



Menu and Popup Management Plug-In for 4th Dimension

Developer Reference
Version 3.6

Automated Solutions Group
16742 Gothard Street, Suite 210
Huntington Beach, CA 92647
(800) 375-4ASG • (714) 848-0382 FAX
<http://www.asgsoft.com>

MenuPack and PopupPack written by Michael S. Erickson and Miloslav Bystricky
Manual written by Michael S. Erickson

Software License and Limited Warranty

PLEASE READ THIS LICENSE CAREFULLY BEFORE USING THE SOFTWARE CONTAINED ON THE DISK. BY USING THE SOFTWARE, YOU AGREE TO BECOME BOUND BY THE TERMS OF THIS AGREEMENT, WHICH INCLUDES THE SOFTWARE LICENSE AND WARRANTY DISCLAIMER (collectively referred to herein as the "Agreement"). THIS AGREEMENT CONSTITUTES THE COMPLETE AGREEMENT BETWEEN YOU AND AUTOMATED SOLUTIONS GROUP. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, DO NOT USE THE SOFTWARE AND PROMPTLY RETURN THE PACKAGE FOR A FULL REFUND OF THE PURCHASE PRICE, NOT INCLUDING SHIPPING OR HANDLING.

- Ownership of Software.** The enclosed manual and computer programs ("Software") were developed and are copyrighted by Automated Solutions Group ("ASG") and are licensed, not sold, to you by ASG for use under the following terms, and Automated Solutions Group reserves any rights not expressly granted to you. You own the disk(s) on which any software is recorded, but Automated Solutions Group retains ownership of all copies of the Software itself. Neither the manual nor the Software may be copied in whole or in part except as explicitly stated below.
- License.** Automated Solutions Group, as Licensor, grants to you, the LICENSE, a non-exclusive, non-transferable right to use this Software subject to the terms of the license as described below:
 - You may make backup copies of the Software for your use provided they bear the ASG copyright notice.
 - You may use this Software in an unlimited number of custom or commercial databases or applications created by the original license. No additional product license or royalty is required.
- Restrictions.** You may not distribute copies of the Software to others (except as an integral part of a database or application within the terms of this License) or electronically transfer the Software from one computer to another over a network. You may not distribute copies of the Software as an integral part of a development shell.
- Termination.** This license is effective until terminated. This license will terminate immediately without notice from ASG if you fail to comply with any of its provisions. Upon termination you must destroy the Software and all copies thereof, and you may terminate this license at any time by doing so.
- Update Policy.** ASG may create, from time to time, updated versions of the Software. At its option, ASG will make such updates available to the Licensee.
- Warranty Disclaimer.** THE SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. ASG DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIALS IN THE TERMS OF CORRECTIONS, ACCURACY, RELIABILITY, CURRENTNESS OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. IF THE SOFTWARE OR WRITTEN MATERIALS ARE DEFECTIVE YOU, AND NOT ASG OR IT'S DEALERS, DISTRIBUTORS, AGENTS, OR EMPLOYEES, ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. However, ASG warrants to the original Licensee that the disk(s) on which the Software is recorded is free from defects in materials and workmanship under normal use and service for a period of thirty (30) days from the date of delivery as evidenced by a copy of the receipt.

THIS IS THE ONLY WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, THAT IS MADE BY ASG ON THIS SOFTWARE PRODUCT. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY ASG, IT'S DEALERS, DISTRIBUTORS, AGENTS, OR EMPLOYEES SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY, AND YOU MAY NOT RELY ON SUCH INFORMATION OR ADVICE. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHER RIGHTS, WHICH VARY FROM STATE TO STATE.
- Governing Law.** This agreement shall be governed by the laws of the State of California.

Copyright and Trademarks

All trade names referenced in this document are the trademark or registered trademark of their respective holder.

MenuPack is copyright Automated Solutions Group.

4th Dimension, ACI, ACI US, and 4D Compiler are registered trademarks and 4D, 4D Server, 4D Client, and 4D Insider are trademarks of ACI/ACI US, Inc.

Windows is a trademark of Microsoft Corporation.

Macintosh, MacOS, and ResEdit are trademarks of Apple Computer, Inc.

[This page left blank intentionally]

Table of Contents

Software License and Limited Warranty	ii
Copyrights and Trademarks.....	iii
Table of Contents.....	v
Preface	viii
About this Manual	ix
Acknowledgments.....	ix

Chapter 1:	Introduction	
	MenuPack Features	12
	What's New in MenuPack 3.6.....	12
	System and Hardware Requirements.....	13
	Product Support	14
Chapter 2:	Installation	
	Plug-In Objects — Overview	16
	MenuPack Plug-In	16
	PopupPack Plug-In.....	16
	Installation: Plug-In – Macintosh	16
	Installation: Plug-In – Windows.....	17
Chapter 3:	Tutorial	
	MenuPack	19
	Installing a Hierarchical Menu	15
	Handling Custom Menu Selection	19
	PopupPack	15
	Creating a Basic Popup Menu Control.....	21
	Creating a Hierarchical Popup Menu Control.....	23
	Using PopupPack Popup Control with Arealist™ Pro	24
	Using PopupPack Offscreen Popup Control with ALP™	26
Chapter 4:	Command Reference	
	MenuPack Routines	
	MP_NewMenu	28
	MP_ArrayMenu.....	29
	MP_TextMenu	31
	MP_NewResMenu.....	33
	MP_GetMenu	34
	MP_UpdateMenu.....	36
	MP_InsertMenu	37
	MP_PowerMenu	38
	MP_CoutMItems.....	41
	MP_DeleteMenu.....	42
	MP_DeleteMItem.....	43
	MP_GetMenuItem	44
	MP_GetMenuTitle	45
	MP_SetItemMark.....	46
	MP_SetItemText	47
	MP_SetItemIcon	48
	MP_EnableItem	49
	MP_DisableMenu	50

MP_EnableMenu.....	51
MP_HideMenu.....	52
MP_ShowMenu.....	53
MP_DrawMBar.....	54
MP_GetMenuVers.....	55
MP_Register.....	56

PopupPack Routines	
%PopupArea.....	57
MP_NewArea.....	58
MP_DestroyArea	59
MP_ArrayPop.....	60
MP_TextPop	61
MP_NewGlobal.....	62
MP_InsertGlobal.....	63
MP_DeletePop	64
MP_DeleteGlobal	65
MP_GlobalList	66
MP_EnbPopItem	67
MP_GetPopItem.....	68
MP_SetPopText	69
MP_SetPopMark	70
MP_SetPopItem	71
MP_SetPopIcon.....	72
MP_SetPopPICT	73
MP_GetPopMark.....	74
MP_AreaMenuID.....	75
MP_HideArea	76
MP_ShowArea	77
MP_DisablePop.....	78
MP_EnablePop	79
MP_DrawPop	80
MP_SetPopOpts	81
MP_GetPopOpts.....	83
MP_SetPopFont	84
MP_GetPopFont	85
MP_GetPopVers.....	86
MP_RegisterPop	87

Chapter 5: **MenuPack Constants and Error Codes**

MenuPack Constants	89
MenuPack Error Codes.....	89

Appendix

Technical Support	90
-------------------------	----

Index

Command Index Plug-In Routines	
MenuPack Routines - Alphabetical.....	92
PopupPack Routines - Alphabetical.....	93

Preface

MenuPack is an external package that provides the ability to create custom menus and popups, including complete hierarchical menu support.

Using MenuPack, you can create enhanced menus and popups which are not possible with standard 4th Dimension menus and popup controls.

About this Manual

Throughout this manual, you will see various formatting options to better distinguish the use of external routines, 4th Dimension commands, and 4th Dimension procedures/functions. Listed below is a sample of each reference used.

4D Commands and Functions	SEARCH
4D Procedures	<i>mpSTARTUP</i>
4D External Procedures	<i>MP_ArrayPop</i>
Code Examples:	iErr= <i>MP_ArrayPop</i> (0;0;"myMenu";atItems;iMenu)

Acknowledgments

Design and Programming by **Michael S. Erickson**

Published and distributed by Automated Solutions Group

Telephone	(800) 375-4ASG
Tech Support	(714) 375-4257 or support@asgsoft.com
Facsimile	(714) 848-0382
Internet	http://www.asgsoft.com sales@asgsoft.com

by mail	16742 Gothard Street, Suite 210 Huntington Beach, CA 92647 USA
---------	---

This manual may not, whole or in part, be copied, photocopied, reproduced, translated, or converted to any electronic or machine-readable form without prior written consent of Automated Solutions Group.

MenuPack and PopupPack are trademarks of Automated Solutions Group.
All other trade names are the trademark or registered trademark of their respective holders.

1 — Introduction

MenuPack is a plug-in for 4th Dimension that provides the ability to create and manage custom menus and popup menus for using in Custom application.

MenuPack Features

Included in MenuPack™ are the following features:

- Procedurally create and manage custom menus
- Create hierarchical menus (supports up to 5 sub-levels)
- Enhanced menu management routines
- Display Contextual menus, using Power Menu routine
- Procedurally create and manage custom popup menus, including advanced formatting options
- Create hierarchical popup menus (supports up to 5 sub-levels).
- Create menus and popup menus based on arrays, delimited text, resources or MBAR resources

What's New in MenuPack 3.6

MenuPack 3.6 offers a number of new features, including:

- **Support for 4th Dimension v6.8**

MenuPack 3.6 is fully compatible with 4th Dimension v6.8 including 4D Server v6.8.

- **Support for MacOS X and Windows XP**

MenuPack 3.6 is now compatible MacOS X and Windows XP.

- **AreaList™ Pro Integration Support**

MenuPack 3.6 now provides direct integration with AreaList™ Pro. When creating popup menus (using PopupPack) you can use these menu references within an ALP popup menu – including full support for hierarchical submenus.

NOTE: Requires AreaList 7.7.1 or greater.

System Requirements

4th Dimension

Macintosh

MenuPack is compatible with 4th Dimension v6.x and higher, or 4D Server 6.x or higher.

Windows

MenuPack is compatible with 4th Dimension v6.x and higher, or 4D Server 6.x or higher.

4D Compiler

MenuPack is compatible with compiled applications, using 4D Compiler 6.x and higher.

System Software

Macintosh and Windows

MenuPack is compatible with any version of system software that is capable of running versions of 4th Dimension outlined above.

Product Support

Technical Support

Technical support for MenuPack is provided free of charge to all registered users. Please refer to the Appendix for complete information on contacting Automated Solutions Group's technical support department.

Product Updates

In the event updates are created for MenuPack, they will be made available to all registered users. All updates will be made available our web site or via mail for a nominal shipping charge.

Registration

In order to receive technical support and notification of updates, please complete your registration card. Refer to the Appendix for information on where to send your completed registration card.

Each MenuPack disk contains a unique registration number, which is required, the first time you launch the MenuPack Installer. Please make sure you keep this registration number handy, as it will be required for technical support or updates.

NOTE: Make sure you register each component bundled with MenuPack so that you're notified of future upgrades of each independent component. For a complete list of bundled components, please refer to the Installation instructions.

2 — Installation

This chapter outlines the steps necessary for installing MenuPack and PopupPack plug-in's into your existing applications. When installing MenuPack or PopupPakc, you must have access to the source code version (4th Dimension plug-in's cannot be installed into compiled applications).

Plug-In Objects — Overview

MenuPack and PopupPack consist of a standard 4th Dimension plug-in object and is compatible with Macintosh and Windows.

MenuPack Plug-In

The MenuPack plug-in includes all the routines for creating and managing custom menus within a 4th Dimension application.

PopupPack Plug-In

The PopupPack plug-in includes all the routines for creating and managing custom popup menus within a 4th Dimension application.

Installation: Plug-In 4D 2003

After you have completed the installation of MenuPack and PopupPack for Macintosh using the MenuPack Installer application, perform the following tasks:

Step 1: Locate **MenuPack™** plug-in located in the **MenuPack 3.6** folder created by the MenuPack Installer.

