' Object type

Values

DigitalSound.AttackSize (page 442) [RW]
DigitalSound.AttackSpeed (page 441) [RW]
DigitalSound.IsDecayWaitForLoop (page 442) [RW]
DigitalSound.IsInfiniteLoop (page 442) [RW]
DigitalSound.LogDecayRatio (page 442) [RW]
DigitalSound.LoopEnd (page 443) [RW]
DigitalSound.NativeDigitalData (page 443) [R_]
DigitalSound.Next (page 441) [R_]
DigitalSound.NumberOfSamples (page 443) [R_]
DigitalSound.Previous (page 441) [R_]
DigitalSound.RecordedPitch (page 442) [RW]
DigitalSound.ReleaseSpeed (page 441) [RW]
DigitalSound.SampleSize (page 442) [RW]

Methods

DigitalSound.Delete(...) (page 443)
DigitalSound.Export(...) (page 443)

'DigitalData' Object type

Values

DigitalData.EmbeddingDigitalSound (page 446) [R_]
DigitalData.NumberOfSamples (page 445) [R_]
DigitalData.SampleSize (page 445) [RW]

Methods

DigitalData.Crop(...) (page 447)
DigitalData.Delete(...) (page 446)
DigitalData.GetBlock(...) (page 447)
DigitalData.Insert(...) (page 446)

'DigitalTrack' Object type

Values

DigitalTrack.BackgroundColor (page 449) [RW]
DigitalTrack.BeginSelection (page 450) [RW]
DigitalTrack.EndSelection (page 450) [RW]
DigitalTrack.FrameColor (page 449) [RW]
DigitalTrack.IsDisplayTime (page 449) [RW]
DigitalTrack.IsFramed (page 449) [RW]
DigitalTrack.NormalizedDigitalData (page 449) [R_]
DigitalTrack.NumberOfMarkers (page 450) [R_]
DigitalTrack.Volume (page 449) [RW]

Methods

DigitalTrack.AddMarker(...) (page 450)
DigitalTrack.DeleteMarker(...) (page 450)
DigitalTrack.SetMarker(...) (page 450)

'RawTable' Object type

Global functions:

NewRawTable(...) (page 451)

Values

RawTable.AverageValue (page 452) [R_]
RawTable.ContentType (page 451) [R_]
RawTable.MinimumValue (page 452) [R_]
RawTable.Size (page 451) [R_]

Methods

RawTable.ApplyWindow(...) (page 453)
RawTable.Bound(...) (page 453)
RawTable.Dispose(...) (page 452)
RawTable.FFT(...) (page 454)
RawTable.FromBase64(...) (page 452)
RawTable.ICepstrum(...) (page 455)
RawTable.Operation(...) (page 453)
RawTable.SineSynth(...) (page 456)