NOTE: The MenuPack 3.6 Installer folder will be located in a new folder labeled Automated Solutions Group, which will contain all tools provided by Automated Solutions Group and created automatically by the installer.

Step 2: Locate the 4th Dimension structure you wish to install the MenuPack plug-in package.

Step 3: If you don't already have a directory labeled "Mac4DX", create one now.

Step 4: Copy the MenuPack™ plug-in to your applications Mac4DX folder.

NOTE: If you have an existing version of MenuPack already installed in the Mac4DX directory, please make sure it is removed before launching your application.

Step 5: Repeat Step 1 through 4 if you wish to install the PopupPack plug-in.

Installation: Plug-In 4D v11

Depending on the method in which you obtained Toolchest, the physical location of the Toolchest plug-in will vary. In most cases, the plug-in will be located in a folder labeled **Toolchest**, located in an **Automated Solutions Group** folder.

New 4D v11 Databases

If you are installing into a database that was created with 4D v11 (not converted from a previous version of 4D, you will have a single bundled project (.4dbase).

The following instructions work under the pretense that you are installing into a 4D v11 formatted database (form information on installing into converted 4D 2003/2004 databases, please refer to **Converted 4D Databases** below)

Step 1: Locate your database (.4dbase)

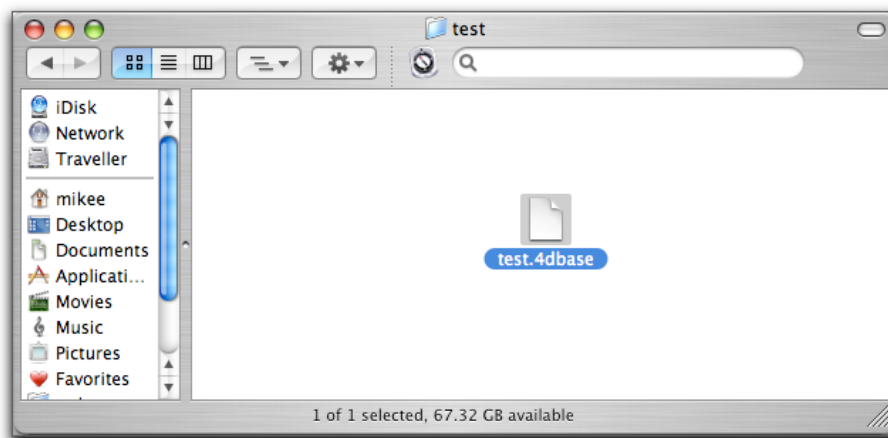


Figure 4 – 4D v11 Package

Step 2: Control-Click or Right Mouse Click on the icon to display the file properties

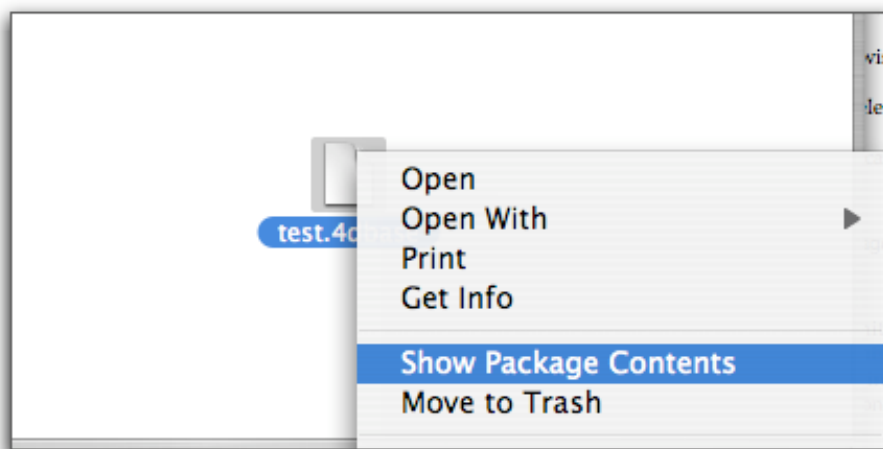


Figure 5 – Mac OS X Package

Step 3: Select Show Package Contents

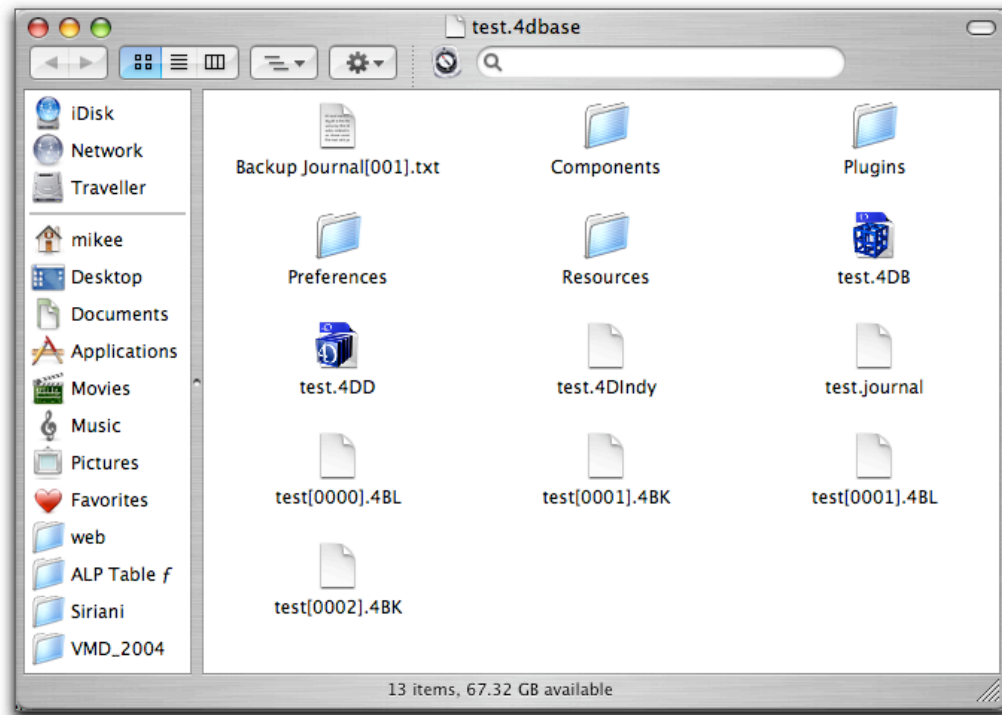


Figure 6 – 4D v11 Package Contents

Step 4: Open Plugins folder

Step 5: Drag **MenuPack.bundle** and/or **PopupPack.bundle** into the Plugins folder

Installation: Plug-In — Windows

After you have completed the installation of MenuPack for Windows using the MenuPack 3.6 Setup application (MENUPACK32.EXE), perform the following tasks:

- Step 1:** Locate the "Win4DX" folder located in the MenuPack30 directory created by the MenuPack Setup application.
- Step 2:** You will see two files that are the Windows versions of the MenuPack plug-in package.
- MENUPACK.4DX
 - MENUPACK.RSR
- Step 3:** Locate the 4th Dimension structure you wish to install the MenuPack plug-in packages.
- Step 4:** If you don't already have a directory labeled "Win4DX", create one now.
- Step 5:** Copy the MenuPack plug-in and resource file to your applications Win4DX directory.
- Step 6:** Repeat Step 1 through 5 if you wish to install the PopupPack plug-in
- POPUPPACK.4DX
 - POPUPPACK.RSR

3 — Tutorial

This chapter provides a number of examples of ways in which you can use MenuPack and PopupPack within your applications. As with all 4th Dimension plug-ins, there are a variety of ways in which these tools can be used — we will cover some of the most commonly used methods of these plug-in's.

MenuPack

MenuPack provides a suite of routines which enables you to create and manage custom menu (those which exist in the application menu bar) and extend the default menu management routines included with the standard 4th Dimension package.

Installing a Hierarchical Menu

Perhaps the most widely used feature of MenuPack is its ability to add hierarchical menus (submenus) to an existing 4th Dimension menu. After you have created your default 4th Dimension menu structure (for more information on creating menus in 4th Dimension, please refer to the Design Reference).

Step 1: Open the form method of the form that you are displaying where you wish to add a custom hierarchical menu.

Step 2: In the forms **On Activated** event, add the following code.

NOTE: Make sure that you have enabled the On Activated event for this form in the Form Properties.

```
MENU BAR(1) `reset the menu bar
iFileSubMenuID:=220 `define the default menuID
$Err:=MP_DeleteMenu(iFileSubMenuID) `delete any existing menuID
$Err:=MP_TextMenu (1;1;"";"Item1;Item2";0; iFileSubMenuID) `create custom menu
MP_DrawMBar `update the menu bar
```

In addition to using the **MP_TextMenu** routine, you can also use any of the other menu creation routines (such as **MP_ArrayPop**) available in MenuPack to create your custom menus.

Handling Custom Menu Selection

Using standard 4th Dimension menus, all menu handler code are defined in the 4th Dimension Menu Editor, thus requiring no additional menu handler code.

With MenuPack, since all menus are created procedurally at runtime, all menu handling code must also be managed procedurally. Fortunately, 4th Dimension has provided an interface whereby developers can hook into the menu selection event and perform the necessary menu handling code.

Step 1: Open the form method of the form that you are displaying where you wish to add a custom menu handler code.

Step 2: In the forms **On Menu Selected** event, add the following code.

NOTE: Make sure that you have enabled the On Menu Selected event for this form in the Form Properties.

```
$MenuID:=Menu selected\65536
$itemID:=Menu selected%65536
If ($menuID=iFileSubMenuID) `make sure we have selected one of our custom menus
    $MenuStr:=MP_GetMenuItem ($MenuID;$ ItemID)
    ALERT($MenuStr)
End if
```

NOTE: When obtaining menu information for a custom menu, you must use the appropriate MenuPack routine to get (and set) the desired menu information, as 4th Dimension is not familiar with custom menus created by MenuPack, thus does not have the necessary menu information.

PopupPack

PopupPack provides a plug-in area and a suite of routines which enables you to create and manage custom popup menus (those which exist on your forms) and extend the default popup controls included with the standard 4th Dimension package.

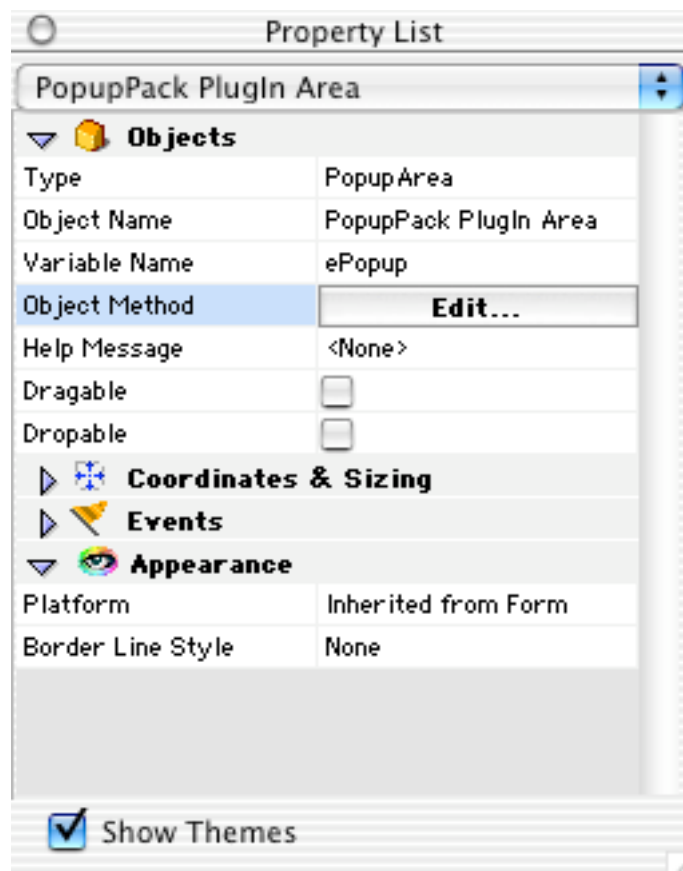
Creating a Basic Popup Menu Control

PopupPack provides a myriad of methods for creating and displaying custom popup controls. Unlike standard 4D popup menus which are typically based on 4th Dimension arrays, PopupPack controls can be created based on delimited text strings, arrays, MENU resources, etc.