'FXProcessor' Object type

Values

FXProcessor.AreaBottom (page 460) [RW]
FXProcessor.AreaLeft (page 459) [RW]
FXProcessor.AreaRight (page 460) [RW]
FXProcessor.AreaRightAbsolutePixel (page 460) [RW]
FXProcessor.Comments (page 459) [RW]
FXProcessor.DelayPower (page 467) [RW]
FXProcessor.DistortionGain (page 464) [RW]
FXProcessor.DistortionPower (page 463) [RW]
FXProcessor.DistortionType (page 463) [RW]
FXProcessor.EqualizerBass (page 465) [RW]
FXProcessor.EqualizerLimiter (page 465) [RW]
FXProcessor.EqualizerTreble (page 465) [RW]
FXProcessor.Face (page 461) [RW]
FXProcessor.FixedFadingValue (page 466) [RW]
FXProcessor.FixedResonatorFrequency (page 461) [RW]
FXProcessor.FixedResonatorGain (page 462) [RW]
FXProcessor.FlangerMaxDelay (page 464) [RW]
FXProcessor.FlangerMinDelay (page 464) [RW]
FXProcessor.FlangerPower (page 464) [RW]
FXProcessor.FlangerType (page 464) [RW]
FXProcessor.Index (page 459) [R_]
FXProcessor.IsBouncedDelay (page 467) [RW]
FXProcessor.IsBouncedReverb (page 468) [RW]
FXProcessor.IsPrinted (page 461) [RW]
FXProcessor.IsSmoothEqualizerBassTreble (page 465) [RW]
FXProcessor.IsSmoothEqualizerGain (page 465) [RW]
FXProcessor.IsSmoothFixedResonatorFrequency (page 461) [RW]
FXProcessor.IsSmoothFixedResonatorResonance (page 462) [RW]
FXProcessor.IsSmoothWahResonatorResonance (page 463) [RW]
FXProcessor.LeftPixelOffset (page 460) [RW]
FXProcessor.LFOFaderMinValue (page 466) [RW]
FXProcessor.LFOPanningDirection (page 467) [RW]
FXProcessor.LFOPanningMinValue (page 466) [RW]
FXProcessor.LFOPanningPeriod (page 467) [RW]
FXProcessor.LFOResonatorDirection (page 462) [RW]
FXProcessor.LFOResonatorGain (page 462) [RW]
FXProcessor.LFOResonatorMinFrequency (page 462) [RW]
FXProcessor.LFOResonatorPeriod (page 462) [RW]
FXProcessor.LFOResonatorStartFrequency (page 462) [RW]
FXProcessor.MultibandEQFrequencies (page 465) [R_]
FXProcessor.MultibandEQLevels (page 465) [RW]
FXProcessor.MultibandEQNumberOfBands (page 465) [R_]
FXProcessor.MultipleDelayTime (page 467) [RW]
FXProcessor.Next (page 458) [R_]
FXProcessor.OverdrivePower (page 464) [RW]
FXProcessor.OverdriveTone (page 464) [RW]
FXProcessor.Previous (page 459) [R_]
FXProcessor.ResonatorType (page 461) [RW]
FXProcessor.ReverbTime (page 467) [RW]
FXProcessor.ReverbType (page 467) [RW]
FXProcessor.RightPixelOffset (page 461) [RW]
FXProcessor.Size (page 461) [RW]
FXProcessor.WahResonatorGain (page 463) [RW]
FXProcessor.WahResonatorMaxFrequency (page 463) [RW]
FXProcessor.WahResonatorMinFrequency (page 463) [RW]
FXProcessor.WahResonatorMinVolume (page 463) [RW]

Methods

FXProcessor.Delete(...) (page 468)
FXProcessor.Edit(...) (page 468)
FXProcessor.EraseIcon(...) (page 470)
FXProcessor.GetIconPixel(...) (page 469)
FXProcessor.SelectIcon(...) (page 469)
FXProcessor.SetIconPixel(...) (page 469)

'Curve' Object type

Values

Curve.Color (page 472) [RW]
Curve.IsActive (page 472) [RW]
Curve.IsGlobal (page 472) [RW]
Curve.MaxValue (page 472) [RW]
Curve.MinValue (page 472) [RW]
Curve.Next (page 471) [R_]
Curve.Previous (page 471) [R_]
Curve.Type (page 471) [RW]

Methods

Curve.Delete(...) (page 473)
Curve.DeleteValue(...) (page 474)
Curve.GetPreviousValueIndex(...) (page 475)
Curve.GetValue(...) (page 473)

'VSVoice' Object type

Values

VSVoice.Amplification (page 479) [RW]
VSVoice.ChoirDetuning (page 480) [RW]
VSVoice.DefaultFragments (page 480) [RW]
VSVoice.DriftF0DownwardRatio (page 480) [RW]
VSVoice.DriftF0MinTime (page 480) [RW]
VSVoice.DriftF0UpwardRatio (page 480) [RW]
VSVoice.Formant2Ratio (page 477) [RW]
VSVoice.Formant3Ratio (page 478) [RW]
VSVoice.Formant5And6Ratio (page 478) [RW]
VSVoice.FragmentString (page 481) [RW]
VSVoice.GlotalSourceSaturation (page 478) [RW]
VSVoice.IsActive (page 476) [RW]
VSVoice.JitterF0Max (page 480) [RW]
VSVoice.JitterF0Speed (page 480) [RW]
VSVoice.JitterF1Speed (page 481) [RW]
VSVoice.JitterF2Maximum (page 481) [RW]
VSVoice.JitterMaximum (page 479) [RW]
VSVoice.JitterSpeed (page 479) [RW]
VSVoice.LanguageCode (page 477) [RW]
VSVoice.MinimalRestDuration (page 478) [RW]
VSVoice.PanningPlacement (page 476) [RW]
VSVoice.PitchAttack (page 480) [RW]
VSVoice.SingingFormantPower (page 478) [RW]
VSVoice.SingingFormantRiseTime (page 478) [RW]
VSVoice.TimeShift (page 478) [RW]
VSVoice.TrebleBassRatio (page 478) [RW]
VSVoice.VibratoDelayBefore (page 481) [RW]
VSVoice.VibratoFrequencyRatio (page 481) [RW]
VSVoice.VibratoPower (page 481) [RW]
VSVoice.VibratoPowerFrequencyRatio (page 481) [RW]
VSVoice.VolumeEnvelopeConsonantAttack (page 479) [RW]
VSVoice.VolumeEnvelopeConsonantEnd (page 479) [RW]
VSVoice.VolumeEnvelopeRelease (page 479) [RW]
VSVoice.VolumeEnvelopeVowelAttack (page 479) [RW]
VSVoice.VolumeEnvelopeVowelMaximum (page 479) [RW]
VSVoice.VolumeEnvelopeVowelSustain (page 479) [RW]