The following example will demonstrate how you can create popup controls using either delimited text values (MP_TextPop) or arrays (MP_ArrayPop).

Step 1: The first and most important step is to place the PopupPack control on your form where you wish to display the popup menu.

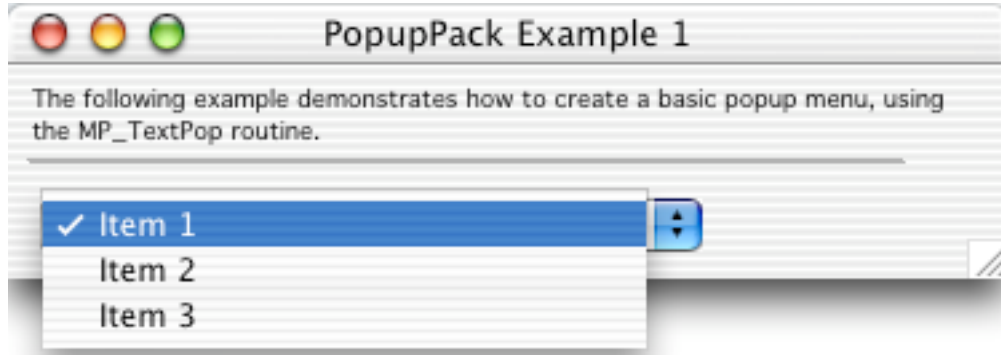
When creating the control reference, assign it a variable name, which will be used by subsequent PopupPack routines to create and interface with user selection.



Step 2: After you have created the form object, you can use the PopupPack routines to create and populate the form control with the desired popup elements.

```
iMenuID:=200
```

```
$ret:=MP_TextPop (ePopup;0;0;""; "Item 1;Item 2;Item 3";iMenuID)
```



Creating a Hierarchical Popup Menu Control

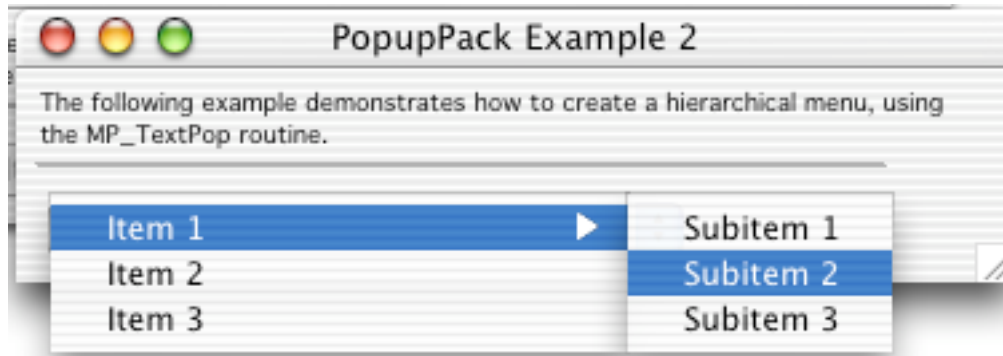
While this above is not exhaustive and does not differ much from a standard 4th Dimension popup menu, the real power of PopupPack is its further extensibility to customize the appearance and interaction of popup controls.

The next example will build on the popup control created in the Basic Popup Control example and will demonstrate how you can use PopupPack to create hierarchical popup menus.

Step 1: After you have created the form object, you can use the PopupPack routines to create and populate the form control with the desired popup elements.

```
iMenuID:=200  
iSubmenuID_1:=128
```

```
$ret:=MP_TextPop (ePopup;0;0;""; "Parent 1;Parent 2;Parent 3";iMenuID)  
$ret:=MP_TextPop (ePopup;iMenuID;1;""; "Subitem 1;Subitem 2";iSubmenuID_1)
```



Using PopupPack Popup Control with AreaList™ Pro

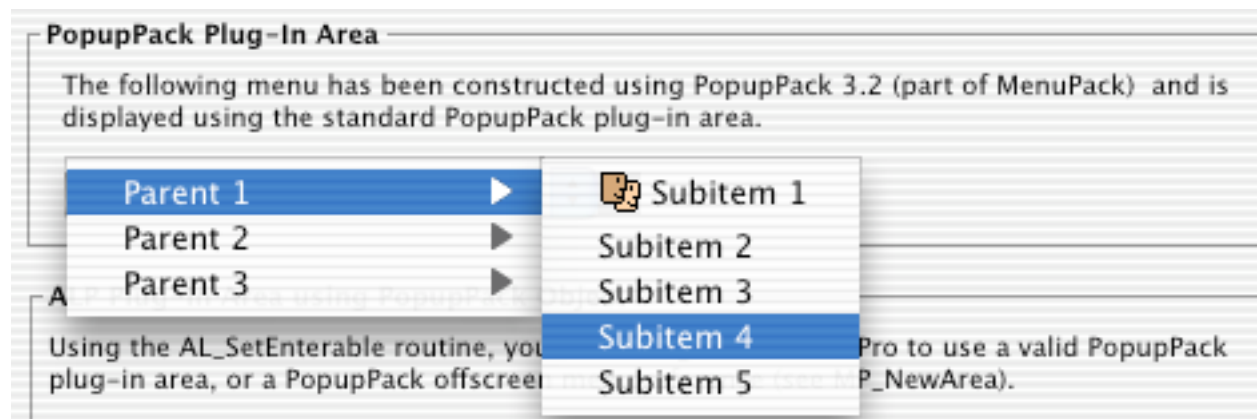
In addition to the standard popup controls that you can place on 4D forms, you can also use popup controls with AreaList Pro as popup controls for individual cells (requires ALP 7.7.2 or greater).

NOTE: The screen shots used in this example show additional information (ie icons) but the basic constructs used to integrate PopupPack with AreaList Pro are the same.

Step 1: The first step to using PopupPack with AreaList Pro is to create a valid PopupPack popup control, using the plug-in area object (ePopup). This example will use the `MP_TextPop` routine, however, you can use any valid means of creating a Popup control (such as `MP_ArrayPop`)

```
iMenuID:=200  
iSubmenuID_1:=128
```

```
$ret:=MP_TextPop (ePopup;0;0;""; "Parent 1;Parent 2;Parent 3";iMenuID)  
$ret:=MP_TextPop (ePopup;iMenuID;1;""; "Subitem 1;Subitem 2";iSubmenuID_1)
```

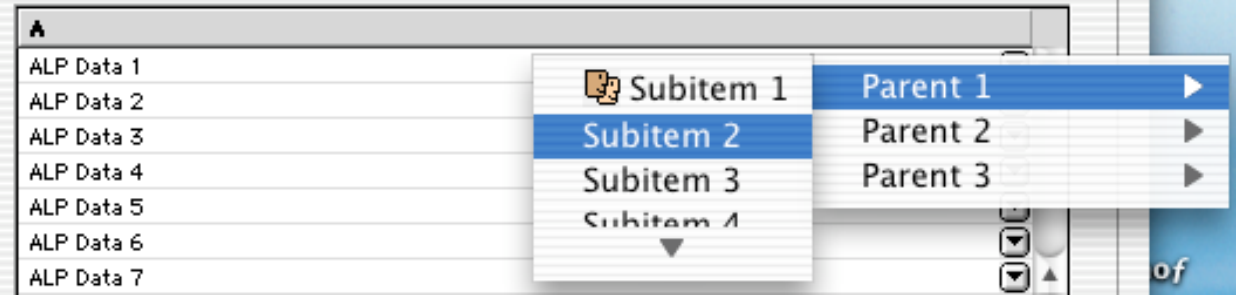


Step 2: After you have created the popup control reference, the next step will be to instruct AreaList Pro to use the popup control within the list. This is accomplished using the `AL_SetEnterable` command (see AreaList Pro Developer Reference for more information).

```
AL_SetEnterable (eALP;3;2;aALPDummy;ePopup)
```

ALP Plug-In Area using PopupPack Object

Using the `AL_SetEnterable` routine, you can configure AreaList Pro to use a valid PopupPack plug-in area, or a PopupPack offscreen menu reference (see `MP_NewArea`).



Using PopupPack Offscreen Popup Control with AreaList Pro™

In the event you don't have a popup control on the form, which contains the AreaList Pro control, you can still use PopupPack menus using the `MP_NewArea` routine which creates an "offscreen" menu reference.

Step 1: The first step to using PopupPack offscreen references with AreaList Pro is to create a valid PopupPack menu reference using the `MP_NewArea` routine.

```
iMenuRef:=MP_NewArea
```

Step 2: After you have created the menu reference, you can populate the offscreen menu using the standard PopupPack configuration routines (such as `MP_TextPop` or `MP_ArrayPop`).

```
ARRAY TEXT(atParent;3)
```

```
atParent{1}:= "Parent 1 "
```

```
atParent{2}:= "Parent 2 "
```

```
atParent{3}:= "Parent 3 "
```

```
iMenuID:=200
```

```
iSubmenuID:=128
```

```
$ret:=MP_ArrayPop (iMenuRef;0;0;""; atParent;0;iMenuID)
```

```
$ret:=MP_TextPop (iMenuRef;iMenuID;1;""; "Subitem 1;Subitem 2";iSubmenuID_1)
```

Step 3: After you have created the offscreen popup menu reference, the next step will be to instruct AreaList Pro to use the popup control within the list. This is accomplished using the `AL_SetEnterable` command (see AreaList Pro Developer Reference for more information).

```
AL_SetEnterable (eALP;3;2;aALPDummy;$menuRef)
```

Step 4: Upon completion of the offscreen menu reference, you must properly destroy the menu referencing using the `MP_DestroyArea` routine.

```
$ret:=MP_DestroyArea(iMenuRef)
```

4 — Command Reference

This chapter outlines all the external routines included in the MenuPack package as well as examples on how to use each routine. The routines in this chapter are grouped by category, corresponding to the categories displayed in the 4th Dimension procedure editor. Please refer to the **Command Index** for an alphabetical listing of each MenuPack and PopupPack routine.

The syntax used here is similar to the syntax used by the 4th Dimension Language Reference manual.

MenuPack contains a set of routines for creating and managing custom menus and popups. Each routine has been placed in a specific section for easy identification. These categories include:

- **MenuPack Routines**
These routines are designed to create and manage custom menus.
- **PopupPack Routines**
These routines are designed to create and manage custom popup menus.

MenuPack Routines

The following routines allow you create and manage custom menus, including complete hierarchical menu support.

MP_NewMenu

MP_NewMenu(menuTitle:S; menuItemArray:X; menuOptions:L; newMenuID:L) -> retCode:L

Parameter	Type
menuTitle	C_STRING or C_TEXT
menuItemArray	ARRAY TEXT or ARRAY STRING
menuOptions	C_LONGINT
newMenuID	C_LONGINT
-> retCode	C_LONGINT

MP_NewMenu creates a new global menu which can be inserted as a parent menu (in the main menu bar) or as a submenu to an existing menu item using the **MP_InsertMenu** routine. This routine is similar to the **MP_ArrayMenu** routine but it does not get inserted into the current menu bar until **MP_InsertMenu** is called.

menuTitle — Menu title (ignored if inserted as a submenu).

menuItemArray — A valid 4th Dimension array containing the corresponding menu items.

menuOptions — MenuPack formatting options.

0 - No options, default formatting

1 - Meta Character override. Menu meta characters will not be disabled.

newMenuID — A valid 4th DIMESNION longint variable containing the desired menuID. If the menuID already exists, a replacement menuID will be returned.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will create a global menu, then insert it into the current menu bar and then end of the menu list.

```
C_LONGINT(iMenuID)
```

```
iMenuID:=200
```

```
ARRAY TEXT(atItems;3)
```

```
atItems{1}:="Item 1"
```

```
atItems{2}:="Item 2"
```

```
atItems{3}:="Item 3"  
iErr=MP_NewMenu("myMenu";atItems;0;iMenuID)  
iErr=MP_InsertMenu(0;0;iMenuID)
```

MP_ArrayMenu

MP_ArrayMenu(menuID:L; itemID:L; menuTitle:S; menuItemArray:X; menuOption:L; newMenuID:L) -> retCode:L

Parameter	Type
menuID	C_LONGINT
itemID	C_LONGINT
menuTitle	C_STRING or C_TEXT
menuItemArray	ARRAY TEXT or ARRAY STRING
menuOptions	C_LONGINT
newMenuID	C_LONGINT
-> retCode	C_LONGINT

MP_ArrayMenu creates a new menu (in menu bar) or submenu (attached to an existing menu) using a 4th Dimension array as the menu item list.

menuID — Desired menuID where you would like to Insert the menu. You can use any of the following values:

- 0 - Inserts the menu at the end of the current menu list
- >1 -Inserts the menu after the desired 4D menu index (see example)
- menuID -A valid MenuPack menuID, created using MP_InsertMenu, MP_ArrayMenu, etc.

menuItem — Desired menuItem index where you would like menu inserted. You can either insert the menu as a standard menu list, or as a submenu to an existing menu.

- 0 - Insert as stand alone menu
- >0 -Insert as a submenu to desired menuItem.

menuTitle — Menu title (ignored if inserted as a submenu).

menuItemArray — A valid 4th Dimension array containing the corresponding menu items.

menuOptions — MenuPack formatting options.

- 0 - No options, default formatting
- 1 - Meta Character override. Menu meta characters will not be disabled.

newMenuID — A valid 4th DIMESNION longint variable containing the desired menuID. If the menuID already exists, a replacement menuID will be returned.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will demonstrate how to create a custom menu using a 4th Dimension array as the item list.

```
C_LONGINT(iMenuID)
```

```
iMenuID:=200
```

```
ARRAY TEXT(atItems;3)
```

```
atItems{1}:="Item 1"
```

```
atItems{2}:="Item 2"
```

```
atItems{3}:="Item 3"
```

The following example will insert the menu at the end of the current menu bar

```
iErr:=MP_ArrayMenu(0;0;"myMenu";atItems;0;iMenuID)
```

The following example will insert the menu after the first 4D menu.

```
iErr:=MP_ArrayMenu(1;0;"myMenu";atItems;0;iMenuID)
```

The following example will insert the menu as a submenu to the first 4D menu, second menuitem.

```
iErr:=MP_ArrayMenu(1;2;"myMenu";atItems;0;iMenuID)
```

MP_TextMenu

MP_ArrayMenu(menuID:L; itemID:L; menuTitle:S; menuItemList:T; menuOption:L; newMenuID:L) -> retCode:L

Parameter	Type
menuID	C_LONGINT
itemID	C_LONGINT
menuTitle	C_STRING or C_TEXT
menuItemList	C_STRING or C_TEXT
menuOptions	C_LONGINT
newMenuID	C_LONGINT
-> retCode	C_LONGINT

MP_TextMenu creates a new menu (in menu bar) or submenu (attached to an existing menu) using a delimited text string as the menu item list.

menuID — Desired menuID where you would like to Insert the menu. You can use any of the following values:

- 0 - Inserts the menu at the end of the current menu list
- >1 - Inserts the menu after the desired 4D menu index (see example)
- menuID - A valid MenuPack menuID, created using MP_InsertMenu, MP_ArrayMenu, etc.

menuItem — Desired menuItem index where you would like menu inserted. You can either insert the menu as a standard menu list, or as a submenu to an existing menu.

- 0 - Insert as stand alone menu
- >0 - Insert as a submenu to desired menuItem.

menuTitle — Menu title (ignored if inserted as a submenu).

menuItemList — A delimited text object containing the desired menu items. Each menu item must be separated by a semi-color (;).

menuOptions — MenuPack formatting options.

- 0 - No options, default formatting
- 1 - Meta Character override. Menu meta characters will not be disabled.

newMenuID — A valid 4th DIMENSION longint variable containing the desired menuID. If the menuID already exists, a replacement menuID will be returned.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will demonstrate how to create a custom menu using a 4th Dimension text string as the item list.

```
C_LONGINT(iMenuID)  
C_TEXT(tMenuItems)
```

```
iMenuID:=200  
tMenuItems:="Item 1;Item 2;Item 3"
```

The following example will insert the menu at the end of the current menu bar

```
iErr:=MP_TextMenu(0;0;"myMenu";tMenuItems;0;iMenuID)
```

The following example will insert the menu after the first 4D menu.

```
iErr:=MP_TextMenu(1;0;"myMenu";tMenuItems;0;iMenuID)
```

The following example will insert the menu as a submenu to the first 4D menu, second menuitem.

```
iErr:=MP_TextMenu(1;2;"myMenu";tMenuItems;0;iMenuID)
```

MP_NewResMenu

MP_NewResMenu(menuID:L; itemID:L; menuTitle:S; resName:S; newMenuID:L) -> retCode:L

MP_NewResMenu creates a new menu (in menu bar) or submenu (attached to an existing menu) using a standard Macintosh resource. This routine is typically used to create a menu containing a list of fonts or desk accessories.

Parameter	Type
menuID	C_LONGINT
itemID	C_LONGINT
menuTitle	C_STRING or C_TEXT
resName	C_STRING
newMenuID	C_LONGINT
-> retCode	C_LONGINT

menuID — Desired menuID where you would like to Insert the menu. You can use any of the following values:

- 0 - Inserts the menu at the end of the current menu list
- >1 - Inserts the menu after the desired 4D menu index (see example)
- menuID - A valid MenuPack menuID, created using MP_InsertMenu, MP_ArrayMenu, etc.

menuItem — Desired menuItem index where you would like menu inserted. You can either insert the menu as a standard menu list, or as a submenu to an existing menu.

- 0 - Insert as stand alone menu
- >0 - Insert as a submenu to desired menuItem.

menuTitle — Menu title (ignored if inserted as a submenu).

resName — A valid Macintosh resourceType which will be used to create a list of menuItems.

newMenuID — A valid 4th DIMESNION longint variable containing the desired menuID. If the menuID already exists, a replacement menuID will be returned.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will create a custom menu, using the currently installed fonts as the menu items.

```
C_LONGINT(iMenuID)
```

```
iMenuID:=200
```

```
iErr=MP_NewResMenu(0;0;"Fonts";"FONT";iMenuID)
```

MP_GetMenu

MP_GetMenu(menuID:L; itemID:L; menuResID:L) -> retCode:L

Parameter	Type
menuID	C_LONGINT
itemID	C_LONGINT
menuResID	C_LONGINT
-> retCode	C_LONGINT

MP_GetMenu creates a new menu (in menu bar) or submenu (attached to an existing menu) by loading a valid **MENU** resource. The menu's title and menuItems will be based on the information in the **MENU** resource.

menuID — Desired menuID where you would like to Insert the menu. You can use any of the following values:

- 0 - Inserts the menu at the end of the current menu list
- >1 -Inserts the menu after the desired 4D menu index (see example)
- menuID -A valid MenuPack menuID, created using MP_InsertMenu, MP_ArrayMenu, etc.

menuItem — Desired menuItem index where you would like menu inserted. You can either insert the menu as a standard menu list, or as a submenu to an existing menu.

- 0 - Insert as stand alone menu
- >0 -Insert as a submenu to desired menuItem.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

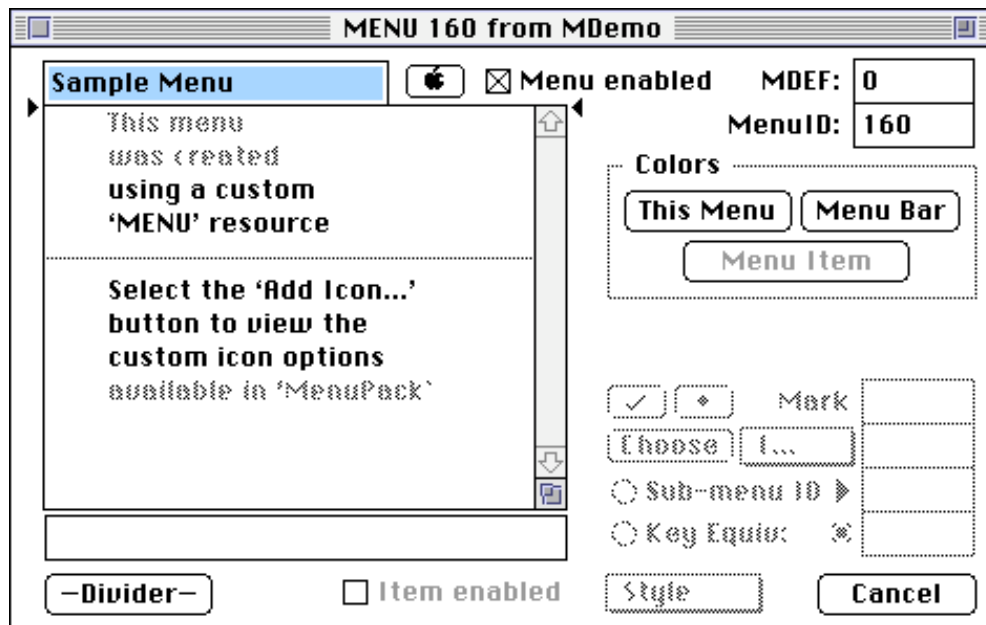
Example:

The following example will create a custom menu, using a **MENU** resource.

```
C_LONGINT(iMenuID)
```

```
iResMenu:=160
```

```
iErr:=MP_GetMenu(0;0;iResMenu)
```



MP_UpdateMenu

MP_UpdateMenu(menuID:L; itemListArray:X; menuOptions:L) -> retCode:L

Parameter	Type
menuID	C_LONGINT
itemListArray	ARRAY TEXT or ARRAY STRING
menuOptions	C_LONGINT
-> retCode	C_LONGINT

MP_UpdateMenu will replace the menu items in a existing menu with the items contained in *itemListArray*.

menuID — A valid MenuPack menuID which contains the list of items you wish to replace.

itemListArray — A valid 4th Dimension array containing the list of replacement menu items.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will demonstrate how to update an existing menu's menu item list based on a 4th Dimension array. We'll be using the menu created in the MP_ArrayMenu example as the original menu.

```
ARRAY TEXT(atNewItem;3)
```

```
atNewItem{1}:="New Item #1"
```

```
atNewItem{2}:="New Item #2"
```

```
atNewItem{3}:="New Item #3"
```

```
iErr:=MP_UpdateMenu(iMenuID;atNewItem)
```

MP_InsertMenu

MP_InsertMenu(menuID:L; itemID:L; newMenuID:L) -> retCode:L

Parameter	Type
menuID	C_LONGINT
itemID	C_LONGINT
newMenuID	C_LONGINT
-> retCode	C_LONGINT

MP_InsertMenu will insert MenuPack global menu (created using **MP_NewMenu** routine) into the menu bar or submenu (attached to an existing menu).

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

See **MP_NewMenu** example for an example of using MP_InsertMenu.

MP_PowerMenu

MP_PowerMenu(menuID:L;defItem:L;useMDEF:L) -> menuInfo:L

Parameter	Type
menuID	C_LONGINT
defItem	C_LONGINT
useMDEF	C_LONGINT
-> menuInfo	C_LONGINT

MP_PowerMenu will display a pop up menu based on the supplied menuID. This routine is similar to the 4th Dimension command Pop up menu, with a few additional features:

- You may specify any existing menu (including 4th Dimension menus or MenuPack menus).
- Pop up menu may include hierarchical menus (up to 5 sub-levels).
- Pop up menu provides complete support for all meta characters (based on creation of menus, see related menu creation routines for more information).