Methods

VSVoice.SetDefaultLanguage(...) (page 482)

'Dialog' Object type

Global functions:

Alert(...) (page 496)
Confirmation(...) (page 496)
Message(...) (page 496)
NewDialog(...) (page 496)

Values

Dialog.AreaHeight (page 498) [RW]
Dialog.AreaLeft (page 498) [RW]
Dialog.AreaWidth (page 498) [RW]
Dialog.ContextHelp (page 497) [RW]
Dialog.DialogItems (page 497) [R_]
Dialog.FirstDialogItem (page 497) [R_]
Dialog.IsFrontFloatingWindow (page 498) [RW]
Dialog.IsFrontWindow (page 498) [RW]
Dialog.MetalAppearance (page 498) [RW]
Dialog.NumberOfDialogItems (page 497) [R_]
Dialog.UserTable (page 497) [R_]

Methods

Dialog.BringToFront(...) (page 503)
Dialog.Dispose(...) (page 503)

Dialog.GetItem(...) (page 503)
Dialog.GetItemUnderMouse(...) (page 504)
Dialog.MakeTableFromItems(...) (page 504)
Dialog.NewBigStaticText(...) (page 499)
Dialog.NewColorSelector(...) (page 500)
Dialog.NewDurationSelector(...) (page 502)
Dialog.NewFrameRect(...) (page 501)
Dialog.NewHeadShapeSelector(...) (page 501)
Dialog.NewList(...) (page 500)
Dialog.NewMediumStaticText(...) (page 499)
Dialog.NewPushButton(...) (page 498)
Dialog.NewRadioButtons(...) (page 500)
Dialog.NewSlider(...) (page 502)
Dialog.NewStaticText(...) (page 499)
Dialog.NewTabView(...) (page 501)
Dialog.NewText(...) (page 501)
Dialog.SetMouseCursorShape(...) (page 505)
Dialog.Show(...) (page 503)

'DialogItem' Object type

Values

DialogItem.AreaBottom (page 507) [RW]
DialogItem.AreaHeight (page 507) [RW]
DialogItem.AreaRight (page 507) [RW]
DialogItem.AreaTop (page 507) [RW]
DialogItem.Caption (page 508) [RW]
DialogItem.ContextHelp (page 511) [RW]
DialogItem.GraphContext (page 511) [R_]
DialogItem.Group (page 511) [RW]
DialogItem.IndexInGroup (page 511) [R_]
DialogItem.IsActive (page 510) [RW]
DialogItem.IsDefault (page 510) [RW]
DialogItem.IsVisible (page 510) [RW]
DialogItem.Name (page 507) [RW]
DialogItem.Next (page 506) [R_]
DialogItem.NumberOfExtraValues (page 510) [R_]
DialogItem.Previous (page 506) [R_]
DialogItem.RelatedText (page 511) [R_]
DialogItem.Source (page 507) [RW]
DialogItem.UserTable (page 510) [R_]
DialogItem.Value (page 509) [RW]

Methods

DialogItem.Delete(...) (page 515)
DialogItem.DeleteExtraValues(...) (page 515)
DialogItem.Duplicate(...) (page 515)
DialogItem.GetExtraValue(...) (page 513)
DialogItem.SetExtraValue(...) (page 514)
DialogItem.Update(...) (page 512)

'Graph' Object type

Values

Graph.AreaBottom (page 519) [R_]
Graph.AreaLeft (page 519) [R_]
Graph.AreaTop (page 519) [R_]