When using MP_PowerMenu, you must place the call in the script of an invisible button or plug-in area, as that is the only 4th Dimension object, which will execute script code while the mouse button is still down.

Using MP_PowerMenu in conjunction with **TC_RightMouseDown**, available in Toolchest, you can easily create a contextual menu (see example below).

NOTE: You could also use the routines in 4th Dimension to easily create a contextual menu. We have supplied the code for TC_RightMouseDown in the example below.

MenuID — Desired MenuPack or 4th Dimension menuID, which you wish to use as the source menu for the displayed Pop up menu.

DefItem — Passing a value greater than zero will select the supplied item when the popup is displayed. If you pass a value greater than the number of items in the menu, no item will be selected when popup menu is displayed.

useMDEF — Passing a value of one (1) will display the popup menu using Geneva 9, instead of the default Chicago 12.

NOTE: This parameter is only used on Macintosh. Passing a value of one (1) on windows will still use the default popup menu display.

-> *menuInfo* — Returns a valid menu information record (the same value returned by the 4th Dimension **Menu selected** routine), which will allow you to determine the selected menuID and menuItem.

NOTE: You should always delete any statically created menus after you have called MP_PowerMenu.

Example:

The following example will create a new menu and a submenu on the second item. We'll use the **MP_NewMenu** routine, which allows you to create a new menu without inserting it into the 4th Dimension menu bar.

```
C_LONGINT(iParent;iSub)
C_LONGINT($menuInfo;$menuID;$menuItem)

iParent:=240 `specify the desired parent menuID
iSub:=241 `specify the sub menuID

ARRAY TEXT(atParent;3)

atParent{1}:=”Menu item #1”
atParent{2}:=”Menu item #2”
atParent{3}:=”Menu item #3”

ARRAY TEXT(atSub;3)

atSub{1}:=”Subitem #1”
atSub{2}:=”Subitem #2”
atSub{3}:=”Subitem #3”

$ret:=MP_NewMenu(””;atParent;0;iParent) `create parent menu
$ret:=MP_ArrayMenu(iParent;1;””;atSub;0;iSub) `create a submenu associated to parent menu

$menuInfo:=MP_PowerMenu(iParent) `display contextual popup menu
If ($menuInfo>0) `result will be greater than zero if a menu item is selected
    $menuID:=$menuInfo%65536 `get menuID
    $menuItem:=$menuInfo\65536 `get menuItem
    $menuText:=MP_GetMenuItem($menuID;$menuItem) `get menu item text
End if

$ret:=MP_DeleteMenu(iSub) `delete submenu first
$ret:=MP_DeleteMenu(iParent) `delete parent menu
```

The following example will create a pop up menu based on an existing 4th Dimension menu or MenuPack menu located in the current menu bar.

```
$menuInfo:=MP_PowerMenu(1) `display the first menu (ie the File menu)
```

The following example will use the `TC_RightMouseDown` routine included in Toolchest to facilitate the creation of contextual menus. In addition, the popup menu will be displayed using Geneva 9 (third parameter)

```
If ( TC_RightMouseDown ) `if right mouse clicked (Win), or Control click (Mac)  
    $menuInfo:=MP_PowerMenu(1;0;MP Use MDEF) `display the first menu (ie the File menu)  
End if
```

```
`FN: TC_RightMouseDown
```

```
C_BOOLEAN($0)
```

```
C_LONGINT($mouseX;$mouseY;$mouseButton)
```

```
GET MOUSE($mouseX;$mouseY;$mouseButton)
```

```
$0:= (Macintosh control down | ($mouseButton =2))
```

MP_CountMItems

MP_CountMItems(menuID:L) -> numItems:L

Parameter	Type
menuID	C_LONGINT
-> numMenuItems	C_LONGINT

MP_CountMItems returns the number of menu items for the desired menuID.

menuID — Desired menuID which you wish to obtain the number of menu items.

-> *numMenuItems* — Returns the number of menu items for the desired menuID.

Example:

The following example will return the number of menu items created in the **MP_ArrayMenu** example.

```
$numItems:=MP_CountMItems(iMenuID)      `returns a value of 3
```

MP_DeleteMenu

MP_DeleteMenu(menuID:L) -> retCode:L

Parameter	Type
menuID	C_LONGINT
-> retCode	C_LONGINT

MP_DeleteMenu deletes the desired menuID and removes all associated menu items.

menuID — Desired menuID you wish to delete.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will demonstrate how to delete a custom menu that was created by MenuPack. We'll delete the menu created in the **MP_GetMenu** example.

```
iErr:=MP_DeleteMenu(iResMenu)
```

```
MP_DrawMBar `we need to redraw the menu bar after procedurally updating menus
```

MP_DeleteMItem

MP_DeleteMItem(menuID:L; menuItemID:L) -> retCode:L

Parameter	Type
menuID	C_LONGINT
itemID	C_LONGINT
-> retCode	C_LONGINT

MP_DeleteMItem deletes in individual menu item referenced by the menuID and itemID.

menuID — Desired menuID associated to the menuItemID you wish to delete.

itemID — Desired menuID you wish to delete.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will demonstrate how to delete a menu item from a custom menu.

Using a menu created by MenuPack routine:

```
iErr:=MP_DeleteMItem(iMenuID;2)    `delete second menu item  
MP_DrawMBar `we need to redraw the menu bar after procedurally updating menus
```

Using a menu created in the design environment:

```
iErr:=MP_DeleteMItem(1;2)    `delete second menu item in the first menu  
MP_DrawMBar `we need to redraw the menu bar after procedurally updating menus
```

MP_GetMenuItem

MP_GetMenuItem(menuID:L; itemID:L) -> menuItem:S

Parameter	Type
menuID	C_LONGINT
itemID	C_LONGINT
-> retCode	C_LONGINT

MP_GetMenuItem returns the menu item text for the desired menuID and menuItem.

menuID — Desired menuID associated to the *itemID* you wish to obtain the menu item text.

itemID — Desired itemID you wish to obtain the menu item text.

-> *menuItem* — Returns the desired menu item text.

Example:

The following example will return the menu item text.

iErr:=**MP_GetMenuItem**(iMenuID;2) `returns the item text for the second menu item

MP_GetMenuTitle

MP_GetMenuTitle(&I):S

Parameter	Type
menuID	C_LONGINT
-> menuTitle	C_STRING or C_TEXT

MP_GetMenuTitle returns the menu title text for the desired menuID.

menuID — Desired menuID for which you wish to obtain the menu title text.

-> *menuTitle* — Returns the desired menu title text.

Example:

The following example will return the menu title of a custom menu.

```
iErr:=MP_GetMenuTitle(iMenuID)    `returns the menu title
```

MP_SetItemMark

MP_SetItemMark(menuID:L; itemID:L; markCharacter:S; clearAllFlag:L) -> retCode:L

Parameter	Type
menuID	C_LONGINT
itemID	C_LONGINT
markCharacter	C_STRING
clearAllFlag	C_LONGINT
-> retCode	C_LONGINT

MP_SetItemMark set menu item character for the desired menuID and menuItem. You can optional clear all previous menuItem marks via the *clearAllFlag* parameter.

menuID — Desired menuID associated to the menuItem you wish to set the menu item mark character.

itemID — Desired itemID you wish to set the menu item mark.

markCharacter — A single character which will be used as the mark character. If you wish to clear the current mark character, pass a null string (""). On Windows, the value will be ignored and the standard Windows menu checkmark will be displayed.

clearAllFlag — Passing a value of one (1) will clear all other menuItem marks; passing a value of zero (0) will leave all existing character marks in tact.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will set the menu item's mark character of the second item in the first menu, clear all other menu item marks.

```
$ret:=MP_SetItemMark(1;2;Char(18);1)
MP_DrawMenu
```

MP_SetItemText

MP_SetItemText(menuID:L; itemID:L; itemText:S) -> retCode:L

Parameter	Type
menuID	C_LONGINT
itemID	C_LONGINT
itemText	C_STRING
-> retCode	C_LONGINT

MP_SetItemText will change the menu item text for the desired menuID and itemID.

menuID — Desired menuID associated to the menuItem you wish to set the menu item text.

itemID — Desired itemID you wish to set the menu item text.

itemText — Desired menu item text.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will set the desired menu item text.

```
$ret:=MP_SetItemText(iMenuID;iItemID;"newItemText")  
MP_DrawMenu
```

MP_SetItemIcon

MP_SetItemIcon(menuID:L; itemID:L; iconID:L; iconOption:L) -> retCode:L

Parameter	Type
menuID	C_LONGINT
itemID	C_LONGINT
iconOption	C_LONGINT
-> retCode	C_LONGINT

MP_SetItemIcon allows you to insert an icon into the desired menuID and menuItem. This routine will use the standard ICON or SICN resource.

menuID — Desired menuID associated to the itemID you wish set the menu item's icon.

itemID — Desired itemID you wish to set the item's icon.

iconOption — There are three (3) options available when setting an item's icon:

- 0 Uses the standard (32x32) icon
- 1 Uses the reduced (16x16) icon
- 2 Uses the SICN resource

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will set the icon associated to the desired menu item, using an existing .

```
$ret:=MP_SetItemIcon(1;2;15000;1)  
MP_DrawMenu
```


MP_DisableMenu

MP_DisableMenu

Parameter	Type
No Parameters	

MP_DisableMenu disables the current menu bar. If the menubar is already disabled, this routine is ignored. Use the routine **MP_EnableMenu** to re-enable the menubar.

Example:

The following example will disable the entire menu bar.

MP_DisableMenu
MP_DrawMenu

MP_EnableMenu

MP_EnableMenu

Parameter	Type
No Parameters	

MP_EnableMenu enables the current menu bar which has been previously disabled using the **MP_DisableMenu** routine. If the menubar is already enabled, this routine is ignored.

NOTE: When a menubar is re-enabled, an individual menu items which were disabled will remain disabled. This routine simply returns the menubar to the state it was when the **MP_DisableMenu** routine was called.

Example:

The following example will enable the entire menu bar.

```
MP_EnableMenu  
MP_DrawMenu
```

MP_HideMenu

MP_HideMenu

Parameter	Type
No Parameters	

MP_HideMenu hides the current menu bar.

NOTE: This routine does not function on 4th Dimension for Windows

Example:

The following example will hide the menu bar.

MP_HideMenu
MP_DrawMenu

MP_ShowMenu

MP_ShowMenu

Parameter	Type
No Parameters	

MP_ShowMenu shows the current menu bar.

NOTE: This routine does not function on 4th Dimension for Windows

Example:

The following example will show a previously hidden menu bar.

```
MP_ShowMenu  
MP_DrawMenu
```

MP DrawMBar

MP_DrawMBar

Parameter	Type
No Parameters	

Redraws the current menu bar. This routine must be called after any menu modifications, using either the 4th Dimension command or MenuPack menu management routines.

MP_GetMenuVers

MP_GetMenuVers:S

Parameter	Type
-> Extension Version	C_STRING or C_TEXT

MP_GetMenuVers returns the current version of the MenuPack external package.

Example:

The following example will return the current MenuPack version.

```
C_STRING(32;sMenuVers)
```

```
sMenuVers:=MP_GetMenuVers
```

```
If (sMenuVers#"3@")
```

```
    ALERT("Time to upgrade, you are using an outdated version of MenuPack")
```

```
End if
```

MP_Register

MP_Register(winRegCode:S{;macRegCode:S}) -> resultCode:L

Parameter	Type	Description
Windows Reg Code	C_STRING(32)	
Macintosh Reg Code	C_STRING(32)	

-> Result Code C_LONGINT

MP_Register registers your copy of MenuPack, removing the trial status that is used by default. If you don't register your copy of MenuPack, you will be presented with the trial notification dialog the first time a MenuPack routine is executed, as well a status message periodically while use the various MenuPack routines.

- If you are using MenuPack with 4D Developer edition, you may use either a developer or deployment license.
- If you are using MenuPack with 4D Server, you must have a valid deployment number, regardless of whether or not you are using a compiled database.
- If you are using MenuPack with 4D Runtime or 4D Engine, you must have valid runtime number, regardless of whether or not you are using a compiled database.

Windows Registration Number — A valid MenuPack registration number for use on Windows clients.

Macintosh Registration Number — A valid MenuPack registration number for use on Macintosh clients.

-> *Result* — A valid MenuPack result code will be returned. In the event you have registered with a valid registration number, but the incorrect environment, MenuPack will still return a value of 1 but you will see the appropriate invalid environment dialog when MenuPack is invoked.

0 – Invalid registration number

1 – Valid registration number

Example:

The following examples outline the various methods, which can be used to register your copy of MenuPack. The registration command should be executed prior to any other call to MenuPack.

C_LONGINT(\$ret)

```
$ret:=MP_Register("winRegCode";"macRegCode") `register for both platforms
```

```
$ret:=MP_Register("";"macRegCode") `register for Macintosh platform only
```

```
$ret:=MP_Register("winRegCode";"") `register for Windows platform only
```


PopupPack Routines

The following routines allow you create and manage custom popup, including complete hierarchical menu popup support.

%PopupArea

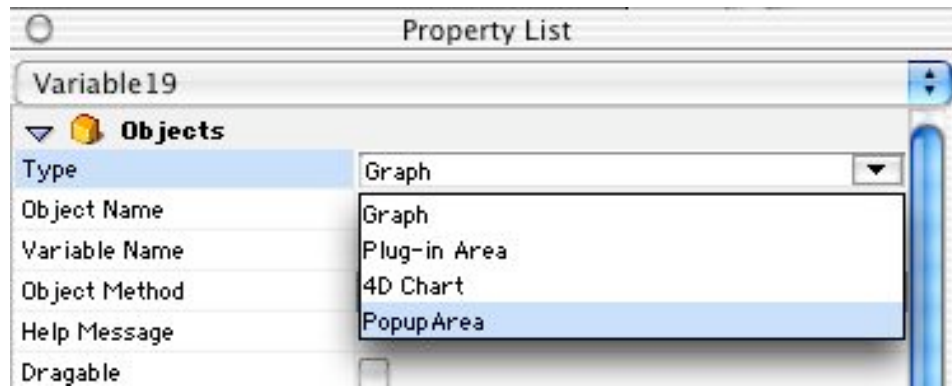
%PopupArea

Parameter	Type
No Parameters	

%PopupArea is the external area object which is placed on your layout where you wish to display a popup menu. You must create a unique popupID for each popup object on a single layout.

The value associated to this popup area is the *areaID* used by each of the follow MenuPack popup management routines.





MP_NewArea

MP_NewArea -> offscreenMenuRef:L

Parameter	Type
NO PARAMETERS	
-> menuReference	C_LONGINT

MP_NewArea will create an offscreen menu reference, which can be used with AreaList Pro to display enhanced popup menu controls in lieu of the standard popup control available in AreaList Pro.

Once the menu reference has been created, you can use any of the standard PopupPack routines (such as **MP_ArrayPop** or **MP_TextPop**) to create and manage the popup menu.

NOTE: It is your responsibility to destroy (see **MP_DestroyArea**) the menu references created by **MP_NewArea**.

-> *menuReference* — Returns a valid PopupPack offscreen menu reference.

Example:

The following example will create an offscreen menu reference.

```
C_LONGINT(iMenuRef)
```

```
iMenuRef:=MP_NewArea
```

```
ARRAY TEXT(atParent;3)
```

```
atParent{1}:= "Parent 1 "
```

```
atParent{2}:= "Parent 2 "
```

```
atParent{3}:= "Parent 3 "
```

```
iMenuID:=200
```

```
iSubmenuID:=128
```

```
$ret:=MP_ArrayPop (iMenuRef ;0;0;""; atParent;0;iMenuID)
```

```
$ret:=MP_TextPop (iMenuRef;iMenuID;1;""; "Subitem 1;Subitem 2";iSubmenuID_1)
```

```
... use the menu with ALP
```

```
$ret:=MP_DestroyArea(iMenuRef)
```

MP_DestroyArea

MP_DestroyArea(menuReference:L) -> retCode:L

Parameter	Type
menuRefernce	C_LONGINT
-> menuReference	C_LONGINT

MP_DestroyArea will destroy an offscreen menu reference created by the **MP_NewArea** routine. It is necessary for you to destroy an offscreen menu references you may have created, otherwise you will eventually run out of menu space.

menuReference — A valid offscreen menu reference created my **MP_NewArea**.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

Please see MP_NewArea example for an example of how to use MP_DestroyArea.

MP_ArrayPop

MP_ArrayPop(areaID:L ;menuID:L; itemID:L; menuItemArray:X; newMenuID:L)
-> retCode:L

Parameter	Type
areaID	C_LONGINT
menuID	C_LONGINT
itemID	C_LONGINT
menuItemArray	C_STRING or C_TEXT
newMenuID	ARRAY TEXT or ARRAY STRING
-> retCode	C_LONGINT

MP_ArrayPop will create a popup menu based, using the array as the menu list.

NOTE: This routine has changed from previous versions. Please make sure you update your code accordingly. In addition, see the routine **MP_Options** for additional information about configuring your popup areas; including font attributes, display formats, etc.

areaID — Desired **%PopupArea** object.

menuID — menuID you wish to append a hierarchical menu. If you are creating a parent menu, this value should be zero.

Passing a value of zero (0) will create a new parent menu.

Passing a valid menuID and itemID will create a hierarchical menu.

itemID — desired menuItem you wish to append the hierarchical menu. If the menuID parameter is zero or a non-valid menu, this parameter will be ignored.

menuItemArray — Desired menuItemArray (this will now be displayed in the desired location as configured in the options parameter).

newMenuID — A valid 4th Dimension array containing a list of menuItems to be used when creating popup menu.

retCode — A valid 4th Dimension variable containing a unique menuID. If the menuID you have provided already exist, PopupPack will assign a new value and update the 4D variable. You must use a 4D variable, otherwise this routine may not function properly.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will create a popup based on a 4th Dimension array.

```
ARRAY STRING(32;asPopItems;3)
asPopItems{1}:="Pop Item #1"
asPopItems{2}:="Pop Item #2"
asPopItems{3}:="Pop Item #3"
iMenuID:=220 `set the default menuID, it will be changed if menuID already exists
iMPerr:=MP_ArrayPop(popArea;0;0;";";atPopItems;iMenuID)
```

MP_TextPop

MP_TextPop(areaID:L; menuID:L; itemID:L; menuItemList:S; newMenuID:L) -> retCode:L

Parameter	Type
areaID	C_LONGINT
menuID	C_LONGINT
itemID	C_LONGINT
menuItem	C_STRING or C_TEXT
menuItemArray	ARRAY TEXT or ARRAY STRING
newMenuID	C_LONGINT
-> retCode	C_LONGINT

MP_TextPop description

areaID — Desired %**PopupArea** object.

menuID — Desired menuID associated to the itemID you wish to create popup.

Passing a value of zero (0) will create a new parent menu.

Passing a valid menuID and itemID will create a hierarchical menu.

itemID — Desired itemID which you wish to use as the parent menu for a hierarchical menu. If you pass a value of zero (0) or an item greater than the number of menu items in the desired menu will return an error code and no submenu will be created.

menuItem — Desired menuItem for popup menu. This parameter is only used when the menuID is contains a value of zero (0), meaning you are create a parent (top-level) menu.

menuItemList — A valid 4th Dimension delimited text object containing the menu item list. Each separate menu item must be separated by a semi-color (;).

```
menuItem #1;menuItem #2; menuItem #3
```

newMenuID — A valid 4th DIMESNSION variable which contains the desired menuID for the new menu. If you pass a menuID which already exists, it will be renumbered and returned in this parameter.

NOTE: It is important that you pass a variable to this routine as MenuPack may change this value and return it back in this parameter.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will create a popup based on a 4th Dimension array.

```
C_TEXT(tPopItems)
```

```
tPopItems:="Pop Item #1;Pop Item #2;Pot Item #3"
```

```
iMenuID:=220 `set the default menuID, it will be changed if menuID already exists
```

```
iMPerr:=MP_TextPop(popArea;0;0;"";tPopItems;iMenuID)
```

MP_NewGlobal

MP_NewGlobal(menuTitle:S; itemList:X; metaOption:L; newMenuID:L) -> retCode:L

Parameter	Type
menuTitle	C_STRING or C_TEXT
menuItemList	ARRAY TEXT or ARRAY STRING
metaOption	C_LONGINT
newMenuID	C_LONGINT

MP_NewGlobal will create a new global menu which can be inserted into a popup control using the **MP_InsertGlobal** routine.

menuTitle — Desired menu title.

itemList — A valid 4th Dimension array containing the menuItems for new global menu.

metaOption — Meta character option. Passing a value of 1 will create the menu with meta-characters disabled (meaning you can include any meta-character in your itemList); passing a value of zero (0) will treat any meta-character in your item list as a valid meta-character. See example below for more information.

newMenuID — A valid 4th DIMENSION variable which contains the desired menuID for the new menu. If you pass a menuID which already exists, it will be renumbered and returned in this parameter.

Example:

The following example will create a global menu which will be inserted into a popup control in the **MP_InsertGlobal** example.

```
ARRAY STRING(32;asPopItems;3)
asPopItems{1}:= "Pop Item #1 "
asPopItems{2}:= "Pop Item #2 "
asPopItems{3}:= "Pop Item #3 "
iGlobalMenu:=222 `set the default menuID, it will be changed if menuID already exists
iMPerr:=MP_NewGlobal("";asPopItems;0;iGlobalMenu)
```

MP_InsertGlobal

MP_InsertGlobal(areaID:L; menuID:L; itemID:L; globalMenuID:L) -> retCode:L

Parameter	Type
areaID	C_LONGINT
menuID	C_LONGINT
itemID	C_LONGINT
globalMenuID	C_LONGINT
-> retCode	C_LONGINT

MP_InsertGlobal inserts a global menu (created using the **MP_NewGlobal**) routine in the desired popup menu control. This routine is useful for creating menus which the items don't change, such as a list of files.

areaID — Desired **%PopupArea** object.

menuID — menuID you wish to append a hierarchical menu. If you are creating a parent menu, this value should be zero.

Passing a value of zero (0) will create a new parent menu.

Passing a valid menuID and itemID will create a hierarchical menu.

itemID — desired menuItem you wish to append the hierarchical menu. If the menuID parameter is zero or a non-valid menu, this parameter will be ignored.

globalMenuID— The desired global menuID you wish to insert.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will insert the global menu created in the **MP_NewGlobal** example. In this example, we'll be creating a hierarchical menu which will be attached to the first menu item (menu created in MP_ArrayPop routine).

```
iMPerr:=MP_InsertGlobal(popArea;iMenuID;1;iGlobalMenu)    `add to first menu item
```

MP_DeletePop

MP_DeletePop(areaID:L; menuID:L) -> retCode:L

Parameter	Type
areaID	C_LONGINT
menuID	C_LONGINT
-> retCode	C_LONGINT

MP_DeletePop will delete the desired menuID (and all submenus) for the defined popup area.

areaID — Desired **%PopupArea** object.

menuID — Desired menuID you wish to delete for the desired popup object.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will demonstrate how to delete a menu from a popup control.

```
iMPerr:=MP_DeletePop(popArea;iMenuID)
```

MP_DeleteGlobal

MP_DeleteGlobal(globalMenuID:L) -> retCode:L

Parameter	Type
Global Menu ID	C_LONGINT
-> retCode	C_LONGINT

MP_DeleteGlobal will delete a global menu from memory.