Methods

Graph.BackColor(...) (page 522)
Graph.BackNormal(...) (page 520)
Graph.CharWidth(...) (page 528)
Graph.ClosePolygon(...) (page 530)
Graph.CopyRect(...) (page 523)
Graph.CurveTo(...) (page 523)
Graph.DrawPaletteButton(...) (page 533)
Graph.DrawPicture(...) (page 529)
Graph.DrawStaff(...) (page 532)
Graph.DrawString(...) (page 528)
Graph.EraseArc(...) (page 525)
Graph.EraseCurvePolygon(...) (page 531)
Graph.ErasePolygon(...) (page 530)
Graph.EraseRect(...) (page 523)
Graph.ForeColor(...) (page 522)
Graph.FrameArc(...) (page 525)
Graph.FrameOval(...) (page 524)
Graph.FramePolygon(...) (page 531)
Graph.FrameRoundRect(...) (page 524)
Graph.GetClipRect(...) (page 533)
Graph.GetFontID(...) (page 527)
Graph.GetFontName(...) (page 527)
Graph.GetPictureSize(...) (page 529)
Graph.GetPixel(...) (page 522)
Graph.InvertArc(...) (page 526)
Graph.InvertOval(...) (page 525)

Graph.InvertRect(...) (page 523)
Graph.InvertRoundRect(...) (page 524)
Graph.KillPolygon(...) (page 532)
Graph.Line(...) (page 522)
Graph.MenuBarHeight(...) (page 533)
Graph.Move(...) (page 519)
Graph.NewPictFile(...) (page 533)
Graph.OffsetPolygon(...) (page 532)
Graph.PaintArc(...) (page 526)
Graph.PaintCurvePolygon(...) (page 531)
Graph.PaintPolygon(...) (page 531)
Graph.PaintRect(...) (page 523)
Graph.PenMode(...) (page 520)
Graph.PenNormal(...) (page 520)
Graph.PenSize(...) (page 520)
Graph.RGBBackColor(...) (page 522)
Graph.RGBToComponents(...) (page 521)
Graph.RGBToIndex(...) (page 521)
Graph.SetClipRect(...) (page 533)
Graph.SetMusicalFont(...) (page 527)
Graph.StringWidth(...) (page 528)
Graph.TextFace(...) (page 527)
Graph.TextFontByName(...) (page 526)
Graph.TextMode(...) (page 528)
Graph.Update(...) (page 534)
Graph.WritePictFile(...) (page 533)

'Surface' Object type

Values

Surface.GraphContext (page 535) [R_]
Surface.Height (page 535) [R_]

Methods

Surface.AddMargins(...) (page 552)
Surface.AdjustBCS(...) (page 548)
Surface.Blur(...) (page 538)
Surface.BrightnessCurve(...) (page 549)
Surface.ComponentCurve(...) (page 550)
Surface.Crop(...) (page 552)
Surface.Distort(...) (page 558)
Surface.DropInnerShadow(...) (page 559)
Surface.Duplicate(...) (page 537)
Surface.DUSkew(...) (page 556)
Surface.Flip(...) (page 554)
Surface.GetPixel(...) (page 551)
Surface.Highlight(...) (page 547)
Surface.LoadPicture(...) (page 540)
Surface.New(...) (page 536)
Surface.OpacityBlur(...) (page 544)
Surface.OpacityFade(...) (page 543)
Surface.OpacityFromColor(...) (page 541)
Surface.OpacityInnerBlur(...) (page 545)
Surface.OpacityInvert(...) (page 546)
Surface.OpacitySet(...) (page 541)
Surface.OpacityToLuminance(...) (page 545)
Surface.Rectangle(...) (page 539)
Surface.Resize(...) (page 553)
Surface.Rotate(...) (page 555)
Surface.SavePicture(...) (page 541)
Surface.SetPixel(...) (page 552)

'TextWindow' Object type

Global functions:

FindTextWindowByTitle(...) (page 560)
NewTextWindow(...) (page 560)

Values

TextWindow.GraphContext (page 561) [R_]
TextWindow.RelatedText (page 561) [R_]
TextWindow.WindowBottomMargin (page 562) [R_]
TextWindow.WindowHeight (page 561) [RW]
TextWindow.WindowLeftMargin (page 561) [R_]
TextWindow.WindowRightMargin (page 561) [R_]
TextWindow.WindowVerticalPosition (page 561) [RW]
TextWindow.WindowWidth (page 561) [RW]