WARNING: Don't call this routine if you the global menu is in use.

Example:

The following example will demonstrate how to delete a menu from a popup control.

globalMenuID — Global menuID you wish to delete.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will delete the global menu created in the **MP_NewGlobal** example.

```
iMPerr:=MP_DeleteGlobal(iGlobalMenu)
```

MP_GlobalList

MP_GlobalList(menuIDArray:X; menuTitleArray:X) -> retCode:L

Parameter	Type
menuIDArray	ARRAY LONGINT
menuTitleArray	ARRAY TEXT
-> retCode	C_LONGINT

MP_GlobalList will build an array of all global menus.

menuID — A valid 4th Dimension array which will receive a list of all global menuID's.

menuTitle — A valid 4th Dimension array which will receive a list of all global menu title's.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will build an array of all global menuID's and menuTitle's.

```
ARRAY LONGINT(aiMenuID;0)  
ARRAY STRING(32;asMenuTitle;0)  
iMPerr:=MP_GlobalList(aiMenuID:asMenuTitle)
```

MP_EnbPopItem

MP_EnbPopItem(areaID:L; menuID:L; itemID:L; enableFlag:L):L

Parameter	Type
areaID	C_LONGINT
menuID	C_LONGINT
itemID	C_LONGINT
enableFlag	C_LONGINT
-> retCode	C_LONGINT

MP_EnbPopItem will set the enabled/disabled state for the desired menu item in a popup area.

areaID — Desired **%PopupArea** object.

menuID — Desired menuID you wish to delete for the desired popup object.

itemID — Desired itemID you wish to set the enabled status for popup item.

enableFlag — Passing a value of one (1) will enable the desired menu item; passing a value of zero (0) will disable the desired menu item.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will disable the first menu item.

```
iMPerr:=MP_EnbPopItem(popArea;iMenuID;1;0) `disable first item  
iMPerr:=MP_DrawPop(popArea) `redraw area
```

MP_GetPopItem

MP_GetPopItem(areaID:L; menuTitle:S; itemText:S; menuID:L; itemID:L) -> retCode:L

Parameter	Type
areaID	C_LONGINT
menuTitle	C_STRING or C_TEXT
itemText	C_STRING or C_TEXT
menuID	C_LONGINT
itemID	C_LONGINT
-> retCode	C_LONGINT

MP_GetPopItem will return the currently selected object. All parameters (except the areaID) must be valid 4th DIMENSION process or interprocess variables (don't use local variables) as each parameter will return the desired value into the supplied variable.

areaID — Desired %**PopupArea** object.

menuTitle — Returns the selected menu's title. If this is a submenu, the parent menu item's text will be returned.

itemText — Returns the selected menu item text.

menuID — MenuID associated to the current menu item.

itemID — ItemID associated to the current menu item.

-> *selectedItem* — Returns the selected item for the menu. This is the same value which is returned when using a standard 4th Dimension popup, however, it works with all hierarchical menus as well as the parent menu.

Possible Return Values

- 0 No item was selected (the user moved off the popup control)
- >1 Selected item
- 5 If a disabled popup control is selected

Example:

The following example demonstrates how to obtain the current item in a popup control. This could be typically placed in the popup objects script

```
If (During)
  iRetCode:=MP_GetPopItem(sMenuTitle;sMenuItem;iSelMenu;iSelItem)
  If (iRetCode>0)      `we selected an item
    ALERT("You selected "+sMenuItem+".")
  End if
```

End if

MP_SetPopText

MP_SetPopText(areaID:L; menuID:L; itemID:L; itemText:S) -> retCode:L

Parameter	Type
areaID	C_LONGINT
menuID	C_LONGINT
itemID	C_LONGINT
itemText	C_STRING
-> retCode	C_LONGINT

MP_SetPopText will set the item text for the desired menu item in a popup area.

areaID — Desired **%PopupArea** object.

menuID — Desired menuID you wish to set the item text.

itemID — Desired itemID you wish to set item text.

itemText — Item text for desired menu item.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will change the popup menu item's text.

```
iMPerr:=MP_SetPopText(popArea;iMenuID;1;"New Popup Item")  
iMPerr:=MP_DrawPop(popArea)
```

MP_SetPopMark

MP_SetPopMark(areaID:L; menuID:L; itemID:L; itemMark:S; clearAll:L) -> retCode:L

Parameter	Type
areaID	C_LONGINT
menuID	C_LONGINT
itemID	C_LONGINT
itemMark	C_STRING
-> retCode	C_LONGINT

MP_SetPopMark will set the item mark for the desired menu item in a popup area.

areaID — Desired %**PopupArea** object.

menuID — Desired menuID you wish to set the item mark.

itemID — Desired itemID you wish to set item mark.

itemText — Item mark for desired menu item.

clearAll — Passing a value of one (1) will clear all existing marks for desired menuID; passing a value of zero (0) will leave all existing marks in tact.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will set the item mark for the desired popup menu item.

```
iMPerr:=MP_SetPopMark(popArea;iMenuID;1;Char(18);1)    `sets the mark character the checkmark  
iMPerr:=MP_DrawPop(popArea)
```

MP_SetPopItem

MP_SetPopItem(areaID:L; menuID:L; itemID:L) -> retCode:L

Parameter	Type
areaID	C_LONGINT
menuID	C_LONGINT
itemID	C_LONGINT
-> retCode	C_LONGINT

MP_SetPopItem will set the current item in a popup area. This will not change the current popup mark, you must call **MP_SetPopMark** to change the mark character.

areaID — Desired **%PopupArea** object.

menuID — Desired menuID you wish to set as the current menu item.

itemID — Desired itemID you wish to set as the current item.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will change the current popup item.

```
iMPerr:=MP_SetPopItem(popArea;iMenuID;1)
iMPerr:=MP_SetPopMark(popArea;iMenuID;1;Char(18);1)    `update mark
iMPerr:=MP_DrawPop(popArea)
```

MP_SetPopIcon

MP_SetPopIcon(areaID:L; menuID:L; itemID:L; iconID:L) -> retCode:L

Parameter	Type
areaID	C_LONGINT
menuID	C_LONGINT
itemID	C_LONGINT
iconID	C_LONGINT
-> retCode	C_LONGINT

MP_SetPopIcon will set the icon which will be displayed for the desired menuID and itemID.

If you need to create menu items which contain more than 9 icons, you will need to use the **MP_SetPopIcon** routine as the standard meta-character icon feature only supports 9 icons.

areaID — Desired **%PopupArea** object.

menuID — Desired menuID you wish to set the item icon.

itemID — Desired itemID you wish to set the item icon.

iconID — Desired iconID you wish to display with the associated menu item. The following icon types can be used:

cicn
ICON

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will associate an icon to the desired menu item.

```
iResID:=15000  `use resID 15000  
iMPerr:=MP_SetPopIcon(popArea;iMenuID;1;iResID)  
iMPerr:=MP_DrawPop(popArea)
```

MP_SetPopPICT

MP_SetPopPICT(areaID:L; resID:L; resName:S) -> retCode:L

Parameter	Type
areaID	C_LONGINT
resID	C_LONGINT
resName	C_STRING
-> retCode	C_LONGINT

MP_SetPopPict will set the PICT resource which will be drawn instead of the standard popup graphic for the desired popup control.

areaID — Desired **%PopupArea** object.

resID — Desired **PICT** resource ID you wish to display. If the specified resource ID is not found, the *resourceName* parameter will be used.

itemID — Desired **PICT** resource name you wish to display.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will configure the desired popup area to display a PICT resource instead of drawing the standard popup graphic. This example will read the PICT resource based on the *resName* as we have defined the *resID* as zero (0).

```
iResID:=0  
sResName:="myPicture"  
iMPerr:=MP_SetPopPict(popArea;iResID;sResName)  
iMPerr:=MP_DrawPop(popArea)
```

MP_GetPopMark

MP_GetPopMark(areaID:L; menuID:L; itemID:L) -> popMarkCharacter:L

Parameter	Type
areaID	C_LONGINT
menuID	C_LONGINT
itemID	C_LONGINT
enableFlag	C_LONGINT
-> retCode	C_LONGINT

MP_GetPopMark will return the ASCII value of the item mark for the desired menu item.

areaID — Desired **%PopupArea** object.

menuID — Desired menuID you wish to obtain the current popup mark.

itemID — Desired item you wish to obtain the current popup mark.

-> *markCharacter* — Returns the ASCII value of the item mark. If the desired menu item is a parent menu to a submenu, a value of 100 will be returned.

Example:

The following example will return the mark character for the desired menu item. In this case, a value of 100 will be returned, signifying that we have chosen a menu item which has a submenu attached.

```
iParMenu:=220
iSubMenu:=221
iMPerr:=MP_ArrayPop(popArea;"";0;0;atParent;iParMenu)
iMPerr:=MP_ArrayPop(popArea;"";iParMenu;1;atSubItem)
iMPmark:=MP_GetPopMark(popArea;iParMenu;1)    `returns a value of 100 as this is a submenu
```

MP_AreaMenuID

MP_AreaMenuID(areaID:L; parentMenu:L) -> retCode:L

Parameter	Type
areaID	C_LONGINT
parentMenu	C_LONGINT
-> retCode	C_LONGINT

MP_AreaMenuID returns the parent menuID for the desired popup area. This routine is a useful routine when writing modular routines for managing popup controls.

areaID — Desired **%PopupArea** object.

parentMenu — A valid 4th Dimension variable which will receive the parent menuID for the desired popup control.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example demonstrates how to use the MP_AreaMenuID routine.

C_LONGINT(iParMenu)

```
iParMenu:=220
iMPerr:=MP_ArrayPop(popArea;"";0;0;atItems;iParMenu)
iMPerr:=MP_AreaMenuID(popArea;iMenuID)      `get the parent menuID
iMPerr:=MP_DeletePop(popArea;iMenuID)     `delete entire popup
```

MP_HideArea

MP_HideArea(areaID:L) -> retCode:L

Parameter	Type
areaID	C_LONGINT
-> retCode	C_LONGINT

MP_HideArea will hide the desired popup area. You can use the routine **MP_ShowArea** to display a hidden popup area.

areaID — Desired **%PopupArea** object.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will hide a popup control.

```
iMPerr:=MP_HideArea(popArea)
```

MP_ShowArea

MP_ShowArea(areaID:L) -> retCode:L

Parameter	Type
areaID	C_LONGINT
-> retCode	C_LONGINT

MP_ShowArea will show a previously hidden area using the **MP_HideArea** routine.

areaID — Desired **%PopupArea** object.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will show the popup control hidden in the MP_HideArea example.

```
iMPerr:=MP_ShowArea(popArea)  
iMPerr:=MP_DrawPop(popArea)
```

MP_DisablePopup

MP_DisablePopup(areaID:L) -> retCode:L

Parameter	Type
areaID	C_LONGINT
-> retCode	C_LONGINT

MP_DisablePopup will disable a popup area. You can use the routine **MP_EnablePopup** to enable a disabled popup area.

TIP: If you wish to disable the popup control when the layout is loaded, you can use the **MP_SetPopOpts** routine (value 4) or the **MP_DisablePop** routine.

areaID — Desired **%PopupArea** object.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will disable the entire popup control.

```
iMPerr:=MP_DisablePopup(popArea)
```

MP_EnablePopup

MP_EnablePopup(areaID:L) -> retCode:L

Parameter	Type
areaID	C_LONGINT
-> retCode	C_LONGINT

MP_EnablePopup will enable a previously disabled popup area using the **MP_DisablePopup** routine.

areaID — Desired **%PopupArea** object.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will enable the popup disabled in the **MP_DisablePopup** example.

```
iMPerr:=MP_EnablePopup(popArea)
```

MP_DrawPop

MP_DrawPop(areaID:L) -> retCode:L

Parameter	Type
areaID	C_LONGINT
-> retCode	C_LONGINT

MP_DrawPop will force the desired popup area to be redrawn. This should be called anytime you modify a popup control in the during phase.

areaID — Desired **%PopupArea** object.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

MP_SetPopOpts

MP_SetPopOpts(areaID:L; titleOptions:L; setupOptions:L):L

Parameter	Type
areaID	C_LONGINT
titleOptions	C_LONGINT
setupOptions	C_LONGINT
-> retCode	C_LONGINT

MP_SetPopOpts allows you to configure the setup parameters for a popup menu. This routine should be called before calling **MP_ArrayPop** (or related routine) to initialize the popup area.

Should you need to re-initialize a popup menu or would like to change the popup format (ie change it from standard to 3D interface), you'll need to delete the existing popup (using **MP_DeletePop**) and re-create the popup control.

areaID — Desired **%PopupArea** object.

titleOptions — Configures the title options for a popup object. The following values are available when a popup control is created:

Option	Description
---------------	--------------------

0	Popup Title Left Justified
1	Popup Title Centered
255	Popup Title Right Justified

setupOptions — Configures the setup options for a popup object. You can use the MenuPack Popup Constants defined in the Appendix to provide easier identification when using multiple setup options.

The following values are available when a popup control is created:

NOTE: To configure multiple parameters when setting the default options, you can pass the sum of the parameters you wish to activate. For example, to configure a popup to drawn the size of the external area (value 1) and display in the 3D interface (value 8), pass a value of nine (1+8=9) as the setupOption.

<u>Option</u>	<u>Description</u>
1	Popup Drawn Size of External Area <i>If this is not set, the popup size will be determined based on popup items.</i>
2	Use small arrow when displaying popup. In addition, if you have the Auto Check value set, the default checkmark will be a bullet.
4	Disable standard meta-characters <i>Use this option when you wish to display "/" or other meta-characters in your popup menu.</i>
8	Disable popup using 3D Interface
16	Display arrow only, no text will be drawn
32	Auto Check selected item.
64	Configure popup to be disabled when initially displayed. <i>Use the routine MP_EnablePop to enable a disabled popup</i>
128	Configure popup to be hidden when initially displayed <i>Use the routine MP_ShowPop to show a hidden popup</i>

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will configure a popup to contain these settings:

- Draw popup size of external area (value 1)
- Use small arrow (value 2)
- Display 3D popup (value 8)
- Auto check selected item (value 32)

`$Err:=MP_SetPopOpts (eEnable;0;MP_Popup_Fixed+MP_Popup_Small+MP_Popup_Use3D+MP_Popup_AutoCheck)`

MP_GetPopOpts

MP_GetPopOpts(areaID:L; titleOptions:L; setupOptions:L) -> retCode:L

Parameter	Type
areaID	C_LONGINT
titleOptions	C_LONGINT
setupOptions	C_LONGINT
-> retCode	C_LONGINT

MP_GetPopOpts will obtain the current options for the desired popup area.

areaID — Desired **%PopupArea** object.

titleOptions — A valid 4th Dimension variable which will receive the titleOptions.

setupOptions — A valid 4th Dimension variable which will receive the setupOptions.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will obtain the current options for the desired popup control. The values set in the **MP_SetPopOpts** will be returned.

C_LONGINT(iTitOpts;iSetOpts)

```
iTitOpts:=0  
iSetOpts:=0  
iMPerr:=MP_GetPopOpts(popArea;iTitOpts;iSetOpts)
```

```
iTitOpts      equals      0  
iSetOpts      equals      43 ( 1 + 2 + 8 + 32 )
```

MP_SetPopFont

MP_SetPopFont(areaID:L; fontName:S; fontSize:L; fontStyle:) -> retCode:L

Parameter	Type
areaID	C_LONGINT
fontName	C_STRING
fontSize	C_LONGINT
fontStyle	C_LONGINT
-> retCode	C_LONGINT

MP_SetPopFont will set the font attributes for the desired menu item in a popup area.

areaID — Desired %**PopupArea** object.

fontName — Font name for desired popup control,

fontSize — Font size for desired popup contro.

fontStyle — Font style for desired popup control.

<u>Attribute</u>	<u>Description</u>
------------------	--------------------

0	Plain
1	Bold
2	<i>Italic</i>
4	<u>Underline</u>
8	Outline
16	Shadow
32	Condensed
64	E x t e n d e d

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will set the font attributes for the desired popup control.

```
iMPerr:=MP_SetPopFont(popArea;"Times";12;1)    `set font attributes times, 12 point, bold
```

MP_GetPopFont

MP_GetPopFont(areaID:L; fontName:S; fontSize:L; fontStyle:L) -> retCode:L

Parameter	Type
areaID	C_LONGINT
fontName	C_STRING
fontSize	C_LONGINT
fontStyle	C_LONGINT
-> retCode	C_LONGINT

MP_GetPopFont will return the font attributes for the desired menu item in a popup area.

areaID — Desired %**PopupArea** object.

fontName — A valid 4th Dimension variable which will receive the current font name.

fontSize — A valid 4th Dimension variable which will receive the current font size.

fontStyle — A valid 4th Dimension variable which will receive the current font style.

-> *retCode* — Returns a valid MenuPack return code. For complete details about all return codes in MenuPack, please refer to the **MenuPack Return Codes** chapter.

Example:

The following example will return the font attributes for the desired popup control. The value set in the **MP_SetPopFont** example will be returned.

```
C_STRING(32;sFontName)
C_LONGINT(iFontSize;iFontStyle)
```

```
sFontName:=""
iFontSize:=0
iFontStyle:=0
```

```
iMPerr:=MP_SetPopFont(popArea;sFontName;iFontSize;iFontStyle)
```

```
sFontName      equals      Times
iFontSize      equals      12
iFontStyle     equals      1
```

MP_GetPopVers

MP_GetPopVers -> versionString:S

Parameter	Type
-> Popup Plug-In Version	C_STRING or C_TEXT

MP_GetPopVers returns the current version of the MenuPack popup external package.

Example:

The following example returns the current version.

```
C_STRING(32;sVers)
```

```
sVers:=""
```

```
sVers:=MP_GetPopVers
```

```
If ( sVers#"3@")
```

```
    ALERT("Time to upgrade!")
```

```
End if
```

MP_RegisterPop

MP_RegisterPop(winRegCode:S{;macRegCode:S}) -> resultCode:L

Parameter	Type	Description
Windows Reg Code	C_STRING(32)	
Macintosh Reg Code	C_STRING(32)	
-> Result Code	C_LONGINT	

MP_RegisterPop registers your copy of PopupPack, removing the trial status that is used by default. If you don't register your copy of PopupPack, you will be presented with the trial notification dialog the first time a PopupPack routine is executed, as well a status message periodically while use the various PopupPack routines.

- If you hare using PopupPack with 4D Developer edition, you may use either a developer or deployment license.
- If you are using PopupPack with 4D Server, you must have a valid deployment number, regardless of whether or not you are using a compiled database.
- If you hare using PopupPack with 4D Runtime or 4D Engine, you must have valid runtime number, regardless of whether or not you are using a compile database.

Windows Registration Number — A valid PopupPack registration number for use on Windows clients.

Macintosh Registration Number — A valid PopupPack registration number for use on Macintosh clients.

-> *Result* — A valid PopupPack result code will be returned. In the event you have registered with a valid registration number, but the incorrect environment, PopupPack will still return a value of 1 but you will see the appropriate invalid environment dialog when PopupPack is invoked.

0 – Invalid registration number

1 – Valid registration number

Example:

The following examples outline the various methods, which can be used to register your copy of PopupPack. The registration command should be executed prior to any other call to PopupPack.

C_LONGINT(\$ret)

```
$ret:=MP_RegistePopr("winRegCode";"macRegCode") `register for both platforms
```

```
$ret:=MP_RegisterPop("", "macRegCode") `register for Macintosh platform only
```

```
$ret:=MP_RegisterPop("winRegCode";"") `register for Windows platform only
```


5 — MenuPack Constants and Error Codes

This chapter outlines constant values (used for easy parameter configuration) and all error codes which can be returned by MenuPack creating and managing custom menus and popups.

MenuPack Constants

MenuPack contains a set of custom 4th DIMESNION Constants (required 4D v6) which can be used when creating MenuPack Popup menus (see DD_DDDD routine).

NOTE: Make sure you have called the **mpSTARTUP** routine before calling any MenuPack external routine.

<u>Constant Name</u>	<u>Value</u>	<u>Description</u>
MP Popup Fixed	1	Display popup menu based on size of external area.
MP Popup Small	2	Use small arrow when displaying popup
MP Popup NoMeta	4	Disable meta characters
MP Popup Use3D	8	Display popup using 3D display
MP Popup Arrow Only	16	Display popup with arrow only, no text is displayed
MP Popup Auto Check	32	Auto check selected menu item
MP Popup Disable	64	Disable popup menu created
MP Popup Hidden	128	Hide popup menu when created
MP Use MDEF	1	Use Geneva 9 when using MP_PowerMenu

MenuPack Error Codes

MenuPack will return a variety of error codes when creating custom menus and popups. The following outlines all the possible error codes which may be returned by MenuPack.

<u>Constant Name</u>	<u>Value</u>	<u>Description</u>
MP No Error	0	No error
MP Invalid MenuID	-1	Invalid menuID

Appendix - Technical Support

This appendix lists the various places where you can obtain technical support for MenuPack and supporting components. Automated Solutions Group provides free technical support for all registered users via electronic mail, telephone or standard mail. Primary support is offered via electronic mail.

Electronic Mail

support@asgsoft.com US Customers

World Wide Web

<http://www.asgsoft.com>
<ftp://ftp.asgsoft.com>

Our WWW server provides a variety of additional information, including FAQ's (Frequently Asked Questions) and our exclusive Technical Notes as well as demo versions of other Automated Solutions Group products.

Telephone

In you are unable to reach us via electronic mail, you can reach our telephone technical support representatives at **(714) 375-4257**. We recommend you have sample code or screen shots ready in the event our support representative needs additional information to better assist you.

Fax

You can also obtain technical support by faxing us your problem at **(714) 848-0382**.

Mailing Address

As a last resort you can send technical questions via standard US Mail using the following address:

Automated Solutions Group / Technical Support (US and Canada)
16742 Gothard Street, Suite 210
Huntington Beach, CA 92647 US

Index

MenuPack Routines – Alphabetical

MP_ArrayMenu	19
MP_CountItems	31
MP_DeleteMenu	32
MP_DeleteItem	33
MP_DisableMenu	40
MP_DrawMBar	44
MP_EnableItem	39
MP_EnableMenu	41
MP_GetMenu	24
MP_GetMenuItem	34
MP_GetMenuTitle	35
MP_GetMenuVers	45
MP_HideMenu	42
MP_InsertMenu	27
MP_NewMenu	18
MP_NewResMenu	23
MP_PowerMenu	28
MP_Register	46
MP_SetItemIcon	38
MP_SetItemMark	36
MP_SetItemText	37
MP_ShowMenu	43
MP_TextMenu	21
MP_UpdateMenu	26

PopupPack Routines – Alphabetical

%PopupArea	47
MP_AreaMenuID	63
MP_ArrayPop	48
MP_DeleteGlobal.....	53
MP_DeletePop	52
MP_DisablePop	66
MP_DrawPop.....	68
MP_EnablePop	67
MP_EnbPopItem	55
MP_GetPopFont	73
MP_GetPopItem	56
MP_GetPopMark	62
MP_GetPopOpts	71
MP_GetPopVers	74
MP_GlobalList	54
MP_HideArea.....	64
MP_InsertGlobal	51
MP_NewGlobal.....	50
MP_RegisterPop.....	75
MP_SetPopFont.....	72
MP_SetPopIcon.....	60
MP_SetPopItem	59
MP_SetPopMark	58
MP_SetPopOpts.....	69
MP_SetPopPict.....	61
MP_SetPopText.....	57
MP_ShowArea	65
MP_TextPop.....	49