Methods

TextWindow.BringToFront(...) (page 563)
TextWindow.Dispose(...) (page 562)
TextWindow.GlobalStyle(...) (page 563)
TextWindow.Hide(...) (page 562)

TextWindow.Save(...) (page 564)
TextWindow.Show(...) (page 562)

'Palette' Object type

Values

Palette.IsOpen (page 571) [R_]
Palette.IsVertical (page 571) [R_]
Palette.Next (page 571) [R_]

Methods

Palette.Close(...) (page 573)
Palette.FirstPalette(...) (page 572)
Palette.GetPosition(...) (page 573)
Palette.GetScale(...) (page 574)
Palette.HideAll(...) (page 571)
Palette.InvertOrientation(...) (page 572)
Palette.KeyReleased(...) (page 575)
Palette.Magnetize(...) (page 573)
Palette.PseudoClick(...) (page 574)
Palette.Reset(...) (page 575)
Palette.ResetKeyboard(...) (page 575)
Palette.SendHelpMessage(...) (page 574)
Palette.SetScale(...) (page 574)
Palette.SetSize(...) (page 573)

'ProgressBar' Object type

Methods

ProgressBar.Close(...) (page 577)
ProgressBar.Maximum(...) (page 577)
ProgressBar.Status(...) (page 576)
ProgressBar.Write(...) (page 576)

'Text' Object type

Values

Text.Data (page 578) [R_]
Text.FontFace (page 579) [R_]
Text.FontSize (page 579) [R_]
Text.NumberOfCharacters (page 578) [R_]
Text.SelectedData (page 578) [R_]
Text.SelectionEnd (page 579) [R_]

Methods

Text.ChangeStyle(...) (page 581)
Text.Copy(...) (page 580)
Text.DeleteSelection(...) (page 579)
Text.GetLinePosition(...) (page 579)
Text.Insert(...) (page 579)
Text.Key(...) (page 580)
Text.SetSelect(...) (page 581)

'Jukebox' Object type

Values

Jukebox.AreaHeight (page 584) [RW]
Jukebox.AreaLeft (page 584) [RW]
Jukebox.AreaWidth (page 584) [RW]
Jukebox.CurrentEntry (page 583) [R_]
Jukebox.IsChanged (page 582) [RW]
Jukebox.IsFrontWindow (page 584) [RW]
Jukebox.IsOpen (page 582) [R_]
Jukebox.IsPlayKaraoke (page 583) [RW]
Jukebox.IsRandomPlaying (page 583) [RW]
Jukebox.NumberOfEntries (page 583) [R_]
Jukebox.TotalDuration (page 583) [R_]
Jukebox.TotalSize (page 583) [R_]

Methods

Jukebox.AddEntry(...) (page 585)
Jukebox.BuildList(...) (page 585)
Jukebox.Close(...) (page 584)
Jukebox.DeleteEntry(...) (page 585)
Jukebox.GetColumnInfo(...) (page 587)
Jukebox.GetEntryAttribute(...) (page 586)

Jukebox.GetSortingMode(...) (page 587)
Jukebox.HTMLExport(...) (page 589)
Jukebox.NextEntry(...) (page 588)
Jukebox.Open(...) (page 584)
Jukebox.Save(...) (page 585)
Jukebox.SelectEntry(...) (page 588)
Jukebox.SetEntryAttribute(...) (page 586)
Jukebox.SetSortingMode(...) (page 587)
Jukebox.StopPlaying(...) (page 588)

'File' Object type

Global functions:

AddExtension(...) (page 593)
AdjustFileName(...) (page 599)
BrowseFolder(...) (page 594)
ConvertFileExtensionToFileFormat(...) (page 594)
CreateFolder(...) (page 595)
F05062009(...) (page 603)
FileExist(...) (page 592)
FileModificationTime(...) (page 593)
GetCurrentPath(...) (page 596)
GetDocumentationPathName(...) (page 597)
GetFileNameLoad(...) (page 591)
GetFileNameSave(...) (page 592)
GetMusicFileNameLoad(...) (page 591)
GetPreferencesPathName(...) (page 597)
GetTemporaryFilesPathName(...) (page 597)
GetUserExportPathName(...) (page 598)
GetUserSettingsPathName(...) (page 598)
IsFolderExist(...) (page 595)
OpenFile(...) (page 600)
OpenFileWithApplication(...) (page 596)
OpenFolderOnDesktop(...) (page 595)
PackFile(...) (page 602)
RenameFile(...) (page 602)
SetCurrentPath(...) (page 597)
SplitName(...) (page 593)
TemporaryFileName(...) (page 602)

Values

File.CREncoding (page 604) [RW]
File.Encoding (page 603) [RW]
File.Position (page 604) [RW]

Methods

File.Close(...) (page 605)
File.Flush(...) (page 605)
File.Read(...) (page 606)
File.ReadChar(...) (page 607)
File.ReadLongMotorola(...) (page 608)
File.ReadRawTable(...) (page 608)
File.ReadShortMotorola(...) (page 607)
File.ReadSignedChar(...) (page 607)
File.ReadUnsignedLongMotorola(...) (page 608)
File.ReadUnsignedShortIntel(...) (page 607)
File.SetPos(...) (page 609)
File.WriteChar(...) (page 606)
File.WriteString(...) (page 605)
File.XMLParse(...) (page 609)

'Midi' Object type

Methods

Midi.AllNotesOff(...) (page 613)
Midi.BankChange(...) (page 612)
Midi.EchoInstrument(...) (page 610)
Midi.Expression(...) (page 613)
Midi.InBegin(...) (page 610)
Midi.InEnd(...) (page 611)
Midi.NoteOff(...) (page 612)
Midi.NoteOn(...) (page 612)
Midi.OutEnd(...) (page 611)
Midi.Panning(...) (page 613)
Midi.PitchBend(...) (page 613)
Midi.ProgramChange(...) (page 612)
Midi.Reverb(...) (page 612)
Midi.Setup(...) (page 610)

'Internet' Object type

Methods

Internet.DownloadFile(...) (page 614)
Internet.OpenWebPage(...) (page 614)

'Movie' Object type

Global functions:

CreateMovieFile(...) (page 616)
OpenMovieFile(...) (page 616)

Values

Movie.Duration (page 618) [R_]
Movie.Height (page 617) [R_]
Movie.IsText (page 617) [R_]
Movie.IsVideo (page 617) [R_]
Movie.Volume (page 617) [RW]
Movie.Width (page 617) [R_]

Methods

Movie.AddEffect(...) (page 618)
Movie.AddImage(...) (page 618)
Movie.AddVideo(...) (page 620)
Movie.AttachToDialog(...) (page 619)
Movie.ExportDigitalTrack(...) (page 621)
Movie.GetSoundCompression(...) (page 622)
Movie.ImportDigitalTrack(...) (page 620)
Movie.Pause(...) (page 619)
Movie.RemoveEffects(...) (page 620)
Movie.RemoveSounds(...) (page 620)
Movie.RemoveVideo(...) (page 620)
Movie.Resume(...) (page 619)
Movie.SetArea(...) (page 618)
Movie.ValidateChanges(...) (page 620)

5.2. Not Yet Done

5.2.1. Overview

You will find in this chapter a short description of all the objects types and features thatare scheduled to be added to MyrScript language.

5.2.2. Movie object

Implementation
· Movie object not implemented on Linux.

5.2.3. Score object

Additions to the Score (page 198) object.
· Accompaniment and rhythm array
· General effects

5.2.4. Staff object

Additions to the Staff (page 252) object.
· Chord diagrams (page 363) (at present, only partially implemented)

5.3. Possible features

5.3.1. Overview

You will find in this chapter a description of some enhancements of the MyrScript language we are planning. If, as a script developer, you need one of these features (or another one), we engage you to start a discussion thread on the dedicated Myriad Forum : http://www.myriad-online.com/cgi-bin/bbs/YaBB.pl or to contact us.

5.3.2. Custom staves

As script would be able to manage new staff types. The script would be embedded in the .mus file and its methods called when the staff is drawn or when the user acts on it.
Some limitations and problems however :

· If this kind of feature is implemented, MyrScript language (or a simplified version of it) will need to be included in the Myriad Music Plug-in too

A consequence will be a growth of the plug-in file size.

· This kind of staff will be available in Harmony only and not managed in Melody (but it is already the case for Accordion Tablatures for example).

5.3.3. Search tool

Additions to the Search tool object: Multiple search in all the object methods.